## Exploring Analytic Geometry with Mathematica®

#### LIMITED WARRANTY AND DISCLAIMER OF LIABILITY

ACADEMIC PRESS ("AP") AND ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION OR PRODUCTION OF THE ACCOMPANYING CODE ("THE PRODUCT") CANNOT AND DO NOT WAR-RANT THE PERFORMANCE OR RESULTS THAT MAY BE OBTAINED BY USING THE PRODUCT. THE PRODUCT IS SOLD "AS IS" WITHOUT WARRANTY OF ANY KIND (EXCEPT AS HEREAFTER DESCRIBED), EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY WAR-RANTY OF PERFORMANCE OR ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. AP WARRANTS ONLY THAT THE CD-ROM ON WHICH THE CODE IS RECORDED IS FREE FROM DEFECTS IN MATERIAL AND FAULTY WORKMANSHIP UN-DER THE NORMAL USE AND SERVICE FOR A PERIOD OF NINETY (90) DAYS FROM THE DATE THE PRODUCT IS DELIVERED. THE PURCHASER'S SOLE AND EXCLUSIVE REMEDY IN THE EVENT OF A DEFECT IS EXPRESSLY LIMITED TO EITHER REPLACEMENT OF THE CD-ROM OR REFUND OF THE PURCHASE PRICE, AT AP'S SOLE DISCRETION.

IN NO EVENT, WHETHER AS A RESULT OF BREACH OF CONTRACT, WARRANTY, OR TORT (INCLUDING NEGLIGENCE), WILL AP OR ANYONE WHO HAS BEEN INVOLVED IN THE CREATION OR PRODUCTION OF THE PRODUCT BE LIABLE TO PURCHASER FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF INABILITY TO USE THE PRODUCT OR ANY MODIFICA-TIONS THEREOF, OR DUE TO THE CONTENTS OF THE CODE, EVEN IF AP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

Any request for replacement of a defective CD-ROM must be postage prepaid and must be accompanied by the original defective CD-ROM, your mailing address and telephone number, and proof of date of purchase and purchase price. Send such requests, stating the nature of the problem, to Academic Press Customer Service, 6277 Sea Harbor Drive, Orlando, FL 32887, 1-800-321-5068. AP shall have no obligation to refund the purchase price or to replace a CD-ROM based on claims of defects in the nature or operation of the Product.

Some states do not allow limitation on how long an implied warranty lasts, nor exclusions or limitations of incidental or consequential damages, so the above limitations and exclusions may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from jurisdiction to jurisdiction.

THE RE-EXPORT OF UNITED STATES ORIGIN SOFTWARE IS SUBJECT TO UNITED STATES LAWS UNDER THE EXPORT ADMINISTRATION ACT OF 1969 AS AMENDED. ANY FURTHER SALE OF THE PRODUCT SHALL BE IN COMPLIANCE WITH THE UNITED STATES DEPARTMENT OF COM-MERCE ADMINISTRATION REGULATIONS. COMPLIANCE WITH SUCH REGULATIONS IS YOUR RESPONSIBILITY AND NOT THE RESPONSIBILITY OF AP.

Mathematica and MathReader are registered trademarks of Wolfram Research, Inc. Acrobat Reader is a registered trademark of Adobe Systems, Inc.

### Exploring Analytic Geometry with Mathematica®

### Donald L. Vossler

BME, Kettering University, 1978 MM, Aquinas College, 1981



#### ACADEMIC PRESS

San Diego London Boston New York Sydney Tokyo Toronto

### Preface

The study of two-dimensional analytic geometry has gone in and out of fashion several times over the past century, however this classic field of mathematics has once again become popular due to the growing power of personal computers and the availability of powerful mathematical software systems, such as *Mathematica*, that can provide an interactive environment for studying the field. By combining the power of *Mathematica* with an analytic geometry software system called *Descarta2D*, the author has succeeded in meshing an ancient field of study with modern computational tools, the result being a simple, yet powerful, approach to studying analytic geometry. Students, engineers and mathematicians alike who are interested in analytic geometry can use this book and software for the study, research or just plain enjoyment of analytic geometry.

Mathematica provides an attractive environment for studying analytic geometry. Mathematica supports both numeric and symbolic computations, meaning that geometry problems can be solved numerically, producing approximate or exact answers, as well as producing general formulas with variables. Mathematica also has good facilities for producing graphical plots which are useful for visualizing the graphs of two-dimensional geometry.

#### Features

*Exploring Analytic Geometry with Mathematica*, Mathematica and Descarta<sub>2D</sub> provide the following outstanding features:

- The book can serve as classical analytic geometry textbook with in-line *Mathematica* dialogs to illustrate key concepts.
- A large number of examples with solutions and graphics is keyed to the textual development of each topic.
- Hints are provided for improving the reader's use and understanding of *Mathematica* and *Descarta2D*.
- More advanced topics are covered in explorations provided with each chapter, and full solutions are illustrated using *Mathematica*.

- A detailed reference manual provides complete documentation for *Descarta2D*, with complete syntax for over 100 new commands.
- Complete source code for *Descarta2D* is provided in 30 well-documented *Mathematica* notebooks.
- The complete book is integrated into the *Mathematica* Help Browser for easy access and reading.
- A CD-ROM is included for convenient, permanent storage of the Descarta2D software.
- A complete software system and mathematical reference is packaged as an affordable book.

#### **Classical Analytic Geometry**

*Exploring Analytic Geometry with Mathematica* begins with a traditional development of analytic geometry that has been modernized with in-line chapter dialogs using *Descarta2D* and *Mathematica* to illustrate the underlying concepts. The following topics are covered in 21 chapters:

Coordinates • Points • Equations • Graphs • Lines • Line Segments • Circles • Arcs • Triangles • Parabolas • Ellipses • Hyperbolas • General Conics • Conic Arcs • Medial Curves • Transformations • Arc Length • Area • Tangent Lines • Tangent Circles • Tangent Conics • Biarcs.

Each chapter begins with definitions of underlying mathematical terminology and develops the topic with more detailed derivations and proofs of important concepts.

#### **Explorations**

Each chapter in *Exploring Analytic Geometry with Mathematica* concludes with more advanced topics in the form of exploration problems to more fully develop the topics presented in each chapter. There are more than 100 of these more challenging explorations, and the full solutions are provided on the CD-ROM as *Mathematica* notebooks as well as printed in Part VIII of the book. Sample explorations include some of the more famous theorems from analytic geometry:

Carlyle's Circle • Castillon's Problem • Euler's Triangle Formula • Eyeball Theorem • Gergonne's Point • Heron's Formula • Inversion • Monge's Theorem • Reciprocal Polars • Reflection in a Point • Stewart's Theorem • plus many more.

#### Descarta2D

Descarta2D provides a full-scale Mathematica implementation of the concepts developed in *Exploring Analytic Geometry with Mathematica*. A reference manual section explains in detail the usage of over 100 new commands that are provided by *Descarta2D* for creating, manipulating and querying geometric objects in *Mathematica*. To support the study and enhancement of the *Descarta2D* algorithms, the complete source code for *Descarta2D* is provided, both in printed form in the book and as *Mathematica* notebook files on the CD-ROM.

#### CD-ROM

The CD-ROM provides the complete text of the book in Abode Portable Document Format (PDF) for interactive reading. In addition, the CD-ROM provides the following *Mathematica* notebooks:

- Chapters with Mathematica dialogs, 24 interactive notebooks
- Reference material for Descarta2D, three notebooks
- Complete Descarta2D source code, 30 notebooks
- Descarta2D packages, 30 loadable files
- Exploration solutions, 125 notebooks.

These notebooks have been thoroughly tested and are compatible with *Mathematica* Version 3.0.1 and Version 4.0. Maximum benefit of the book and software is gained by using it in conjunction with *Mathematica*, but a passive reading and viewing of the book and notebook files can be accomplished without using *Mathematica* itself.

#### Organization of the Book

Exploring Analytic Geometry with Mathematica is a 900-page volume divided into nine parts:

- Introduction (Getting Started and Descarta2D Tour)
- Elementary Geometry (Points, Lines, Circles, Arcs, Triangles)
- Conics (Parabolas, Ellipses, Hyperbolas, Conics, Medial Curves)
- Geometric Functions (Transformations, Arc Length, Area)
- Tangent Curves (Lines, Circles, Conics, Biarcs)
- Descarta2D Reference (philosophy and command descriptions)
- Descarta2D Packages (complete source code)

- Explorations (solution notebooks)
- Epilogue (Installation Instructions, Bibliography and a detailed index).

#### About the Author

Donald L. Vossler is a mechanical engineer and computer software designer with more than 20 years experience in computer aided design and geometric modeling. He has been involved in solid modeling since its inception in the early 1980's and has contributed to the theoretical foundation of the subject through several published papers. He has managed the development of a number of commercial computer aided design systems and holds a US Patent involving the underlying data representations of geometric models.

# Contents

| Ι        | Int          | troduction            | 1  |
|----------|--------------|-----------------------|----|
| 1        | Get          | ting Started          | 3  |
|          | 1.1          | Introduction          | 3  |
|          | 1.2          | Historical Background | 3  |
|          | 1.3          | What's on the CD-ROM  | 4  |
|          | 1.4          | Mathematica           | 5  |
|          | 1.5          | Starting Descarta2D   | 6  |
|          | 1.6          | Outline of the Book   | 7  |
| <b>2</b> | Des          | carta2D Tour          | 9  |
|          | 2.1          | Points                | 9  |
|          | 2.2          | Equations             | 10 |
|          | 2.3          | Lines                 | 12 |
|          | 2.4          | Line Segments         | 13 |
|          | 2.5          | Circles               | 14 |
|          | 2.6          | Arcs                  | 15 |
|          | 2.7          | Triangles             | 16 |
|          | 2.8          | Parabolas             | 17 |
|          | 2.9          | Ellipses              | 18 |
|          | 2.10         | Hyperbolas            | 19 |
|          | 2.11         | Transformations       | 20 |
|          | 2.12         | Area and Arc Length   | 20 |
|          | 2.13         | Tangent Curves        | 21 |
|          | 2.14         | Symbolic Proofs       | 22 |
|          | 2.15         | Next Steps            | 23 |
| II       | $\mathbf{E}$ | lementary Geometry    | 25 |
| ૧        | Coc          | ordinates and Points  | 97 |

| 3 | Coo | rdinates and Points  | <b>27</b> |
|---|-----|--|-----------|
|   | 3.1 | Numbers  | 27        |
|   | 3.2 | $Rectangular \ Coordinates \ . \ . \ . \ . \ . \ . \ . \ . \ . \ $ | 28        |

|   | $3.3 \\ 3.4 \\ 3.5 \\ 3.6 \\ 3.7$ | Line Segments and Distance3Midpoint between Two Points3Point of Division of Two Points3Collinear Points3Explorations3 | $0\\ 3\\ 3\\ 6\\ 7$ |
|---|-----------------------------------|---|---------------------|
| 4 | Equ                               | ations and Graphs 33  | 9                   |
|   | 4.1                               | Variables and Functions   | 9                   |
|   | 4.2                               | Polynomials   | 9                   |
|   | 4.3                               | Equations   | 1                   |
|   | 4.4                               | Solving Equations   | 2                   |
|   | 4.5                               | Graphs  | 6                   |
|   | 4.6                               | Parametric Equations  | 7                   |
|   | 4.7                               | Explorations 4  | 8                   |
| 5 | Line                              | es and Line Segments 5  | 1                   |
| - | 5.1                               | General Equation  | 1                   |
|   | 5.2                               | Parallel and Perpendicular Lines  | 4                   |
|   | 5.3                               | Angle between Lines   | 5                   |
|   | 5.4                               | Two-Point Form  | 6                   |
|   | 5.5                               | Point–Slope Form  | 8                   |
|   | 5.6                               | Slope–Intercept Form  | 2                   |
|   | 5.7                               | Intercept Form  | 4                   |
|   | 5.8                               | Normal Form   | 5                   |
|   | 5.9                               | Intersection Point of Two Lines   | 9                   |
|   | 5.10                              | Point Projected Onto a Line   | 0                   |
|   | 5.11                              | Line Perpendicular to Line Segment  | 2                   |
|   | 5.12                              | Angle Bisector Lines    7      7  | 3                   |
|   | 5.13                              | Concurrent Lines  | 4                   |
|   | 5.14                              | Pencils of Lines  | 5                   |
|   | 5.15                              | Parametric Equations  | 8                   |
|   | 5.16                              | Explorations  | 1                   |
| 6 | Circ                              | eles 8  | 5                   |
|   | 6.1                               | Definitions and Standard Equation   | 5                   |
|   | 6.2                               | General Equation of a Circle  | 8                   |
|   | 6.3                               | Circle from Diameter  | 9                   |
|   | 6.4                               | Circle Through Three Points   | 0                   |
|   | 6.5                               | Intersection of a Line and a Circle   | 1                   |
|   | 6.6                               | Intersection of Two Circles   | 2                   |
|   | 6.7                               | Distance from a Point to a Circle   | 5                   |
|   | 6.8                               | Coaxial Circles   | 6                   |
|   | 6.9                               | Radical Axis  | 7                   |
|   | 6.10                              | Parametric Equations  | 9                   |

x

|   | 6.11 | Explorations                    |
|---|------|---------------------------------|
| 7 | Arc  | s 105                           |
|   | 7.1  | Definitions                     |
|   | 7.2  | Bulge Factor Arc                |
|   | 7.3  | Three–Point Arc                 |
|   | 7.4  | Parametric Equations            |
|   | 7.5  | Points and Angles at Parameters |
|   | 7.6  | Arcs from Ray Points            |
|   | 7.7  | Explorations                    |
| 8 | Tria | ingles 117                      |
|   | 8.1  | Definitions                     |
|   | 8.2  | Centroid of a Triangle          |
|   | 8.3  | Circumscribed Circle            |
|   | 8.4  | Inscribed Circle                |
|   | 8.5  | Solving Triangles               |
|   | 8.6  | Cevian Lengths                  |
|   | 8.7  | Explorations                    |

xi

#### III Conics

| 9  | Para | abolas                                 | 135 |  |
|----|------|--|-----|--|
|    | 9.1  | Definitions                            | 135 |  |
|    | 9.2  | General Equation of a Parabola         | 135 |  |
|    | 9.3  | Standard Forms of a Parabola           | 136 |  |
|    | 9.4  | Reduction to Standard Form             | 139 |  |
|    | 9.5  | Parabola from Focus and Directrix      | 140 |  |
|    | 9.6  | Parametric Equations                   | 141 |  |
|    | 9.7  | Explorations                           | 142 |  |
|    |      |  |     |  |
| 10 | EIII | pses                                   | 145 |  |
|    | 10.1 | Definitions                            | 145 |  |
|    | 10.2 | General Equation of an Ellipse         | 147 |  |
|    | 10.3 | Standard Forms of an Ellipse           | 147 |  |
|    | 10.4 | Reduction to Standard Form             | 150 |  |
|    | 10.5 | Ellipse from Vertices and Eccentricity | 151 |  |
|    | 10.6 | Ellipse from Foci and Eccentricity     | 153 |  |
|    | 10.7 | Ellipse from Focus and Directrix       | 153 |  |
|    | 10.8 | Parametric Equations                   | 155 |  |
|    | 10.9 | Explorations                           | 156 |  |

| 11 Hyperbolas                                 | 159   |
|---|-------|
| 11.1 Definitions                              | . 159 |
| 11.2 General Equation of a Hyperbola          | . 161 |
| 11.3 Standard Forms of a Hyperbola            | . 161 |
| 11.4 Reduction to Standard Form               | . 166 |
| 11.5 Hyperbola from Vertices and Eccentricity | . 167 |
| 11.6 Hyperbola from Foci and Eccentricity     | . 168 |
| 11.7 Hyperbola from Focus and Directrix       | . 169 |
| 11.8 Parametric Equations                     | . 170 |
| 11.9 Explorations                             | . 173 |
| 12 Company Contra                             | 175   |
| 12 General Comics                             | 175   |
| 12.1 Conic from Quadratic Equation            | . 1(0 |
| 12.2 Classification of Conics                 | . 184 |
| 12.3 Center Point of a Conic                  | . 184 |
| 12.4 Conic from Point, Line and Eccentricity  | . 185 |
| 12.5 Common Vertex Equation                   | . 180 |
| 12.6 Conic Intersections                      | . 189 |
| 12.7 Explorations                             | . 190 |
| 13 Conic Arcs                                 | 193   |
| 13.1 Definition of a Conic Arc                | . 193 |
| 13.2 Equation of a Conic Arc                  | . 194 |
| 13.3 Projective Discriminant                  | . 196 |
| 13.4 Conic Characteristics                    | . 196 |
| 13.5 Parametric Equations                     | . 198 |
| 13.6 Explorations                             | . 199 |
|   | 001   |
| 14 Medial Curves                              | 201   |
| 14.1 Point–Point                              | . 201 |
| 14.2 Point-Line                               | . 202 |
| 14.5 POINT-UIRCIE                             | . 204 |
| 14.4 Line–Line                                | . 200 |
|   | . 207 |
| 14.0 Urcle-Urcle                              | . 210 |
| 14.7 Explorations                             | . 212 |
| IV Geometric Functions                        | 215   |
|   | 210   |
| 15 Transformations                            | 217   |
| 15.1 Translations                             | . 217 |
| 15.2 Rotations                                | . 219 |
| 15.3 Scaling                                  | . 222 |
|   |       |

#### Contents

|    | 15.4 | Reflections                     |
|----|------|---------------------------------|
|    | 15.5 | Explorations                    |
|    |      | T 11                            |
| 16 | Arc  | Length 229                      |
|    | 16.1 | Lines and Line Segments         |
|    | 16.2 | Perimeter of a Triangle         |
|    | 16.3 | Polygons Approximating Curves   |
|    | 16.4 | Circles and Arcs                |
|    | 16.5 | Ellipses and Hyperbolas         |
|    | 16.6 | Parabolas                       |
|    | 16.7 | Chord Parameters                |
|    | 16.8 | Summary of Arc Length Functions |
|    | 16.9 | Explorations                    |
|    | •    |                                 |
| 17 | Area | a 237                           |
|    | 17.1 | Areas of Geometric Figures      |
|    | 17.2 | Curved Areas                    |
|    | 17.3 | Circular Areas                  |
|    | 17.4 | Elliptic Areas                  |
|    | 17.5 | Hyperbolic Areas                |
|    | 17.6 | Parabolic Areas                 |
|    | 17.7 | Conic Arc Area                  |
|    | 17.8 | Summary of Area Functions       |
|    | 17.9 | Explorations                    |
|    |      |                                 |

### V Tangent Curves

| 18 Tangent Lines                              | 255 |
|---|-----|
| 18.1 Lines Tangent to a Circle                | 255 |
| 18.2 Lines Tangent to Conics                  | 266 |
| 18.3 Lines Tangent to Standard Conics         | 273 |
| 18.4 Explorations                             | 280 |
| 19 Tangent Circles                            | 283 |
| 19.1 Tangent Object, Center Point             | 283 |
| 19.2 Tangent Object, Center on Object, Radius | 285 |
| 19.3 Two Tangent Objects, Center on Object    | 286 |
| 19.4 Two Tangent Objects, Radius              | 287 |
| 19.5 Three Tangent Objects                    | 288 |
| 19.6 Explorations                             | 289 |

| 20 Tangent Conics                    | 293 |
|--------------------------------------|-----|
| 20.1 Constraint Equations            | 293 |
| 20.2 Systems of Quadratics           | 294 |
| 20.3 Validity Conditions             | 296 |
| 20.4 Five Points                     | 296 |
| 20.5 Four Points, One Tangent Line   | 298 |
| 20.6 Three Points, Two Tangent Lines | 301 |
| 20.7 Conics by Reciprocal Polars     | 306 |
| 20.8 Explorations                    | 310 |
| 21 Biarcs                            | 311 |
| 21.1 Biarc Carrier Circles           | 311 |
| 21.2 Knot Point                      | 314 |
| 21.3 Knot Circles                    | 316 |
| 21.4 Biarc Programming Examples      | 317 |
| 21.5 Explorations                    | 322 |
|                                      |     |
| VI Reference                         | 323 |
| 22 Technical Notes                   | 325 |
| 22.1 Computation Levels              | 325 |
| 22.2 Names                           | 326 |
| 22.3 Descarta2D Objects              | 326 |
| 22.4 Descarta2D Packages             |     |
| 22.5 Descarta2D Functions            | 338 |
| 22.6 Descarta2D Documentation        |     |
| 23 Command Browser                   | 3/1 |
| 25 Command Drowser                   | 041 |
| 24 Error Messages                    | 367 |
| VII Packages                         | 385 |
| D2DArc2D                             | 387 |
| D2DArcLength2D                       | 395 |
|                                      | 399 |
| D2DCircle2D                          | 405 |
| D2DConic2D                           | 411 |
| D2DConic4rc2D                        | 415 |
| D2DE11ipse2D                         | 421 |
| D2DEquations2D                       |     |
| D2DExpressions2D                     | 429 |
| D2DGeometrv2D                        | 437 |
|                                      |     |

| D2DHyperbola2D $\ldots$ $\ldots$ $\ldots$ $445$  |
|--|
| D2DIntersect2D $\ldots$ $\ldots$ $\ldots$ $453$  |
| D2DLine2D  |
| D2DLoci2D  |
| D2DMaster2D $\ldots$ $\ldots$ $\ldots$ $\ldots$ $469$  |
| D2DMedial2D $\ldots$ $\ldots$ $\ldots$ $\ldots$ $473$  |
| D2DNumbers2D $\ldots$ $\ldots$ $\ldots$ $477$  |
| D2DParabola2D  |
| D2DPencil2D $\ldots$ $\ldots$ $\ldots$ $\ldots$ $485$  |
| D2DPoint2D $\ldots$ $\ldots$ $\ldots$ $\ldots$ $489$   |
| D2DQuadratic2D $\ldots$ $\ldots$ $\ldots$ $497$  |
| D2DSegment2D $\ldots$ $\ldots$ $\ldots$ $\ldots$ $\ldots$ $\ldots$ $505$   |
| D2DSketch2D  |
| D2DSolve2D $\ldots$ $\ldots$ $\ldots$ $\ldots$ $\ldots$ $515$  |
| t D2DTangentCircles2D $	t 519$   |
| D2DTangentConics2D $\ldots \ldots 523$ |
| D2DTangentLines2D $\ldots$ $\ldots$ $\ldots$ $\ldots$ $\ldots$ $\ldots$ $\ldots$ $\ldots$ $\ldots$ $531$   |
| D2DTangentPoints2D $\ldots \ldots 537$ |
| D2DTransform2D $\ldots$ $\ldots$ $\ldots$ $\ldots$ $539$   |
| D2DTriangle2D  |
|  |

#### VIII Explorations

| 5 | 5 | 5 |
|---|---|---|
| J | J | J |

| apollon.nb,             | Circle of Apollonius   |
|-------------------------|--|
| arccent.nb,             | Centroid of Semicircular Arc   |
| arcentry.nb,            | Arc from Bounding Points and Entry Direction   |
| arcexit.nb,             | Arc from Bounding Points and Exit Direction  |
| archimed.nb,            | Archimedes' Circles  |
| arcmidpt.nb,            | Midpoint of an Arc $\ldots \ldots 567$ |
| <pre>caarclen.nb,</pre> | Arc Length of a Parabolic Conic Arc  |
| caarea1.nb,             | Area of a Conic Arc (General)  |
| caarea2.nb,             | Area of a Conic Arc (Parabola)   |
| cacenter.nb,            | Center of a Conic Arc  |
| cacircle.nb,            | Circular Conic Arc   |
| camedian.nb,            | Shoulder Point on Median   |
| caparam.nb,             | Parametric Equations of a Conic Arc  |
| carlyle.nb,             | Carlyle Circle   |
| castill.nb,             | Castillon's Problem  |
| catnln.nb,              | Tangent Line at Shoulder Point   |
| center.nb,              | Center of a Quadratic  |
| chdlen.nb,              | Chord Length of Intersecting Circles   |
| cir3pts.nb,             | Circle Through Three Points  |
| circarea.nb,            | One-Third of a Circle's Area   |
|                         |  |

| cirptmid.nb,           | Circle–Point Midpoint Theorem                |
|------------------------|--|
| cramer2.nb,            | Cramer's Rule (Two Equations)                |
| cramer3.nb,            | Cramer's Rule (Three Equations)              |
| deter.nb,              | Determinants                                 |
| elfocdir.nb,           | Focus of Ellipse is Pole of Directrix        |
| elimlin.nb,            | Eliminate Linear Terms                       |
| <pre>elimxy1.nb,</pre> | Eliminate Cross-Term by Rotation             |
| elimxy2.nb,            | Eliminate Cross-Term by Change in Variables  |
| elimxy3.nb,            | Eliminate Cross-Term by Change in Variables  |
| elldist.nb,            | Ellipse Locus, Distance from Two Lines       |
| ellfd.nb,              | Ellipse from Focus and Directrix             |
| ellips2a.nb,           | Sum of Focal Distances of an Ellipse         |
| elllen.nb,             | Length of Ellipse Focal Chord                |
| ellrad.nb,             | Apoapsis and Periapsis of an Ellipse         |
| ellsim.nb,             | Similar Ellipses                             |
| ellslp.nb,             | Tangent to an Ellipse with Slope             |
| eqarea.nb,             | Equal Areas Point                            |
| eyeball.nb,            | Eyeball Theorem                              |
| gergonne.nb,           | Gergonne Point of a Triangle                 |
| heron.nb,              | Heron's Formula                              |
| hyp2a.nb,              | Focal Distances of a Hyperbola               |
| hyp4pts.nb,            | Equilateral Hyperbolas                       |
| hyparea.nb,            | Areas Related to Hyperbolas                  |
| hypeccen.nb,           | Eccentricities of Conjugate Hyperbolas       |
| hypfd.nb,              | Hyperbola from Focus and Directrix           |
| hypinv.nb,             | Rectangular Hyperbola Distances              |
| hyplen.nb,             | Length of Hyperbola Focal Chord              |
| hypslp.nb,             | Tangent to a Hyperbola with Given Slope      |
| hyptrig.nb,            | Trigonometric Parametric Equations           |
| intrsct.nb,            | Intersection of Lines in Intercept Form      |
| inverse.nb,            | Inversion                                    |
| johnson.nb,            | Johnson's Congruent Circle Theorem           |
| knotin.nb,             | Incenter on Knot Circle                      |
| lndet.nb,              | Line General Equation Determinant            |
| lndist.nb,             | Vertical/Horizontal Distance to a Line       |
| lnlndist.nb,           | Line Segment Cut by Two Lines                |
| lnquad.nb,             | Line Normal to a Quadratic                   |
| lnsdst.nb,             | Distance Between Parallel Lines              |
| lnsegint.nb,           | Intersection Parameters of Two Line Segments |
| lnsegpt.nb,            | Intersection Point of Two Line Segments      |
| lnsperp.nb,            | Equations of Perpendicular Lines             |
| lntancir.nb,           | Line Tangent to a Circle                     |
| lntancon.nb,           | Line Tangent to a Conic                      |

| Medial Curve, Circle–Circle              |
|--|
| Medial Curve, Line–Circle                |
| Medial Curve, Line–Line                  |
| Medial Curve, Point–Circle               |
| Medial Curve, Point–Line                 |
| Medial Curve, Point–Point                |
| Medial Curve Type                        |
| Monge's Theorem                          |
| Approximate Arc Length of a Curve        |
| Normals and Minimum Distance             |
| Parabola Through Three Points            |
| Parabola Through Four Points             |
| Parabola Intersection Angle              |
| Parabolic Arch                           |
| Arc Length of a Parabola                 |
| Parabola Determinant                     |
| Length of Parabola Focal Chord           |
| Tangent to a Parabola with a Given Slope |
| Circle Tangent to a Parabola             |
| Perpendicular Tangents to a Parabola     |
| Polar Equation of a Circle               |
| Collinear Polar Coordinates              |
| Polar Equation of a Conic                |
| Distance Using Polar Coordinates         |
| Polar Equation of an Ellipse             |
| Polar Equations                          |
| Polar Equation of a Hyperbola            |
| Polar Equation of a Parabola             |
| Non-uniqueness of Polar Coordinates      |
| Parameterization of a Quadratic          |
| Collinear Points                         |
| Radical Axis of Two Circles              |
| Radical Center                           |
| Radical Axis Ratio                       |
| Reciprocal of a Circle                   |
| Reciprocals of Points and Lines          |
| Reciprocal of a Quadratic                |
| Reflection in a Point                    |
| Angle Inscribed in a Semicircle          |
| Circle Inscribed in a Right Triangle     |
| Coordinates of Shoulder Point            |
| Stewart's Theorem                        |
| Circle Tangent to Circle, Given Center   |
|  |

 $\mathbf{845}$ 

| <pre>tancir2.nb,</pre> | Circle Tangent to Circle, Center on Circle, Radius |       | <br>. 79   | 91 |
|------------------------|--|-------|------------|----|
| <pre>tancir3.nb,</pre> | Circle Tangent to Two Lines, Radius                | <br>  | <br>. 79   | 93 |
| <pre>tancir4.nb,</pre> | Circle Through Two Points, Center on Circle        |       | <br>. 79   | 95 |
| <pre>tancir5.nb,</pre> | Circle Tangent to Three Lines                      | <br>  | <br>. 79   | 97 |
| tancirpt.nb            | , Tangency Point on a Circle                       | <br>  | <br>. 79   | 99 |
| <pre>tetra.nb,</pre>   | Area of a Tetrahedron's Base                       | <br>  | <br>. 80   | 01 |
| tncirtri.nb            | , Circles Tangent to an Isosceles Triangle         | <br>• | <br>. 80   | 03 |
| <pre>tnlncir.nb,</pre> | Construction of Two Related Circles                | <br>  | <br>. 80   | 07 |
| triallen.nb            | , Triangle Altitude Length                         | <br>• | <br>. 80   | 09 |
| <pre>trialt.nb,</pre>  | Altitude of a Triangle                             | <br>• | <br>. 8    | 11 |
| <pre>triarea.nb,</pre> | Area of Triangle Configurations                    | <br>• | <br>. 8    | 13 |
| triarlns.nb            | , Area of Triangle Bounded by Lines                | <br>• | <br>. 8    | 15 |
| <pre>tricent.nb,</pre> | Centroid of a Triangle                             |       | <br>. 8    | 17 |
| <pre>tricev.nb,</pre>  | Triangle Cevian Lengths                            |       | <br>. 8    | 19 |
| <pre>triconn.nb,</pre> | Concurrent Triangle Altitudes                      |       | <br>. 82   | 23 |
| <pre>tridist.nb,</pre> | Hypotenuse Midpoint Distance                       | <br>• | <br>. 82   | 27 |
| trieuler.nb            | , Euler's Triangle Formula                         |       | <br>. 82   | 29 |
| <pre>trirad.nb,</pre>  | Triangle Radii                                     |       | <br>. 83   | 33 |
| trisides.nb            | , Triangle Side Lengths from Altitudes             | <br>• | <br>. 83   | 35 |
|                        |  |       |            |    |
|                        |  |       | ~ ~        | _  |
| IX Epilogu             | e  |       | 83         | 7  |
| Installation Ins       | structions   |       | 83         | 39 |
| Bibliography           |  |       | <b>8</b> 4 | 43 |

| ibilography |  |  |  |
|-------------|--|--|--|
|             |  |  |  |
|             |  |  |  |

Index

Part I Introduction

### Chapter 1

### **Getting Started**

#### 1.1 Introduction

The purpose of this book is to provide a broad introduction to analytic geometry using the *Mathematica* and *Descarta2D* computer programs to enhance the numerical, symbolic and graphical nature of the subject. The book has the following objectives:

- To provide a computer-based alternative to a traditional course in analytic geometry.
- To provide a geometric research tool that can be used to explore numerically and symbolically various theorems and relationships of two-dimensional analytic geometry. Due to the nature of the *Mathematica* environment in which *Descarta2D* was written, the system can be easily enhanced and extended.
- To provide a reference of geometric formulas from analytic geometry that are not generally provided in more broad-based mathematical textbooks, nor included in mathematical handbooks.
- To provide a large-scale *Mathematica* programming tutorial that is instructive in the techniques of object oriented programming, modular packaging and good overall system design. By providing the full source code for the *Descarta2D* system, students and researchers can modify and enhance the system for their own purposes.

#### 1.2 Historical Background

The word *geometry* is derived from the Greek words for "earth measure." Early geometers considered measurements of line segments, angles and other planar figures. Analytic geometry was introduced by René Descartes in his *La Géométrie* published in 1637. Accordingly, after his name, analytic or coordinate geometry is often referred to as *Cartesian* geometry. It is essentially a method of studying geometry by means of algebra. Earlier mathematicians had



Figure 1.1: Organization of the CD-ROM.

continued to resort to the conventional methods of geometric reasoning as set forth in great detail by Euclid and his school some 2000 years before. The tremendous advances made in the study of geometry since the time of Descartes are largely due to his introduction of the coordinate system and the associated algebraic or analytic methods.

With the advent of powerful mathematical computer software, such as *Mathematica*, much of the tedious algebraic manipulation has been removed from the study of analytic geometry, allowing comfortable exploration of the subject even by amateur mathematicians. *Mathematica* provides a programmable environment, meaning that the user can extend and expand the capabilities of the system including the addition of completely new concepts not covered by the kernel *Mathematica* system. This notion of expandability serves as the basis for the implementation of the *Descarta2D* system, which is essentially an extension of the capabilities of *Mathematica* cast into the world of analytic geometry.

#### 1.3 What's on the CD-ROM

The CD-ROM supplied with this book is organized as shown in Figure 1.1. Detailed instructions for installing the software can be found in the chapter entitled "Installation Instructions" near the end of the book. The file readme.txt on the CD contains essentially the same information as the "Installation Instructions" chapter.

There are four folders at the highest directory level on the CD. The folder AcrobatReader contains Adobe's Acrobat Reader (used to view \*.pdf files) and the folder MathReader contains Wolfram Research's *MathReader* (used to view \*.nb files). The folder Book contains a complete copy of the book in Adobe Portable Document Format (PFD).

The folder Descarta2D contains the software described in this book as shown in Figure 1.2. These files are organized so that they can easily be installed for usage by *Mathematica*. The correct placement of these files on your computer's hard drive is described in the "Installation Instructions" chapter.



Figure 1.2: Organization of the Descarta2D folder.

All of the software packages and explorations in this book were developed on a Pentium Pro computer system using version 4.0 of the Windows NT operating system and *Mathematica* version 3.0.1. Due to the portability of *Mathematica*, the software should operate identically on other computer systems, including other Intel-based personal computers, Macintoshes and a wide range of Unix workstations. The Adobe pdf files on the CD are also portable and should be readable on a variety of operating systems.

#### 1.4 Mathematica

In this book an assumption is made that you have at least a rudimentary understanding of how to run the *Mathematica* program, how to enter commands and receive results, and how to arrange files on a computer disk so that programs can locate them. A sufficient introduction to *Mathematica* would be gained by reading the "Tour of Mathematica" in Stephen Wolfram's book *Mathematica: A System for Doing Mathematics by Computer.* 

The syntax Mathematica uses for mathematical operations differs somewhat from traditional mathematical notation. Since Descarta2D is implemented in the Mathematica programming language it follows all the syntactic conventions of the Mathematica system. See Wolfram's Mathematica book for more detailed descriptions of the syntax. Once you become familiar with Mathematica you will begin to appreciate the consistency and predictability of the system.

#### 1.5 Starting Descarta2D

All of the underlying concepts of analytic geometry presented in this book are implemented in a *Mathematica* program called *Descarta2D*. *Descarta2D* consists of a number of *Mathematica* programs (called *packages*) that provide a rich environment for the study of analytic geometry. In order to avoid loading all the packages at one time, a master file of package declarations has been provided. You must load this file at the beginning of any *Mathematica* session that will make use of the *Descarta2D* packages. Once the package declarations have been loaded, all of the additional packages will be loaded automatically when they are needed. To load the *Descarta2D* package declarations from the file *init.m* use the command

In[1]: << Descarta2D`</pre>

If this is the first command in the *Mathematica* session, the *Mathematica* kernel will be loaded first, and then the declarations will be loaded. Depending on the speed of your computer this may take a few seconds or several minutes. After the initial start-up, packages will load at automatically as new *Descarta2D* functions are used for the first time. When a package is first loaded, you may notice a delay in computing results; after the package is loaded, results are computed immediately and the time taken depends on the complexity of the computation.

The examples in this book that illustrate the usage of *Descarta2D* were chosen primarily for their simplicity, rather than to correspond to significant calculations in analytic geometry. At the end of each chapter a section entitled "Explorations" provides more realistic applications of *Descarta2D*. All of the examples in this book were generated by running an actual copy of *Mathematica* version 3.0.1. The interactive dialogs of each *Mathematica* session are provided in the corresponding chapter notebook on the CD, so very little typing is required to replicate the output and plots in each chapter. If you choose to enter the commands yourself instead of using the notebook on the CD, you should enter the commands exactly as they are printed (including all spaces and line breaks). This will insure that you obtain the same results as printed in the text. Once you become more familiar with *Mathematica* and *Descarta2D*, you will learn what deviations from the printed text are acceptable.

#### Plotting Descarta2D Objects

Graphically rendering (plotting) the geometric objects encountered in a study of analytic geometry greatly enhances the intuitive understanding of the subject. *Mathematica* provides a wide variety of commands for plotting objects including Graphics, Plot and ParametricPlot. There are also specialized commands such as ImplicitPlot and PolarPlot. Each of these commands has a wide variety of options, giving the user detailed control over the plotted output.

These Mathematica commands can also be used to plot Descarta2D objects, and, in fact, the figures found in this book were generated using the Mathematica plotting commands named above. However, the Descarta2D system provides another command, Sketch2D, for plotting Descarta2D objects. The Sketch2D command has a very simple syntax as illustrated in the following example.

**Example.** Plot these objects using the Sketch2D command: Point2D[{-1,2}], Line2D[2,-3,1] and Circle2D[{1,0},2]. (The meaning of these geometric objects will be explained in subsequent chapters; for now it is sufficient to understand that we are plotting a point, a line and a circle.)

**Solution.** The Descarta2D function Sketch2D[objList] plots a list of geometric objects.

In[2]: Sketch2D[{Point2D[{-1, 2}], Line2D[2, -3, 1], Circle2D[{1, 0}, 2]}];



#### **1.6** Outline of the Book

The book is divided into nine sections. The first five sections deal with the subject matter of analytic geometry; the remaining sections provide a reference manual for the use of the *Descarta2D* computer program and a listing of the source code for the packages that implement *Descarta2D*, as well as the solutions to the explorations.

Part I of the book serves as an introduction and begins with the material in this chapter aimed at getting started with the subject; the next chapter continues the introduction by providing a high-level tour of *Descarta2D*. Part II introduces the basic geometric objects studied in analytic geometry, including points and coordinates, equations and graphs, lines, line segments, circles, arcs and triangles. Part III continues by studying second-degree curves, parabolas, ellipses and hyperbolas. In addition, Part III provides a more general study of conic curves by examining general conics, conic arcs and medial curves.

Part IV covers geometric functions including transformations (translation, rotation, scaling and reflection) and the computation of areas and arc lengths. The subject of tangent curves is covered in Part V with specific chapters dedicated to tangent lines, tangent circles and tangent conics. The final chapter in Part V is an overview of biarc circles, which are a special form of tangent circles. The intent of this chapter is to illustrate how new capabilities can be added to *Descarta2D*. Generally, the chapters comprising Parts I through V present material in sections with simple examples. The examples are sometimes supplemented with *Descarta2D* and *Mathematica* Hints that illustrate the more subtle usages of the commands. Each chapter ends with an "Explorations" section containing several more challenging problems in analytic geometry. The solutions for the explorations are provided as *Mathematica* notebooks on the CD, as well as being listed alphabetically in Part VIII.

Parts VI and VII serve as a reference manual for the *Descarta2D* system. The reference manual includes a description of the geometric objects provided by *Descarta2D*, a browser for quickly finding command syntax and options, and a listing of the error messages that may be generated. Part VII provides a complete listing, with comments, of all the packages comprising *Descarta2D*.

Part VIII of the book contains reproductions of the notebooks which provide the solutions to the explorations found at the end of each chapter. The notebooks are listed in alphabetical order by their file names. The exploration notebook files may also be reviewed directly off the CD using Mathematica or MathReader.

Part IX contains the instructions for installing *Descarta2D* on your computer system as well as a Bibliography and a detailed index.

### Chapter 2

## **Descarta2D Tour**

The purpose of this chapter is to provide a tour consisting of examples to show a few of the things *Descarta2D* can do. Concepts introduced informally in this chapter will be studied in detail in subsequent chapters. The tour is not intended to be a complete overview of *Descarta2D*, but just a sampling of a few of the capabilities provided by *Descarta2D*.

#### 2.1 Points

The simplest geometric object is a point in the plane. The location of a point is specified by a pair of numbers called the x- and y-coordinates of the point and is written as (x, y). In Mathematica and Descarta2D point coordinates are enclosed in curly braces as  $\{x, y\}$ . In Descarta2D a point with coordinates (x, y) is represented as Point2D[ $\{x, y\}$ ]. The following commands are used to plot the points (1, 2), (3, -4) and (-2, 3):

```
In[1]: Sketch2D[{Point2D[{1, 2}], Point2D[{3, -4}],
Point2D[{-2, 3}]}];
3
2
1
0
-1
-2
-3
-4
-2-1 0 1 2 3
```

Mathematica allows us to assign symbolic names to expressions. The commands

```
In[2]: p1 = Point2D[{1, 2}];
    p2 = Point2D[{3, -4}];
    p3 = Point2D[{-2, 3}];
```

assign the names p1, p2 and p3 to the points sketched previously. After a name is assigned, we can refer to the object by using its name.

```
In[3]: {p1, p2, p3}
Out[3] {Point2D[{1, 2}], Point2D[{3, -4}], Point2D[{-2, 3}]}
```

Descarta2D provides numerous commands for constructing points. These commands have the name Point2D followed by a sequence of arguments, separated by commas and enclosed in square brackets. For example, the command

```
In[4]: p3 = Point2D[p1 = Point2D[{-3, -2}], p2 = Point2D[{2, 1}]]
Out[4] Point2D[{-1/2, -1/2}]
```

constructs a point, named p3, that is the midpoint of two other points named p1 and p2.

#### 2.2 Equations

The underlying principle of analytic geometry is to link algebra to the study of geometry. There are two fundamental problems studied in analytic geometry: (1) given the *equation* of a curve determine its shape, location and other geometric characteristics; and (2) given a description of the plot of a curve (its *locus*) determine the equation of the curve. Equations are represented in *Mathematica* and *Descarta2D* in a manner that is very similar to standard algebra. For example, the linear equation 2x + 3y - 4 = 0 is entered using the following command:

Mathematica Hint. Mathematica uses the double equals sign, ==, to represent the equality in an equation; the single equals sign, =, as has already been shown, is used to assign names. Also, notice that Mathematica sorts all output into a standard order that may be different than the order you typed.

The left side of the equation above is called a *linear polynomial* in two unknowns. The general form of a linear polynomial in two unknowns is given by

$$Ax + By + C$$

Since linear polynomials occur frequently in the study of analytic geometry, Descarta2D provides a special format for linear polynomials which is of the form Line2D[A, B, C] where A is the coefficient of the x term, B the coefficient of the y term and C is the constant term. Descarta2D also provides functions for converting between linear polynomials and Line2D objects.

Frequently we will also be interested in *quadratic* equations which represent such curves as circles, ellipses, hyperbolas and parabolas. The algebraic form of a quadratic equation is

$$Ax^{2} + Bxy + Cy^{2} + Dx + Ey + F = 0.$$

Descarta2D provides a special form for representing a quadratic polynomial which is

Quadratic2D[
$$A$$
,  $B$ ,  $C$ ,  $D$ ,  $E$ ,  $F$ ]

and functions for converting between polynomials and Quadratic2D objects.

```
In[9]: Clear[x, y];
    poly1 = 2 * x<sup>2</sup> + 3 * x * y + 3 * y<sup>2</sup> - 4 * x - 5 * y - 3;
    q1 = Quadratic2D[2, 3, 3, -4, -5, -3];
In[10]: Polynomial2D[q1, {x, y}]
Out[10] -3 - 4 x + 2 x<sup>2</sup> - 5 y + 3 x y + 3 y<sup>2</sup>
In[11]: Quadratic2D[poly1, {x, y}]
Out[11] Quadratic2D[2, 3, 3, -4, -5, -3]
```

Equations are often constructed so that they may be *solved* for numbers that make the equality true. For example, the quadratic equation in one unknown,  $x^2 - 7x + 10 = 0$  is solved when x = 2 or x = 5. Mathematica provides powerful functions for solving equations. For example, the **Solve** command can be used to find the solutions to the equation given above.

In[12]: Clear[x]; Solve[x^2 - 7 \* x + 10 == 0, x] Out[12] {{x  $\rightarrow$  2}, {x  $\rightarrow$  5}}

The Solve command returns solutions in the form of *Mathematica rules* which are useful in subsequent computations. We will often need to solve equations in order find the solutions to geometric problems.

#### 2.3 Lines

Intuitively, a straight line is a curve we might draw with a straightedge ruler. In mathematics, a line is considered to be infinite in length extending in both directions. We often think of a line as the shortest path connecting two points, and, in fact, this is one of the many methods provided by *Descarta2D* for constructing a line. Mathematically, a line is represented as a linear equation of the form

$$Ax + By + C = 0$$

where A, B and C are called the *coefficients* of the line and determine its position and direction. For example, in *Descarta2D* the line x - 2y + 4 = 0 is represented as Line2D[1, -2, 4]. The following command constructs a line from two points.

```
In[13]: 11 = Line2D[p1 = Point2D[{-3, -1}], p2 = Point2D[{3, 2}]]
Out[13] Line2D[-3, 6, -3]
```

This is the line -3x + 6y - 3 = 0. We can plot the points and the line to get graphical verification that the line passes through the two points.

```
In[14]: Sketch2D[{p1, p2, 11}];
```



We might be interested in the angle a line makes measured from the horizontal. The angle can be determined using

which indicates that the line makes an angle of approximately 0.463648 radians, or about 26.5651 degrees, measured from the horizontal.

**Descarta2D Hint.** All angles in Descarta2D are expressed in radians. A radian is an angular measure equal to  $180/\pi$  degrees (about 57.2958 degrees). The Mathematica constant Degree has the value  $\pi/180$ . Dividing an angle in radians by Degree converts the angle from radians to degrees.

We may want to construct lines with certain relationships to another line. For example, the following commands construct lines parallel and perpendicular to a given line through a given point.

Out[16] {Line2D[-3, -1, 7], Line2D[1, -3, 1]}

```
In[17]: Sketch2D[{p1, 11, 12, 13}];
```



#### 2.4 Line Segments

Perhaps it is more familiar to us that a line has a definite start point and end point. Such a line is called a *line segment* and is represented in *Descarta2D* as

Segment2D[
$$\{x_0, y_0\}, \{x_1, y_1\}$$
]

where  $(x_0, y_0)$  and  $(x_1, y_1)$  are the coordinates of the start and end points, respectively, of the line segment.



We might want to determine the midpoint of a line segment, and we could use the Point2D[*point*, *point*] function to do so, but *Descarta2D* provides a more convenient function for directly constructing the midpoint of a line segment.

```
In[19]: p1 = Point2D[11]
Out[19] Point2D[{ 1/2, -1/2 }]
In[20]: Sketch2D[{11, p1}];
```



#### 2.5 Circles

A circle's position is determined by its *center* point and its size is specified by its *radius*. The standard equation of a circle is

$$(x-h)^2 + (y-k)^2 = r^2$$

where (h, k) are the coordinates of the center point, and r is the radius of the circle. In *Descarta2D* a circle is represented as Circle2D[{h, k}, r].

As demonstrated by the example, the function Point2D[*circle*] constructs the center point of the circle. The function Radius2D[*circle*] returns the radius of a circle.

```
In[22]: Radius2D[c1]
Out[22] 2
```

Descarta2D provides many functions for constructing circles. For example, we can construct a circle that passes through three given points.

```
In[23]: pl = Point2D[{1, 2}];
    p2 = Point2D[{-1, 2}];
    p3 = Point2D[{0, -2}];
    c1 = Circle2D[p1, p2, p3]
Out[23] Circle2D[{0, 1/8}, 17/8]
```

```
In[24]: Sketch2D[{p1, p2, p3, c1}];
```



#### 2.6 Arcs

Just as a line segment is a portion of a line, an *arc* is a portion of a circle. We can specify the span of the arc in terms of the angles the arc's sector sides make with the horizontal. In *Descarta2D* an arc can be constructed using Arc2D[*point*, r, { $\theta_1$ ,  $\theta_2$ }] (this is not the standard representation of an arc, it is merely one of the ways *Descarta2D* provides for constructing an arc).

In[25]: A1 = Arc2D[Point2D[{2, 1}], 3, {Pi/6, 5 Pi/6}]; Sketch2D[{A1, Point2D[{2, 1}]}];



As with a circle, we can construct an arc in many ways. For example, we can construct an arc passing through three points.

```
In[26]: p1 = Point2D[{2, -1}];
    p2 = Point2D[{1, 2}];
    p3 = Point2D[{-2, 1}];
    a1 = Arc2D[p1, p2, p3]
Out[26] Arc2D[{2, -1}, {-2, 1}, 1]
In[27]: Sketch2D[{p1, p2, p3, a1}];
```



#### 2.7 Triangles

Triangles are defined by three line segments connecting three points called the *vertices* of the triangle. In *Descarta2D* a triangle is represented as

Triangle2D[ $\{x_1, y_1\}, \{x_2, y_2\}, \{x_3, y_3\}$ ].

```
In[28]: t1 = Triangle2D[{1, 4}, {8, 8}, {6, 1}];
Sketch2D[{t1}];
```



We can *inscribe* a circle inside a triangle, as well as *circumscribe* one about a triangle. We can also compute properties such as its center of gravity.

```
In[30]: Sketch2D[{t1, c1, c2, p1}];
```



#### 2.8 Parabolas

A parabola is the cross-sectional shape required for a reflective mirror to focus light to a single point. The standard equation of a parabola centered at (0,0) and opening to the right is  $y^2 = 4fx$ , where f is the *focal length*, the distance from the *vertex point* to the focus. We can apply a rotation,  $\theta$ , to the parabola to produce a parabola of the same shape, but opening in a different direction. In *Descarta2D* the expression Parabola2D[{h, k},  $f, \theta$ ] is used to represent a parabola.



#### 2.9 Ellipses

An ellipse is a shape of the path a planet makes as it orbits the sun. The standard equation for an ellipse is given by

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

where 2a is the length of the longer *major* axis, and 2b is the length of the *minor* axis. Ellipses in other positions and orientations may be obtained by moving the center point or by rotating the ellipse. In *Descarta2D* the expression Ellipse2D[{h, k},  $a, b, \theta$ ] is used to represent an ellipse.

```
In[32]: e1 = Ellipse2D[{0, 0}, 2, 1, 0];
e2 = Ellipse2D[{2, 1}, 3, 2, Pi/4];
Sketch2D[{e1, e2}];
3
0
-1
-2 -1 0 1 2 3 4
```

An ellipse has two *focus* points that can also be plotted.

In[33]: pts = Foci2D[e2]

 $Out[33] \left\{ Point2D\left[\left\{2 + \sqrt{\frac{5}{2}} , 1 + \sqrt{\frac{5}{2}}\right\}\right], Point2D\left[\left\{2 - \sqrt{\frac{5}{2}} , 1 - \sqrt{\frac{5}{2}}\right\}\right] \right\}$
In[34]: Sketch2D[{e2, pts}];



## 2.10 Hyperbolas

A hyperbola in standard position has an equation similar to an ellipse that is given by

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1.$$

As with the ellipse, the constants a and b represent the lengths of certain axes of the hyperbola. The hyperbola plot consists of two separate pieces, called *branches*, both extending to infinity in opposite directions. The lines bounding the extent of the hyperbola are called *asymptotes*. A second hyperbola, closely related to the first, is bounded by the same asymptotes and is called the *conjugate* hyperbola. Hyperbolas can also be rotated in the plane and moved by adjusting their center points. The expression Hyperbola2D[{h, k},  $a, b, \theta$ ] is used to represent a hyperbola in *Descarta2D*.

```
In[35]: h1 = Hyperbola2D[{0, 0}, 2, 1, 0];
       lns = Asymptotes2D[h1];
       h2 = Hyperbola2D[h1, Conjugate2D];
       Sketch2D[{lns}];
       Sketch2D[{lns, h1}];
       Sketch2D[{lns, h2}];
                            2
                           1
                           0
                          -1
                          -2
                                     -2
                                             Ò
                                                    2
                                                           4
                              -4
```



## 2.11 Transformations

We can change the position, size and orientation of an object by applying a *transformation* to the object. Common transformations include translating, rotating, scaling and reflecting. A Descarta2D object can be transformed to produce a new object.

```
In[36]: e1 = Ellipse2D[{0, 0}, 2, 1, 0];
       Sketch2D[{e1,
            Translate2D[e1, {3, 0}],
            Rotate2D[e1, Pi/2],
            Scale2D[e1, 2],
            Reflect2D[e1, Line2D[0, 1, -1]]}];
                          3
                          2
                          1
                          0
                        -1
                        -2
                                  -2
                                                   2
                           -4
                                           0
                                                          4
```

## 2.12 Area and Arc Length

Curves possess certain properties of interest such as area and length. These properties are independent of the position and orientation of the curve.

Additionally, it may be of interest to compute the arc length of a portion of a curve or areas bounded by more than one curve. *Descarta2D* has a variety of functions for performing such computations.

## 2.13 Tangent Curves

When two curves touch at a single point without crossing, the two curves are said to be *tangent* to each other. *Descarta2D* provides a wide variety of functions for computing tangent lines, circles and other tangent curves. This example produces the circles tangent to a line and a circle with a radius of 3/8. There are eight circles that satisfy these criteria.



This example produces the four lines tangent to two given circles.

```
In[39]: c1 = Circle2D[{2, 0}, 1];
c2 = Circle2D[{-3, 0}, 2];
t1 = TangentLines2D[c1, c2];
Sketch2D[{c1, c2, t1}];
```



Conic curves (ellipses, parabolas and hyperbolas) can also be constructed passing through points or tangent to lines. The following example constructs four ellipses that are tangent to three lines and pass through two points.

```
In[40]: l1 = Line2D[1, 0, -1];
l2 = Line2D[0, 1, -1];
l3 = Line2D[{10, 0}, {0, 6}];
p1 = Point2D[{2, 3}];
p2 = Point2D[{4, 2}];
t1 = TangentConics2D[{11, 12, 13, p1, p2}] //N;
```



Figure 2.1: Triangle altitudes theorem.

```
In[41]: Sketch2D[{11, 12, 13, p1, p2, t1},
        PlotRange -> {{0, 10}, {0, 6}},
        CurveLength2D -> 20];
```



## 2.14 Symbolic Proofs

As a final exercise on our tour of *Descarta2D* we will use the symbolic capabilities of *Mathematica* to prove a theorem about the perpendicular bisectors of the sides of a triangle. The symbolic capabilities of *Mathematica* allow us to derive and prove general assertions in analytic geometry. Many of the built-in *Descarta2D* functions were derived using these capabilities.

**Triangle Altitudes.** The three perpendicular bisectors of the sides of a triangle are concurrent in one point. Further, this point is the center of a circle that passes through the three vertices of the triangle.

Without loss of generality, we pick a convenient position for the triangle in the plane as shown in Figure 2.1. One vertex is located at the origin, the second on the +x-axis and the third is arbitrarily placed.

```
In[42]: Clear[a, b, d];
P1 = Point2D[{0, 0}];
P2 = Point2D[{d, 0}];
P3 = Point2D[{a, b}];
```

The perpendicular bisectors of the sides of the triangle pass through the midpoint of each side and are perpendicular to the side. Each of these lines is constructed using the *Descarta2D* command Line2D[*point*, *point*, Perpendicular2D].

```
In[43]: L12 = Line2D[P1, P2, Perpendicular2D];
L13 = Line2D[P1, P3, Perpendicular2D];
L23 = Line2D[P2, P3, Perpendicular2D];
```

By including the semicolon, ;, at the end of each statement, we instruct *Mathematica* to suppress the output from these statements. Since we are treating these lines symbolically, we have no need at this point to examine the output. If you are curious about the form of lines L12, L13 and L23, they can be examined by entering the command {L12, L13, L23}. We now intersect these lines in pairs to determine the points of intersection using the *Descarta2D* function Point2D[*line*, *line*] that constructs the point of intersection of two lines.

```
In[44]: \{P4 = Point2D[L12, L13] // Simplify, P5 = Point2D[L12, L23] // Simplify\}Out[44] \{Point2D[\{\frac{d}{2}, \frac{a^2 + b^2 - a d}{2 b}\}], Point2D[\{\frac{d}{2}, \frac{a^2 + b^2 - a d}{2 b}\}]\}
```

By inspection, the coordinates of these two points are identical, which proves the first part of the theorem. To prove the second part of the theorem we determine the distance from the intersection point to each of the vertex points and show that the distance is the same for all three vertex points.

Many of the explorations provided at the end of upcoming chapters were developed using techniques similar to the one outlined above. Using *Mathematica* and *Descarta2D* to prove general assertions in analytic geometry illustrates the power of these computer programs.

## 2.15 Next Steps

This completes our high-level tour of *Descarta2D*. Many of the concepts introduced informally in this chapter will be studied in detail in subsequent chapters. The explorations provided at the end of each chapter provide additional insight into the subject matter and will give you an opportunity to learn the techniques for solving problems using *Mathematica*. Although many of the chapters can be studied independently, the concepts introduced in earlier chapters are the underlying tools used in subsequent chapters. Therefore, a sequential reading and study of the book is recommended for best understanding and continuity.

Part II Elementary Geometry

## Chapter 3

# **Coordinates and Points**

The fundamental concept of analytic geometry is the one-to-one correspondence established between points in a plane and (x, y) coordinates. This chapter introduces these concepts and develops some simple functions involving points.

## 3.1 Numbers

Integers are the whole numbers used for counting, both negative and positive, as well as zero. Ratios of integers such as 1/2, 5/7, 4/1 and 23/15 are called *rational numbers*. Numbers that can be plotted as distances from a fixed point on a line are called *real numbers*. Examples are -8, 0, 2.1387,  $\sqrt{2}$ , 5/3 and  $\pi$ .

If a and b represent real numbers and  $i = \sqrt{-1}$ , the expression a + bi is a *complex number*. A complex number is the sum of a real number a and a *pure imaginary* number bi. The two complex numbers a + bi and a - bi are called *conjugate* complex numbers.

In general, this book deals with real numbers, but since we are using algebraic techniques to study geometry, complex numbers naturally arise in the formulations. *Mathematica* provides a variety of ways to represent numbers as summarized in Table 3.1.

| Type     | Examples                |  |  |
|----------|-------------------------|--|--|
| Integer  | -4, 0, 1, 2, 3          |  |  |
| Rational | 7/5, 3/4                |  |  |
| Real     | 1.25, 3.0, -45.0        |  |  |
| Complex  | 3 + 2 I, -2.45 - 3.57 I |  |  |

Table 3.1: Numbers in Mathematica.

| Constant                   | Mathematica |
|----------------------------|-------------|
| $\pi \approx 3.14159$      | Pi          |
| $e \approx 2.71828$        | E           |
| $\pi/180\approx 0.0174533$ | Degree      |
| $i = \sqrt{-1}$            | I           |

Table 3.2: Some common constants in Mathematica.

Any given number, integer, rational, real or complex, is a *constant*. Mathematica provides symbols for some common numbers that are fixed value *constants* as shown in Table 3.2. Sometimes we do not wish to specify what the particular constant is and indicate a general constant by any one of the letters  $a, b, c, \ldots, A, B, C, \ldots$ , and such constants are referred to as *parameters*.

### 3.2 Rectangular Coordinates

The basic idea in analytic geometry is to establish a one-to-one correspondence between the points of a plane and number pairs (x, y). This correspondence may be established in many ways, but the one most commonly used is as follows. Consider two perpendicular lines X'X and Y'Y intersecting in the point O. The horizontal line X'X is called the x-axis, and the vertical line Y'Y the y-axis, and together they form a rectangular coordinate system.

These axes divide the plane into four *quadrants* labeled I, II, III and IV as shown in Figure 3.1. The point O is called the *origin*. When numerical scales are established on the axes, positive distances x are laid off to the right of the origin and are called *abscissas*; negative abscissas are laid off to the left. Positive distances y are drawn upwards and are called *ordinates*; negative ordinates are drawn downward. Thus OX and OY have positive sense (or direction) while OX' and OY' have negative sense. The unit scales on the x-axis and the y-axis need not be the same, but problems in analytic geometry often assume the units are equal on both axes.

Clearly such a system of coordinates can be used to describe the positions of points in the plane. For example, by going out +3 units on the x-axis and +2 units on the y-axis a point labeled A is located as shown in Figure 3.2. The point A is said to have the pair of numbers 3 and 2 as its *coordinates*, and it is customary to write A(3, 2) or simply (3, 2). Similarly, B has the coordinates (-2, -1) and lies in the third quadrant. It is evident that for the point  $P_1$  pictured in the second quadrant, the x-coordinate is negative and the y-coordinate is positive. We will write  $P_1(x_1, y_1)$  as the general representation of a point  $P_1$  in the plane at coordinates  $x = x_1$  and  $y = y_1$ .

The fundamental principle of analytic geometric is that there exists a one-to-one correspondence between number pairs and points in the plane: to each pair of numbers there corresponds



Figure 3.1: Coordinate axes and quadrants.



Figure 3.2: Coordinates specifying positions in the plane.

one and only one point and, conversely, to each point in the plane there corresponds one and only one pair of numbers.

**Example.** Plot the points with the following coordinates: (-2,3), (4,2) and (-4,-1).

**Solution.** Descarta2D represents a point (x, y) as Point2D[ $\{x, y\}$ ]. The function Sketch2D[objList] plots a list of objects.

```
In[1]: Sketch2D[{Point2D[{-2, 3}],
    Point2D[{4, 2}],
    Point2D[{-4, -1}]]];
3
2
1
0
-1
-1
-1
-4
-2
0
2
```

The curly brackets surrounding the point's coordinates are optional and may be omitted. *Descarta2D* will automatically add the curly brackets when the point's abscissa and ordinate are given as two arguments, Point2D[x, y], as shown below. A symbolic name may be assigned to a point, and this name can be used later to refer to the point.

4

```
In[2]: pl = Point2D[-2, 3]
Out[2] Point2D[{-2, 3}]
In[3]: pl
Out[3] Point2D[{-2, 3}]
```

## 3.3 Line Segments and Distance

Given two points A and B on the x-axis, or on a line parallel to the x-axis, the line segment  $\overline{AB}$  from point A to point B extends over a certain number of units of length used as the scale on the x-axis. If the direction from A to B points to the right, we say that  $\overline{AB}$  is a positive

E



Figure 3.3: Distance between points.

segment. On the other hand, if the direction from A to B points to the left, we say that  $\overline{AB}$  is a *negative* segment. Then we can assign to the segment  $\overline{AB}$  a positive or negative number indicating the direction and number of units of the segment. This signed number is indicated by AB. The absolute value of  $\overline{AB}$ , indicated by  $|\overline{AB}|$ , is a positive number called the *length* of the line segment. When the context is clear the symbol AB may be used to represent the line containing the points A and B, the line segment  $\overline{AB}$ , or the length of the segment,  $|\overline{AB}|$ .

To calculate the number (positive or negative) of x-units in the segment AB, let  $x_2$  be the abscissa of B and let  $x_1$  be the abscissa of A. Then, if B is to the right of A, the number of x-units in the segment AB is equal to  $x_2 - x_1$ . We define BA to be the negative of segment AB. Thus

$$AB = x_2 - x_1$$
 and  $BA = x_1 - x_2$ .

In the same fashion we can define a directed segment CD on, or parallel to, the y-axis, to be positive or negative depending on whether the arrow from C to D points up (positive direction) or down (negative direction). Thus

$$CD = y_2 - y_1$$
 and  $DC = y_1 - y_2$ .

Let  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$  be two points lying in the first quadrant and draw line segments  $P_1Q$  and  $P_2Q$  parallel to the coordinate axes as shown in Figure 3.3. By subtracting the abscissas,  $P_1Q = x_2 - x_1$ ; similarly subtracting ordinates,  $P_2Q = y_2 - y_1$ . Making use of the Pythagorean Theorem on the right triangle  $P_1QP_2$ , we have

$$(P_1P_2)^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$$

and the positive distance  $P_1P_2$ , d, is given by

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

The same formula holds true regardless of the quadrants in which the points lie and regardless of the order in which the points are taken.

**Example.** Find the distance between the two points (3, -1) and (-4, -2).

**Solution.** Taking the points in the given order, we have

$$d = \sqrt{(-4-3)^2 + (-2-(-1))^2} = \sqrt{50} = 5\sqrt{2}.$$

Or, taking the points in the opposite order,

$$d = \sqrt{(3 - (-4))^2 + (-1 - (-2))^2} = \sqrt{50} = 5\sqrt{2}.$$

The Descarta2D function Distance2D[coord, coord] computes the distance between two locations given as coordinates. The function Distance2D[point, point] computes the distance between two points.

```
In[4]: {Distance2D[{3, -1}, {-4, -2}],
Distance2D[Point2D[{3, -1}], Point2D[{-4, -2}]]}
Out[4] {5 \sqrt{2}, 5 \sqrt{2}}
```

The coordinates of the points may be symbolic and the points themselves may be named points.

```
In[5]: Clear[x1, y1, x2, y2];
    p1 = Point2D[{x1, y1}]; p2 = Point2D[{x2, y2}];
    Distance2D[p1, p2]
Out[5] √(x1-x2)<sup>2</sup> + (y1-y2)<sup>2</sup>
```

Mathematica Hint. The Mathematica function Clear is used in the previous example and throughout other examples in this book to insure that variable names used in the examples are not set to some unintended value from a previous computation.

**Descarta2D Hint.** There are several Descarta2D functions that are handy for working with points and coordinates. Coordinates2D[*point*] returns the (x, y)coordinates of a point as the list  $\{x, y\}$ . The functions XCoordinate2D[*point*] and XCoordinate2D[*coord*] give the x-coordinate, and YCoordinate2D[*point*] and YCoordinate2D[*coord*] give the y-coordinate.

F



Figure 3.4: Midpoint between two points.

## 3.4 Midpoint between Two Points

The midpoint between two points is the point bisecting the line segment connecting the two points. If the coordinates of the two points are  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$  as shown in Figure 3.4, then the midpoint,  $P_{12}$ , has coordinates

$$\left(\frac{x_1+x_2}{2},\frac{y_1+y_2}{2}\right)$$

**Example.** Find the midpoint between the points (-2, 1) and (3, -2).

**Solution.** The function Point2D[*point*, *point*] returns the midpoint of the two points. Alternatively, the function Point2D[*lnseg*] returns the midpoint of a line segment.

In[6]: pl = Point2D[{-2, 1}]; p2 = Point2D[{3, -2}]; p12 = Point2D[p1, p2] Out[6] Point2D[{ 1/2, -1/2}]

F

## 3.5 Point of Division of Two Points

Given a directed line segment such as  $P_1P_2$ , we wish to find the coordinates of the point P which divides  $P_1P_2$  into a given ratio  $r_1/r_2$  as illustrated in Figure 3.5. Let P have the



Figure 3.5: Point of division.

coordinates (x, y) which are to be determined. Sense is important here and P must be located so that  $P_1P/PP_2 = r_1/r_2$ .

Since  $\triangle P_1 PQ$  and  $\triangle PSP_2$  are similar, it follows that  $(x - x_1)/r_1 = (x_2 - x)/r_2$ . Solving this equation for x yields

$$x = \frac{x_1 r_2 + x_2 r_1}{r_1 + r_2}.$$
(3.1)

Similarly,

$$y = \frac{y_1 r_2 + y_2 r_1}{r_1 + r_2}.$$
(3.2)

To find the midpoint of the segment  $P_1P_2$  the ratio  $r_1/r_2$  must be unity; hence  $r_1 = r_2$ and Equations (3.1) and (3.2) specialize to

$$x = \frac{x_1 + x_2}{2}$$
 and  $y = \frac{y_1 + y_2}{2}$ . (3.3)

Equations (3.1), (3.2) and (3.3) also have useful physical interpretations. In (3.1) and (3.2), let x and y be the coordinates of the center of gravity of masses  $r_1$  and  $r_2$  placed at  $P_1$  and  $P_2$ , respectively. If the masses are equal, the center of gravity lies halfway between them as indicated by (3.3).

It is of further interest to note the positions of P for various values of the ratio  $r_1/r_2$ . If this ratio is zero, then P coincides with  $P_1$ , and if this ratio is a positive number, P is an internal point of division. As  $r_1/r_2 \to +\infty$ ,  $P \to P_1$ . For  $-\infty < r_1/r_2 < -1$ , P is an external point of division (in the direction of  $P_1P_2$ ). For  $-1 < r_1/r_2 < 0$ , P is an external point in the opposite direction with  $P_1P$  negative and  $P_2P$  positive.

**Example.** Find the point that divides the line segment between the points  $P_1(-2,5)$  and  $P_2(4,-1)$  into the ratio  $r_1/r_2 = -2$ .



Figure 3.6: Point offset a distance towards a point.

**Solution.** The Descarta2D function Point2D[point, point,  $r_1$ ,  $r_2$ ] returns the point that divides the line segment between the points into the ratio  $r_1/r_2$ .

```
In[7]: Point2D[Point2D[{-2, 5}], Point2D[{4, -1}], -2, 1]
Out[7] Point2D[{10, -7}]
```

Notice that it is invalid for  $r_1 + r_2$  to equal zero in Equations (3.1) and (3.2) as this would tend to generate a point at infinity.

#### Point Offset a Distance

Given two points  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$  we wish to find the point offset a distance, d, from  $P_1$  in the direction of  $P_2$ . We can use the point of division formula from the previous section to determine the coordinates of the offset point. As shown in Figure 3.6 the desired point is a point of division between  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$  where  $r_1/r_2 = d/(D_{12} - d)$  and  $D_{12}$  is the distance between  $P_1$  and  $P_2$ . Using the point of division function from Descarta2D yields

```
 \begin{array}{l} In [8]: \mbox{Clear[x1, y1, x2, y2, d, D12];} \\ \mbox{Point2D[Point2D[{x1, y1}], Point2D[{x2, y2}], d, D12-d]} \\ Out [8] \mbox{Point2D[} \Big\{ \frac{(-d + D12) \ x1 + d \ x2}{D12}, \ \frac{(-d + D12) \ y1 + d \ y2}{D12} \Big\} \Big] \end{array}
```

Rearranging and using standard mathematical notation produces

$$P\left(x_1 + \frac{d}{D_{12}}(x_2 - x_1), y_1 + \frac{d}{D_{12}}(y_2 - y_1)\right)$$
(3.4)

where d is the (possibly negative) offset distance and  $D_{12}$  is the distance between the two points.

**Example.** Find the point offset a distance 2 from the point (3,1) towards the point (-2,4).

**Solution.** The Descarta2D function Point2D[point, point, d] returns the point offset a distance d from the first point to the second point.

```
In[9]: Point2D[Point2D[{3,1}], Point2D[{-2,4}], 2]
Out[9] Point2D[{3-5\sqrt{\frac{2}{17}}, 1+3\sqrt{\frac{2}{17}}}]
```

#### 

## 3.6 Collinear Points

Three distinct points  $P_1(x_1, y_1)$ ,  $P_2(x_2, y_2)$  and  $P_3(x_3, y_3)$  are said to be *collinear* if they lie on the same straight line. We can construct any point,  $P_3$ , on the line  $P_1P_2$  by selecting an appropriate value for d and applying Equation (3.4). All such points  $P_1$ ,  $P_2$  and  $P_3$  are obviously *collinear* by construction. Now consider the value of the determinant

$$\begin{array}{c|cccc} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{array}$$

Mathematica provides the Det command for expanding the value of such a determinant.

```
In[10]: Clear[x1, y1, x2, y2, x3, y3, d, D12];
Det[{{x1, y1, 1}, {x2, y2, 1}, {x3, y3, 1}]] /.
{x3 -> x1 + (x2 - x1) * d / D12, y3 -> y1 + (y2 - y1) * d / D12} // Simplify
Out[10] 0
```

We see from Mathematica that for any value of d, the determinant given is zero. Therefore, the necessary and sufficient condition that three points lie on the same line is given by the determinant equation

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0$$

where the coordinates of the points are  $P_1(x_1, y_1)$ ,  $P_2(x_2, y_2)$  and  $P_3(x_3, y_3)$ .

F

F

**Example.** Show that the three points (1, 2), (7, 6) and (4, 4) are collinear.

**Solution.** The Mathematica function Det[*elemList*] returns the determinant of the nested list of elements.

```
In[11]: Det[{{1, 2, 1}, {7, 6, 1}, {4, 4, 1}}]
Out[11] 0
```

Descarta2D provides a specific function for determining whether three points are collinear: IsCollinear2D[*point*, *point*, *point*] returns True if the points are collinear; otherwise, it returns False.

```
In[12]: IsCollinear2D[Point2D[{1, 2}], Point2D[{7, 6}], Point2D[{4, 4}]]
Out[12] True
```



**Descarta2D Hint.** Using IsCollinear2D is preferable to using the *Mathematica* function Det for determining collinearity because IsCollinear2D accommodates slight round-off errors that may occur in the floating point arithmetic in the computer.



## 3.7 Explorations

COLLINEAR POINTS. ..... ptscol.nbShow that the three points (3a, 0), (0, 3b) and (a, 2b) are collinear.

DISTANCE USING POLAR COORDINATES.....polardis.nb

The location of a point in the plane may be specified using *polar coordinates*,  $(r, \theta)$ , where r is the distance from the origin to the point, and  $\theta$  is the angle the ray to the point from the origin makes with the +x-axis. Show that the distance, d, between two points  $(r_1, \theta_1)$  and  $(r_2, \theta_2)$ , given in polar coordinates, is

$$d = \sqrt{r_1^2 + r_2^2 - 2r_1r_2\cos(\theta_1 - \theta_2)}.$$

$$(r, \theta + 2k\pi)$$
 and  $(-r, \theta + (2k+1)\pi)$ 

represent the same position in the plane for integer values of k.

STEWART'S THEOREM.....stewart.nb



Show that for any  $\triangle ABC$  as shown in the figure above the relationship between the lengths of the labeled line segments is given by

$$a^2m + b^2n = c(d^2 + mn).$$

COLLINEAR POLAR COORDINATES......polarcol.nb Show that the points  $P_1(r_1, \theta_1)$ ,  $P_2(r_2, \theta_2)$  and  $P_3(r_3, \theta_3)$  in polar coordinates are collinear if and only if

$$-r_1r_2\sin(\theta_1 - \theta_2) + r_1r_3\sin(\theta_1 - \theta_3) - r_2r_3\sin(\theta_2 - \theta_3) = 0.$$

HYPOTENUSE MIDPOINT DISTANCE. ..... tridist.nb Prove that the midpoint of the hypotenuse of a right triangle is equidistant from the vertices.

## Chapter 4

## **Equations and Graphs**

Using algebraic techniques to solve geometry problems is the difference in approach between analytic geometry and planar geometry. Use of such techniques links the algebraic concept of an equation to the graphical representation of geometry shown in a graph or plot. This chapter introduces some of the simple algebraic techniques for solving equations that are heavily used in analytic geometry.

## 4.1 Variables and Functions

A variable is a quantity to which arbitrary values may be assigned. Let x be a symbol representing such a variable and let the quantity represented by the symbol y depend on x. We call y a function of x and say that x is the independent variable, and y the dependent variable. Using standard mathematical notation, these statements are written as y = f(x) and is read "y is a function of x." The value of the function at x = a is written f(a). These definitions may be expanded so that a variable z depends on two independent quantities x and y (as in solid analytic geometry), and relationships of this type are written z = f(x, y).

A function y = f(x) is *real-valued* if y is real when x is real. If there is but one value of y for a given value of x, y is said to be a *single-valued* function. If, for a given value of x, y has more than one value, y is said to be *multiple-valued*. The function f(x) is *periodic* if  $f(x + P) \equiv f(x)$  for some *period*, P. Usually it is assumed that P is the least number for which this identity is true.

## 4.2 Polynomials

A mathematical expression consisting of a sum of various positive integer powers of a variable is called a *polynomial*. The largest exponent that appears in a polynomial is called the *degree* of the polynomial. Polynomials of low-degree have special names as shown in Table 4.1.

Polynomials can involve more than one variable. For example the polynomial x + 2y + 3 is a *linear polynomial* in *two* unknowns and  $x^2 + 3xy + 2y^2 - 2x + 4$  is a *quadratic polynomial* in

| Degree | NAME      | EXAMPLE            |
|--------|-----------|--------------------|
| 0      | Constant  | 3                  |
| 1      | Linear    | x + 1              |
| 2      | Quadratic | $ax^2 + bx + c$    |
| 3      | Cubic     | $x^3 - 2x + 7$     |
| 4      | Quartic   | $3t^4 - 2t^2 + 17$ |
| 5      | Quintic   | $s^{5} - 1$        |

Table 4.1: Low-degree polynomials.

two unknowns. Descarta2D provides special objects, called *equation objects*, for representing linear and quadratic polynomials in two unknowns (see Table 4.2).

**Example.** Convert the polynomials 4x - 2y + 1 and  $x^2 - 3xy + 3x - 2y + 4$  into equivalent Line2D and Quadratic2D objects. Perform the inverse conversions.

**Solution.** Line2D[*poly*,  $\{x, y\}$ ] and Quadratic2D[*poly*,  $\{x, y\}$ ] convert linear and quadratic polynomials into equivalent Line2D and Quadratic2D objects. The functions Polynomial2D[*line*,  $\{x, y\}$ ] and Polynomial2D[*quad*,  $\{x, y\}$ ] convert Line2D and Quadratic2D objects, respectively, into polynomials.

```
In[1]: Clear[x, y];
    {l1 = Line2D[4 * x - 2 * y + 1, {x, y}],
    q1 = Quadratic2D[x^2 - 3 * x * y + 3 * x - 2 * y + 4, {x, y}]}
Out[1] {Line2D[4, -2, 1], Quadratic2D[1, -3, 0, 3, -2, 4]}
In[2]: {Polynomial2D[11, {x, y}], Polynomial2D[q1, {x, y}]}
Out[2] {1 + 4 x - 2 y, 4 + 3 x + x<sup>2</sup> - 2 y - 3 x y}
```

| Table 4.2: | Descarta2D | equation | objects. |
|------------|------------|----------|----------|

| Polynomial                        | Descarta2D Object                                |
|-----------------------------------|--|
| Ax + By + C                       | Line2D[ <i>A</i> , <i>B</i> , <i>C</i> ]         |
| $Ax^2 + Bxy + Cy^2 + Dx + Ey + F$ | Quadratic2D[ $A$ , $B$ , $C$ , $D$ , $E$ , $F$ ] |

#### 4.3 Equations



Figure 4.1: Descarta2D objects, polynomials and equations.

## 4.3 Equations

F

If a function of a single variable, f(x), is set equal to zero, the relation f(x) = 0 is called an *equation*. This equation imposes a condition on the variable x which then can assume only certain values. For example, if Ax + B = 0, then x can take on only one value, x = -B/A. If the equation is  $\sin x = 0$ , x can assume an unlimited number of values of the form  $k\pi$ , where k is any integer. The process of finding the values of x that satisfy the equation is called *solving* the equation. The values of x which satisfy f(x) = 0 are called the *solutions* or *roots* of the equation. All of the real solutions of f(x) = 0 may be represented by points on a line such as the x-axis. These points constitute the graph of the equation in one dimension.

If a function of two variables, f(x, y), is set equal to zero the relation f(x, y) = 0 is also an equation. But this equation permits one of the variables to be *independent*, while the other is *dependent* and a function of the first. For example, f(x, y) = 0 might be solved for y in terms of x, yielding  $y = g_1(x)$ , indicating that x is the independent variable and y the dependent variable. Or f(x, y) = 0 might be solved for x yielding  $x = g_2(y)$  interchanging the independent and dependent variables.

In addition to representing polynomials, the Line2D and Quadratic2D objects may also be used to represent equations (the implicit assumption is that they represent polynomials set equal to zero). Figure 4.1 shows the relationships between polynomials, equations and *Descarta2D* equation objects. Table 4.3 summarizes the *Descarta2D* functions that accomplish the conversions labeled 1 to 4 in Figure 4.1.

**Example.** Convert the Descarta2D linear equation object Line2D[2, 3, -1] into an equivalent Mathematica equation. Similarly, convert the Descarta2D quadratric object Quadratic2D[1, -2, 2, 3, -3, 7] into a Mathematica equation.

**Solution.** The Descarta2D function Equation2D[line,  $\{x, y\}$ ] converts a Line2D object into a Mathematica equation. The function Equation2D[quad,  $\{x, y\}$ ] converts a Quadratic2D object into a Mathematica equation.

|   | $Descarta2D$ Function $\Rightarrow$ Result   |
|---|--|
| 1 | Line2D[ $ax + by + c$ , { $x$ , $y$ }] $\Rightarrow$ Line2D[ $a$ , $b$ , $c$ ]     |
|   | $\texttt{Quadratic2D}[ax^2 + bxy + cy^2 + dx + ey + f, \{x, y\}] \Rightarrow$      |
|   | Quadratic2D[a, b, c, d, e, f]  |
| 2 | $\texttt{Polynomial2D[Line2D[}a, b, c], \{x, y\}] \Rightarrow ax + by + c$         |
|   | $\texttt{Polynomial2D[Quadratic2D[}a,  b,  c,  d,  e,  f], \{x,  y\}] \Rightarrow$ |
|   | $ax^2 + bxy + cy^2 + dx + ey + f$  |
| 3 | $\texttt{Equation2D[Line2D[}a, b, c], \{x, y\}] \Rightarrow ax + by + c == 0$      |
|   | $\texttt{Equation2D[Quadratic2D[}a, b, c, d, e, f], \{x, y\}] \Rightarrow$         |
|   | $ax^2 + bxy + cy^2 + dx + ey + f == 0$   |
| 4 | $\texttt{Line2D}[ax + by + c == 0, \{x, y\}] \Rightarrow \texttt{Line2D}[a, b, c]$ |
|   | $\texttt{Quadratic2D}[ax^2 + bxy + cy^2 + dx + ey + f == 0, \{x, y\}] \Rightarrow$ |
|   | Quadratic2D[a, b, c, d, e, f]  |

Table 4.3: Descarta2D conversion functions.

```
In[3]: Clear[x, y];
    {Equation2D[Line2D[2, 3, -1], {x, y}],
        Equation2D[Quadratic2D[1, -2, 2, 3, -3, 7], {x, y}]}
Out[3] {-1+2x+3y == 0, 7+3x+x<sup>2</sup>-3y-2xy+2y<sup>2</sup> == 0}
```

#### 

## 4.4 Solving Equations

In our study of analytic geometry we will often need to solve linear and quadratic equations. We will also need to solve systems of two or more equations. *Mathematica* provides functions for solving individual equations and systems of equations, either exactly (the Solve function) or numerically (the NSolve function). The following subsections illustrate the use of these *Mathematica* functions.

#### One Linear, One Unknown

The equation ax + b = 0 is a linear equation in one unknown. By simple algebra, the solution to this equation is x = -b/a. The equation is invalid (or trivial) and has no solution if a = 0.

F

F

**Example.** Solve the equation 3x + 12 = 0.

**Solution.** The Mathematica function Solve[eqn, variable] returns a list of solutions for an equation in one unknown. The solution(s) are returned in the form of Mathematica rules.

```
\label{eq:In[4]: Clear[x]; Solve[3x+12 == 0, x] \\ Out[4] ~ \{ \{ x \to -4 \} \}
```

### One Quadratic, One Unknown

The quadratic equation  $ax^2 + bx + c = 0$  has two solutions

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The expression under the radical,  $D \equiv b^2 - 4ac$ , is called the *discriminant* of the equation and determines the type of solutions admitted by the equation. Assuming the coefficients are real numbers, D > 0 indicates that the equation has two real and distinct solutions; if D = 0 the equation has two real solutions that are equal; and if D < 0 the equation has two complex solutions that are conjugates of each other.

**Example.** Find the solutions of the equation  $3x^2 - 4x - 5 = 0$ .

**Solution.** The Mathematica function Solve[eqn, variable] returns a list of solutions for an equation in one unknown. The solution(s) are returned in the form of Mathematica rules.

\_\_\_\_\_

#### Two Linears, Two Unknowns

A list of two or more equations that are to be solved *simultaneously* is called a *system* of equations. Consider the system of two linear equations

 $a_1x + b_1y + c_1 = 0$  and  $a_2x + b_2y + c_2 = 0$ .

Simple algebra yields the formulas for x and y that solve the two equations:

$$x = \frac{b_1c_2 - b_2c_1}{a_1b_2 - a_2b_1}$$
 and  $y = \frac{a_2c_1 - a_1c_2}{a_1b_2 - a_2b_1}$ .

If the denominator,  $a_1b_2 - a_2b_1$ , is equal to zero the equations have no solution and are called *inconsistent*.

**Example.** Find the solution of the two linear equations x - 3y + 4 = 0 and 2x + 5y - 3 = 0.

**Solution.** The Mathematica function Solve[eqnList, varList] returns a list of solutions for a system of equations in several variables. The solution(s) are returned in the form of Mathematica rules.

```
In[6]: Clear[x, y];
Solve[{x - 3y + 4 == 0, 2x + 5y - 3 == 0}, {x, y}]
Out[6] {{x → -1, y → 1}}
```

#### One Linear, One Quadratic, Two Unknowns

Consider the linear and quadratic equations

$$a_1x + b_1y + c_1 = 0$$
 and  
 $a_2x^2 + b_2xy + c_2y^2 + d_2x + e_2y + f_2 = 0.$ 

In the general case the system of these two equations can be solved by first solving the linear equation for one of the variables, say x, in terms of the other, y. The expression for x is then substituted into the quadratic equation, yielding a somewhat more complicated quadratic equation in y alone. The quadratic equation in one variable is then solved yielding two values for y which may then be substituted back into the linear equation to determine the corresponding values of x. While this solution technique is straightforward, it produces somewhat complicated expressions for x and y, and special cases must be handled individually (for example, if the linear equation has no y term, then the procedure must be altered to solve for x instead).

**Example.** Solve the system of equations

$$3x + 4y - 1 = 0$$
 and  $2x^2 + y^2 + 6x - 4y + 1 = 0$ 

using the *Mathematica* Solve command.

**Solution.** The Mathematica function Solve[eqnList, varList] returns a list of solutions for a system of equations in several variables. The solution(s) are returned in the form of Mathematica rules.

```
In[7]: Clear[x, y];
ans = Solve[{3x+4y-1 == 0, 2x^2+y^2+6x-4y+1 == 0}, {x, y}]

Out[7] {{x \to \frac{1}{41} (-69 - 4\sqrt{295}), y \to \frac{1}{41} (62 + 3\sqrt{295})},

{x \to \frac{1}{41} (-69 + 4\sqrt{295}), y \to \frac{1}{41} (62 - 3\sqrt{295})}}
```

These somewhat complicated solutions can be approximated by decimal numbers using the *Mathematica*  $\mathbb{N}$  function.

```
\label{eq:integral} \begin{array}{l} \mbox{In[8]: $N[ans]$} \\ \mbox{out[8] } \{ \{ x \rightarrow -3.35859, \ y \rightarrow 2.76894 \}, \ \{ x \rightarrow -0.00726205, \ y \rightarrow 0.255447 \} \} \end{array}
```

#### Two Quadratics, Two Unknowns

E

The system of two quadratic equations in two unknowns

$$a_1x^2 + b_1xy + c_1y^2 + d_1x + e_1y + f_1 = 0$$
 and  
 $a_2x^2 + b_2xy + c_2y^2 + d_2x + e_2y + f_2 = 0$ 

can be solved algebraically using a technique involving a *pencil* of the two quadratic equations. This technique will be discussed in more detail in later chapters. Even though the technique can yield a symbolic formula for the solutions, such a formula is of no practical value, and is riddled with special cases. In spite of these complications, *Mathematica* can solve such systems of equations with numerical coefficients, both in exact form and approximated numerically. These solutions are very useful in the study of conic curves introduced in later chapters.

Example. Find approximate numerical solutions for the system of equations

 $3x^{2} + 2xy - 4y^{2} - 2x - 3y - 4 = 0$  $x^{2} - 4xy + y^{2} + 3x + 4y + 1 = 0$ 

using the Mathematica NSolve command.

**Solution.** The Mathematica function NSolve [eqnList, varList] returns a list of numerical solutions for a system of equations in several variables. The solution(s) are returned in the form of Mathematica rules. The results shown here were computed using Mathematica Version 3.0.1. Version 4.0 computes the same roots, but returns them in a different order.

```
 \begin{array}{l} \mbox{In[9]: Clear[x, y];} \\ \mbox{NSolve[{3x^2 + 2x * y - 4y^2 - 2x - 3y - 4 == 0,} \\ \mbox{x^2 - 4x * y + y^2 + 3x + 4y + 1 == 0}, {x, y}] \\ \mbox{Out[9] } { \{x \to -0.955121, y \to 0.120031\},} \\ { \{x \to 0.476004 - 0.298543 \, I, y \to -0.268381 + 0.962235 \, I\},} \\ { \{x \to 0.476004 + 0.298543 \, I, y \to -0.268381 - 0.962235 \, I\},} \\ { \{x \to 3.81264, y \to 3.46435\}} \\ \end{array}
```

Notice that in this example two of the solution pairs involve only real numbers, and two involve complex numbers. The complex solutions are a conjugate pair.

**Descarta2D Hint.** Descarta2D provides the function Solve2D to supplement the capabilities of the Mathematica Solve function. It provides specialized capabilities that are useful in the implementation of the Descarta2D packages. Refer to the Descarta2D references for a detailed description of the Solve2D function.

## 4.5 Graphs

Consider that F(x, y) = 0 has been solved for y so that y = f(x). We wish to give a geometric interpretation to the equation y = f(x). Now if a value, say  $x_1$ , is assigned to x, then, if f(x)is single-valued, there will be determined a single value y, say  $y_1$ . Another value of x, say  $x_2$ , will produce a value  $y_2$ . If f(x) is multiple-valued, there will be several values of y for a given x. In any event the real number pairs  $(x_1, y_1)$  which satisfy y = f(x) may be plotted in two dimensions as points in the plane. The aggregate of these points constitutes the graph or plot of the equation y = f(x) or of the function f(x).

This is one of the central problems in plane analytic geometry: given a function y = f(x), to plot its graph or to represent it geometrically. We sometimes say that the graph of f(x) is the *locus* of f(x). The word *locus*, in general, carries with it the idea of motion. Thus, the curve traced by a moving point is called the locus of the point. Such a locus is also referred to as a *curve* in the plane.

Through the study of *equations* much can be learned about the geometric properties of *graphs*. Such analysis is one of the roles of analytic geometry. In the study of an equation y = f(x) there are many analyses that can be made in order to intuitively understand the behavior of the graph. *Mathematica* and *Descarta2D* can be used to aid in this understanding. Four properties of significant interest in analytic geometry are

**Intercepts** The points at which the curve crosses the x- and y-axes.

- **Extent** The regions of the plane to which the curve is confined and regions where it tends to infinity.
- **Symmetry** The lines in which the reflection of the curve is a mirror image of the curve itself. Cases of interest include symmetry about the x- or y-axes, symmetry about the origin, and symmetry about the lines y = x or y = -x.
- Asymptotes The behavior of an unbounded curve in the neighborhood of infinity, where either x, y, or both become infinite. In particular, it may happen that the distance from a point P on the curve to some fixed line tends to zero. Such a line is called an *asymptote* of the curve.

The set of all points which satisfy a given condition is called the *locus* of that condition. An *equation* is called the equation of the locus if it is satisfied by the coordinates of every point on the locus and by no other points. There are three common representations of the locus by means of equations:

**Rectangular equations** which involve the rectangular coordinates (x, y)

**Polar equations** which involve the polar coordinates  $(r, \theta)$ 

**Parametric equations** which express x and y (or r and  $\theta$ ) in terms of a third independent variable called a *parameter*.

This book focuses on rectangular and parametric equations, with polar equations covered in the explorations.

## 4.6 Parametric Equations

It is often advantageous to use two equations to represent a curve instead of one. The xcoordinate of a point on the curve will be given by one equation expressing x as some function of a parameter, say  $\theta$  or t, and the y-coordinate will be given by another equation expressing y as a function of the same parameter. Such equations are called parametric equations. Upon eliminating the parameter between the two equations the *implicit* equation, in the form f(x, y) = 0, of the curve may be found. Some loci problems are treated most readily by means of parametric equations. Parametric equations are also the most natural means for generating a sequence of points on a curve, such as those needed to plot the curve. Since a parameter may be chosen in many ways, the parametric equations of a given curve are not unique, and in some cases they will only represent a portion of a curve.

**Example.** Find parametric equations of the locus of a point as it "orbits" about the origin at a distance of 2 units.

**Solution.** Let the parameter  $\theta$  be the angle measured counter-clockwise from the +x-axis that a line segment of length 2 sweeps when anchored at the origin (0,0). Using trigonometry the x- and y-coordinates of the end point of the line segment are given by the parametric equations

 $x = 2\sin\theta$  and  $y = 2\cos\theta$ .

The locus of these parametric equations is a circle. In *Mathematica* a parametric curve may be plotted using ParametricPlot[ $\{x(t), y(t)\}$ ,  $\{t, t_1, t_2\}$ ] where x(t) and y(t) are the parametric equations of the curve, t is the parameter, and  $t_1$  and  $t_2$  are the start and end values of the parameter.

```
In[10]: Clear[t];
ParametricPlot[{2Sin[t], 2Cos[t]}, {t, 0, 2Pi},
AspectRatio -> Automatic];
```



In our study of curves in the plane we will examine both implicit and parametric equations for the curves.

## 4.7 Explorations

Determinants often provide a concise notation for expressing relationships in analytic geometry. Show that the expanded algebraic form for the  $2 \times 2$  determinant

$$\begin{array}{ccc} a_1 & b_1 \\ a_2 & b_2 \end{array}$$

is given by  $-a_2b_1 + a_1b_2$ . Show that the expanded algebraic form for the  $3 \times 3$  determinant

.

is given by  $-a_3b_2c_1 + a_2b_3c_1 + a_3b_1c_2 - a_1b_3c_2 - a_2b_1c_3 + a_1b_2c_3$ .

CRAMER'S RULE (TWO EQUATIONS).....cramer2.nb

Show that the solution to the system of two linear equations in two unknowns

$$a_1x + b_1y + c_1 = 0a_2x + b_2y + c_2 = 0$$

is given by the determinants

$$x = \frac{\begin{vmatrix} -c_1 & b_1 \\ -c_2 & b_2 \end{vmatrix}}{D} \text{ and } y = \frac{\begin{vmatrix} a_1 & -c_1 \\ a_2 & -c_2 \end{vmatrix}}{D},$$
$$D = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}.$$

where

CRAMER'S RULE (THREE EQUATIONS).....cramer3.nb Show that the solution to the system of three linear equations in three unknowns

$$a_1x + b_1y + c_1z + d_1 = 0$$
  

$$a_2x + b_2y + c_2z + d_2 = 0$$
  

$$a_3x + b_3y + c_3z + d_3 = 0$$

is given by the determinants

$$x = \frac{\begin{vmatrix} -d_1 & b_1 & c_1 \\ -d_2 & b_2 & c_2 \\ -d_3 & b_3 & c_3 \end{vmatrix}}{D}, \quad y = \frac{\begin{vmatrix} a_1 & -d_1 & c_1 \\ a_2 & -d_2 & c_2 \\ a_3 & -d_3 & c_3 \end{vmatrix}}{D}, \quad \text{and} \quad z = \frac{\begin{vmatrix} a_1 & b_1 & -d_1 \\ a_2 & b_2 & -d_2 \\ a_3 & b_3 & -d_3 \end{vmatrix}}{D}$$

where

|     | $a_1$ | $b_1$ | $c_1$ |  |
|-----|-------|-------|-------|--|
| D = | $a_2$ | $b_2$ | $c_2$ |  |
|     | $a_3$ | $b_3$ | $c_3$ |  |

POLAR EQUATIONS......polareqn.nb

A curve in polar coordinates may have more than one equation. A given point may have either of two general coordinate representations

$$(r, \theta + 2k\pi),$$
  
$$(-r, \theta + (2k+1)\pi),$$

for any integer k. Hence a given curve  $r = f(\theta)$  may have either of the two equation forms

$$\begin{array}{rcl} r &=& f(\theta+2k\pi),\\ -r &=& f(\theta+(2k+1)\pi). \end{array}$$

The first equation reduces to  $r = f(\theta)$  when k = 0, but may lead to an entirely different equation of the same curve for another value of k. Similarly, the second equation may yield other equations of the curve. Show that in spite of the potential for multiple equations in polar coordinates, a linear equation Ax + By + C = 0 has only one representation in polar coordinates given by

 $r(A\cos\theta + B\sin\theta) + C = 0.$ 

## Chapter 5

## **Lines and Line Segments**

The curve with the simplest equation is a straight line. There are many forms the equation can exhibit, depending on how we wish to construct the line. This chapter develops in detail the analytic geometry of a line and its close relation, the line segment.

## 5.1 General Equation

F

F

Every linear equation in two unknowns can be written in the form

$$Ax + By + C = 0$$

The graph of such a linear equation is a *straight line*. In Descarta2D the line Ax + By + C = 0 is represented as Line2D[A, B, C]. Points (x, y) whose coordinates satisfy the equation Ax + By + C = 0 are said to be on the line.

A line segment is the set of points on a line between two points on the line,  $P_0(x_0, y_0)$  and  $P_1(x_1, y_1)$ . In Descarta2D a line segment is represented as Segment2D[coords, coords] where the coords are lists of the (x, y) coordinates of the start and end points of the line segment.

**Example.** Plot the lines 2x - 3y + 1 = 0 and x + 2y + 2 = 0. Plot the line segment between the points (-1, 2) and (3, -1).

**Solution.** The Descarta2D function Sketch2D[objList] plots a sketch of the objects in the object list.



**Example.** Determine which of the points (-1, 1),  $(2, \frac{5}{3})$ , (3, 1) and  $(-3, -\frac{5}{3})$  are on the line 2x - 3y + 1 = 0.

**Solution.** The Descarta2D function IsOn2D[point, line] returns True if the point is on the line.

```
In[2]: 11 = Line2D[2, -3, 1];
In[3]: {IsOn2D[p1 = Point2D[{-1, 1}], 11],
       IsOn2D[p2 = Point2D[{2, 5/3}], 11],
       IsOn2D[p3 = Point2D[{3, 1}], 11],
       IsOn2D[p4 = Point2D[{-3, -5/3}], 11]}
Out[3] {False, True, False, True}
In[4]: Sketch2D[{l1, p1, p3}];
      Sketch2D[{11, p2, p4}];
          3
                                               3
          2
                                               2
          1
                                               1
                       •
                                     •
          0
                                               0
         -1
                                              -1
         -2
                                               -2
                                                                       2
                   -2
                                 2
                                        4
                                                        -2
                                                                0
                                                                              4
            -4
                          0
                                                  -4
```

F

F



Figure 5.1: Inclination and slope of a line.

#### **Inclination and Slope**

The angle,  $\theta$ , measured counter-clockwise from the +x-axis to a line, is called the *inclination* of the line. The tangent of this angle,  $\tan \theta$ , (generally designated by the letter m) is called the *slope* of the line. It is evident from Figure 5.1 that the slope of line  $P_1P_2$  is given by

$$m = \tan \theta = \frac{y_2 - y_1}{x_2 - x_1}$$

The formula is independent of the position and order of the two points involved.

Let  $L \equiv Ax + By + C = 0$  be the general equation of a line. It is clear that the points (-C/A, 0) and (0, -C/B) are on the line since they satisfy the equation of the line. Therefore, the slope of L is given by

$$m = \frac{0 - (-C/B)}{(-C/A) - 0} = -\frac{A}{B}$$

and the angle of inclination,  $\theta = \tan^{-1} (-A/B)$ .

The slope of the line containing a line segment from point  $(x_0, y_0)$  to point  $(x_1, y_1)$  can be determined directly from the formula given for lines as

$$m = \frac{y_1 - y_0}{x_1 - x_0}$$

**Example.** Find the angle of inclination (in degrees) and the slope of the line x - y + 4 = 0. Find the slope of the line segment between the points (-2, 1) and (3, 2).

**Solution.** The Descarta2D function Angle2D[*line*] returns the inclination of a line (in radians); the function Slope2D[*line*] returns the slope of a line. The function Slope2D[*lnseg*] returns the slope of the line containing the line segment.

Mathematica Hint. The Mathematica symbol Degree equals the constant  $\pi/180$ . In the previous example dividing by Degree converts the angle from radians to degrees. The Mathematica function N[expr] produces a numerical approximation of an expression. The syntax expr //N is equivalent to N[expr].

## 5.2 Parallel and Perpendicular Lines

If two lines have the same slope they are called *parallel lines*. If two lines share all their points they are said to be *coincident*; coincident lines are also considered to be parallel. Two lines are *perpendicular* if the angle between them is a right angle. Let  $m_1 = \tan \theta_1$  and  $m_2 = \tan \theta_2$  be the slopes of two perpendicular lines. Since

$$\theta_2 = \theta_1 \pm \frac{\pi}{2}$$
$$\tan \theta_2 = \tan \left(\theta_1 \pm \frac{\pi}{2}\right)$$
$$m_2 = -\cot \theta_1$$
$$= -\frac{1}{\tan \theta_1}$$
$$= -\frac{1}{m_1}.$$

Therefore, the slopes of two *perpendicular lines* are negative reciprocals of each other related by the equation,  $m_1 = -1/m_2$ . Descarta2D provides functions for querying whether pairs of lines are parallel or perpendicular.

**Example.** Determine which of the following pairs of lines are parallel:

(a) 2x - 3y + 4 = 0 and -4x + 6y - 3 = 0,

F

F

(b) x + 2y - 3 = 0 and -2x + y - 1 = 0, and

(c) 3x - 4y + 2 = 0 and 2x + 4y - 1 = 0.

Additionally, determine which pairs are perpendicular.
**Solution.** The function IsParallel2D[*line*, *line*] will return True if the two lines are parallel; otherwise, it returns False. IsPerpendicular2D[*line*, *line*] returns True if the two lines are perpendicular; otherwise, it returns False.

```
In[6]: l1 = Line2D[2, -3, 4]; l2 = Line2D[-4, 6, -3];
l3 = Line2D[1, 2, -3]; l4 = Line2D[-2, 1, -1];
l5 = Line2D[3, -4, 2]; l6 = Line2D[2, 4, -1];
In[7]: {{IsParallel2D[11, 12], IsPerpendicular2D[11, 12]},
{IsParallel2D[13, 14], IsPerpendicular2D[13, 14]},
{IsParallel2D[15, 16], IsPerpendicular2D[15, 16]}}
Out[7] {{True, False}, {False, True}, {False, False}}
```

Therefore, the lines in pair (a) are parallel, the lines in pair (b) are perpendicular, and the lines in pair (c) are neither parallel or perpendicular.



# 5.3 Angle between Lines

The angle between two non-intersecting (parallel or coincident) lines is zero (radians or degrees). In the case of two intersecting lines,  $L_1$  and  $L_2$ , let  $\theta_{12}$  be the angle between the lines measured counter-clockwise from  $L_1$  to  $L_2$ . Since  $\theta_{12} = \theta_2 - \theta_1$ , it follows that

$$\tan \theta_{12} = \tan \left(\theta_2 - \theta_1\right) = \frac{\tan \theta_2 - \tan \theta_1}{1 + \tan \theta_1 \tan \theta_2}$$

which, in terms of slopes of the lines, yields

$$\tan \theta_{12} = \frac{m_2 - m_1}{1 + m_1 m_2}.$$

-

**Example.** Determine the angle (in radians) between the lines x + 3y - 4 = 0 and -2x + 2y + 1 = 0.

**Solution.** The Descarta2D function Angle2D [*line*, *line*] returns the angle between the two lines (measured in radians from the first line to the second line).

In[9]: Angle2D[11 = Line2D[1, 3, -4], 12 = Line2D[-2, 2, 1]] // N
Out[9] 1.10715

The result, 1.10715 radians, is approximately 63.4349°.

```
In[10]: Sketch2D[{11, 12}];
```



```
In[11]: Angle2D[12, 11] // N
```

Out[11] 2.03444

The angle between the lines taken in the opposite order is 2.0344 radians (approximately  $116.565^{\circ}$ ) which is the supplement of the first angle  $(63.435^{\circ} + 116.565^{\circ} = 180^{\circ})$ .

#### 

# 5.4 Two-Point Form

A line is determined by two distinct points on it,  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$ . Let P(x, y) be any other point on the line as illustrated in Figure 5.2. Then by similar triangles

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}$$

which is called the *two-point form* of a line. The two-point form may also be written as

$$(x - x_1)(y - y_2) = (x - x_2)(y - y_1).$$

F

L\_\_\_\_\_



Figure 5.2: Two-point form of a line.

In general form the line is given by

F

$$-(y_2 - y_1) x + (x_2 - x_1) y + x_1 y_2 - x_2 y_1 = 0.$$

In determinant form the equation is given by

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = 0$$

**Example.** Determine the line through the points (-2, -1) and (3, 2).

**Solution.** The Descarta2D function Line2D[point, point] constructs the line through the two points. Alternately, the function Line2D[lnseg] constructs a line defined by the start and end points of a line segment.

```
In[12]: p1 = Point2D[{-2, -1}];
    p2 = Point2D[{3, 2}];
    {11 = Line2D[p1, p2], 12 = Line2D[Segment2D[p1, p2]]}
Out[12] {Line2D[-3, 5, -1], Line2D[-3, 5, -1]}
In[13]: Sketch2D[{11, p1, p2}];
```



The Descarta2D function Line2D[ $\{x_1, y_1\}, \{x_2, y_2\}$ ] is also provided to allow construction of a line by specifying two point coordinates.

In[14]: Line2D[{-2, -1}, {3, 2}]
Out[14] Line2D[-3, 5, -1]

#### **Collinear Points**

In a previous chapter it was demonstrated that the three points  $P_1(x_1, y_1)$ ,  $P_2(x_2, y_2)$  and  $P_3(x_3, y_3)$  are collinear if their coordinates satisfy the determinant equation

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0.$$

This condition may be stated in a more intuitive form using the two–point form of a line. The line defined by  $P_1$  and  $P_2$  must be satisfied by  $P_3$  yielding the condition

$$-(y_2 - y_1) x_3 + (x_2 - x_1) y_3 + x_1 y_2 - x_2 y_1 = 0$$

which can be put into the more symmetrical form

$$y_1(x_2 - x_3) + y_2(x_3 - x_1) + y_3(x_1 - x_2) = 0.$$

# 5.5 Point-Slope Form

Since  $m = \frac{y_2 - y_1}{x_2 - x_1}$ , the two-point form of a line can be reduced to the *point-slope form* 

$$y - y_1 = m\left(x - x_1\right)$$

as shown in Figure 5.3. In general form the equation of the line is

$$mx - y + (y_1 - mx_1) = 0.$$



Figure 5.3: Point-slope form of a line.

A vertical line cannot be represented in point–slope form. In determinant form the point–slope form is given by

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ 1 & m & 0 \end{vmatrix} = 0.$$

**Example.** Determine the line through the point (1, 2) with a slope of 1/2.

**Solution.** The Descarta2D function Line2D[point, m] constructs a line through the point with a given slope, m.

```
In[15]: 11 = Line2D[p1 = Point2D[{1, 2}], 1/2]
Out[15] Line2D[<sup>1</sup>/<sub>2</sub>, -1, <sup>3</sup>/<sub>2</sub>]
In[16]: Sketch2D[{p1, 11}];
```



7

F

\_\_\_\_\_

#### Line Through a Point Parallel to a Line

A line through a given point  $P_1(x_1, y_1)$  parallel to a given line

$$A_2x + B_2y + C_2 = 0$$

would have a slope  $m = -A_2/B_2$ , and using  $mx - y + (y_1 - mx_1) = 0$  yields

$$A_2x + B_2y - (A_2x_1 + B_2y_1) = 0.$$

The equation can also be written

$$B_2(x - x_1) = A_2(y - y_1).$$

In determinant form the equation is

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ B_2 & -A_2 & 0 \end{vmatrix} = 0$$

#### Line Through a Point Perpendicular to a Line

A line through a given point  $P_1(x_1, y_1)$  perpendicular to a given line

$$A_2x + B_2y + C_2 = 0$$

would have a slope  $m = -1/m_2 = B_2/A_2$ , and using  $mx - y + (y_1 - mx_1) = 0$  yields

$$B_2x - A_2y + (A_2y_1 - B_2x_1) = 0,$$

or, in a simpler form,

F

F

$$A_2(y - y_1) = B_2(x - x_1).$$

In determinant form the equation is

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ A_2 & B_2 & 0 \end{vmatrix} = 0.$$

**Example.** Find the lines through the point (2, 1) which are parallel and perpendicular to the line 3x - 2y + 1 = 0.

**Solution.** Line2D[*point*, *line*, Paralle12D] constructs a line through the point and parallel to the line and Line2D[*point*, *line*, Perpendicular2D] constructs a line through the point and perpendicular to the line.



**Descarta2D Hint.** The function Line2D[*point*, *line*] returns the same results as Line2D[*point*, *line*, Perpendicular2D]; the keyword Perpendicular2D is optional and may be omitted.

In[20]: Line2D[p1, 11]
Out[20] Line2D[-2, -3, 7]

#### Horizontal and Vertical Lines Through a Point

Given a point  $P_1(x_1, y_1)$ , a horizontal line whose slope is 0 will have the equation  $y - y_1 = 0$ . In determinant form the equation is

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ 1 & 0 & 0 \end{vmatrix} = 0.$$

Similarly, a vertical line (whose slope is infinite) has the equation  $x-x_1 = 0$  and its determinant equation is

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ 0 & 1 & 0 \end{vmatrix} = 0.$$

**Example.** Find the horizontal and vertical lines through the point (3, 2).

**Solution.** The function Line2D[*point*, 0] constructs a horizontal line through the point. The function Line2D[*point*, Infinity] constructs a vertical line through the point.

```
In[21]: pl = Point2D[{3, 2}];
        {l1 = Line2D[p1, 0], l2 = Line2D[p1, Infinity]}
Out[21] {Line2D[0, -1, 2], Line2D[1, 0, -3]}
In[22]: Sketch2D[{p1, l1, l2}];
```



L\_\_\_\_\_

#### 5.6 Slope–Intercept Form

Specializing the point  $P_1(x_1, y_1)$  in the point-slope form of a line to the y-intercept point (0, b) as shown in Figure 5.4 gives the *slope-intercept* form of a line y = mx + b. In general form the equation of the line is mx - y + b = 0. The slope-intercept form cannot be used to represent vertical lines. In determinant form the point-slope form is given by

```
\begin{vmatrix} x & y & 1 \\ 0 & b & 1 \\ 1 & m & 0 \end{vmatrix} = 0.
```

**Example.** Find the line with a *y*-intercept of 1 and a slope of 2.

**Solution.** The Descarta2D function Line2D[point, m] constructs a line through the point with the given slope.

F



Figure 5.4: Slope–intercept form of a line.

```
In[23]: 11 = Line2D[p1 = Point2D[{0, 1}], 2]
Out[23] Line2D[2, -1, 1]
```

```
In[24]: Sketch2D[{p1, 11}, PlotRange -> {{-3, 3}, {-3, 3}}];
```







used in the example above explicitly sets the minimum and maximum coordinate range along the x-axis and y-axis, overriding the default setting which is PlotRange->Automatic. The PlotRange option is useful for focusing on a specific portion of the plot.



Figure 5.5: Intercept form of a line.

# 5.7 Intercept Form

Specializing the two points in the two-point form to the intercepts (a, 0) and (0, b) as shown in Figure 5.5 gives (y - b)/x = -b/a, or, rearranging, the *intercept form* 

$$\frac{x}{a} + \frac{y}{b} = 1.$$

In general form the equation of the line is bx + ay - ab = 0; or, dividing Ax + By + C = 0 by  $C \ (C \neq 0)$  gives

$$\frac{x}{(-C/A)} + \frac{y}{(-C/B)} = 1.$$

Thus, in the general equation, the intercepts are given by x = -C/A and y = -C/B. Notice that a line in intercept form cannot pass through the origin, nor can it be horizontal or vertical. In determinant form the intercept form is given by

$$\begin{vmatrix} x & y & 1 \\ a & 0 & 1 \\ 0 & b & 1 \end{vmatrix} = 0.$$

**Example.** Find the line whose x-intercept is 2 and y-intercept is 1.

**Solution.** The function Line2D[*point*, *point*] constructs a line through the two points.

```
In[25]: p1 = Point2D[{2, 0}];
    p2 = Point2D[{0, 1}];
        112 = Line2D[p1, p2]
Out[25] Line2D[-1, -2, 2]
```

L\_\_\_\_\_

In[26]: Sketch2D[{p1, p2, 112}];



#### 5.8 Normal Form

Consider a directed line segment OA of length  $\rho$  starting at the origin O and making an angle  $\theta$  with the +x-axis as shown in Figure 5.6. The line L which is perpendicular to OA and passes through A is completely determined by the parameters  $\rho$  and  $\theta$ . We wish to determine the general equation of the line L. The coordinates of A are  $(\rho \cos \theta, \rho \sin \theta)$  and the slope of L is  $-\cot \theta$  since L is perpendicular to OA which has slope  $\tan \theta$ . Hence, using the point-slope form we obtain, as the *normal form* of the equation of line L,

$$y - \rho \sin \theta = -\cot \theta (x - \rho \cos \theta)$$

which reduces to

F

 $x\cos\theta + y\sin\theta - \rho = 0.$ 

This form of the equation of a straight line is called the *normal form* (sometimes the *perpendicular form*) because its coefficients involve the parameters  $\rho$  and  $\theta$  associated with the *normal* or *perpendicular* segment OA to the line.

To determine the coefficients of the normal form from the general form

$$Ax + By + C = 0,$$

we divide by  $\pm \sqrt{A^2 + B^2}$  yielding

$$\frac{A}{\pm\sqrt{A^2+B^2}}x + \frac{B}{\pm\sqrt{A^2+B^2}}y + \frac{C}{\pm\sqrt{A^2+B^2}} = 0.$$

The sign of  $\sqrt{A^2 + B^2}$  is chosen to be opposite to that of C to make the constant term,  $\rho$ , positive. If C is zero, the line passes through the origin. The process of dividing a linear equation by  $\pm \sqrt{A^2 + B^2}$  is called *normalizing* the line.

**Example.** Normalize the lines 3x - 4y - 5 = 0 and 2x + y - 3 = 0.



Figure 5.6: Normal form of a line.

**Solution.** The Descarta2D function Line2D[*line*] constructs a line with normalized coefficients.

```
In[27]: \{ Line2D[Line2D[3, -4, -5]], Line2D[Line2D[2, 1, -3]] \}
Out[27] \{ Line2D[\frac{3}{5}, -\frac{4}{5}, -1], Line2D[\frac{2}{\sqrt{5}}, \frac{1}{\sqrt{5}}, -\frac{3}{\sqrt{5}}] \}
```

**Example.** Find the line 4 units from the origin whose normal makes an angle of  $30^{\circ}$  with the positive x-axis.

**Solution.** We apply directly the normal form of a line to determine the coefficients of the line in general form.

```
In[28]: Line2D[Cos[30 Degree], Sin[30 Degree], -4] // N
Out[28] Line2D[0.866025, 0.5, -4.]
```

Mathematica Hint. The Mathematica symbol Degree is the constant  $\pi/180$ . Multiplying an angle in degrees by Degree (as illustrated in the previous example) converts the angle to radians; radians are the angular units required in all Descarta2D functions.

F

#### Point Offset a Distance Along a Line

Given a point  $P_1(x_1, y_1)$  we wish to offset the point a distance d in the direction of a given line  $L_2 \equiv A_2x + B_2y + C_2 = 0$ . We note that the coefficients of the normalized form of  $L_2$ immediately give us the unit directions to offset  $P_1$ , so the desired coordinates of the offset point are

$$\left(x_1 + \frac{dA_2}{\sqrt{A_2^2 + B_2^2}}, y_1 + \frac{dB_2}{\sqrt{A_2^2 + B_2^2}}\right)$$

If the point  $P_1$  is on line  $L_2$ , then the offset point will also be on  $L_2$ ; otherwise, the offset point will be on a line parallel to  $L_2$ . The distance d may be positive or negative allowing offsets in either direction parallel to the line.

**Example.** Offset the points (-1, 1), (1, -1) and (0, 0) a distance 2 in both directions along the line 3x - 4y + 1 = 0.

**Solution.** The Descarta2D function Point2D [point, line, d] offsets a point along a line a given distance, d. The distance may be positive or negative.

```
In[29]: p1 = Point2D[\{-1, 1\}];
p2 = Point2D[\{1, -1\}];
p3 = Point2D[\{0, 0\}];
l1 = Line2D[3, -4, 1];
In[30]: pts = \{\{Point2D[p1, 11, 2], Point2D[p1, 11, -2]\},
\{Point2D[p2, 11, 2], Point2D[p2, 11, -2]\},
\{Point2D[q3, 11, 2], Point2D[q3, 11, -2]\}\}
Out[30] \{\{Point2D[\{\frac{3}{5}, \frac{11}{5}\}], Point2D[\{-\frac{13}{5}, -\frac{1}{5}\}]\},
\{Point2D[\{\frac{13}{5}, \frac{1}{5}\}], Point2D[\{-\frac{3}{5}, -\frac{11}{5}\}]\},
\{Point2D[\{\frac{8}{5}, \frac{6}{5}\}], Point2D[\{-\frac{8}{5}, -\frac{6}{5}\}]\}\}
```

```
In[31]: Sketch2D[{p1, p2, p3, l1, pts}];
```



#### Line Offset a Distance from a Line

If a line  $L_1$  is parallel to a second line  $L_2$ , the two lines will be separated by a constant distance, d. The process of constructing a line such as  $L_2$  which is parallel to  $L_1$  at a given distance, d, is called *offsetting* the line. There are two lines offset a distance d from a line Ax + By + C = 0. The general equations of these two lines are easily determined from the normal form as

$$\frac{A}{\sqrt{A^2 + B^2}}x + \frac{B}{\sqrt{A^2 + B^2}}y + \frac{C}{\sqrt{A^2 + B^2}} \pm d = 0.$$

**Example.** Find and plot the two lines offset a distance of two units from the line x - 3y + 1 = 0.

**Solution.** The Descarta2D function Line2D[line, d] constructs a line offset a given distance, d, from a line. The distance may be positive or negative yielding one of the two possible offset lines.

```
In[32]: 11 = Line2D[1, -3, 1];
{12 = Line2D[11, 2], 13 = Line2D[11, -2]}
```

```
Out[32] {Line2D[1, -3, 1 - 2\sqrt{10}], Line2D[1, -3, 1 + 2\sqrt{10}]}
```

```
In[33]: Sketch2D[{11, 12, 13}];
```



Distance from a Point to a Line

The normal form of a line provides a convenient method for determining the distance from a point to a line. Consider a normalized line  $L \equiv px + qy - r = 0$ , where  $p^2 + q^2 = 1$ . A line M offset a distance d from L clearly has the equation  $M \equiv px + qy - r \pm d = 0$ . Any point  $P_1(x_1, y_1)$  on M satisfies the equation of M, therefore,  $px_1 + qy_1 - r \pm d = 0$ . Solving for d and squaring to remove the ambiguous sign yields

$$d^2 = (px_1 + qy_1 - r)^2.$$

Ł\_\_\_\_\_

Thus, the distance d from a point  $P_1(x_1, y_1)$  to a line Ax + By + C = 0 in general form is

$$d = \pm \frac{Ax_1 + By_1 + C}{\sqrt{A^2 + B^2}},$$

where the sign is selected to produce the positive result.

**Example.** Find the distance from the point (3, -2) to the line 3x - 4y + 2 = 0.

**Solution.** The Descarta2D function Distance2D[point, line] returns the distance from the point to the line.

## 5.9 Intersection Point of Two Lines

Two lines  $L_1 \equiv A_1x + B_1y + C_1 = 0$  and  $L_2 \equiv A_2x + B_2y + C_2 = 0$  may be parallel, coincident, or *intersect* in a single point. In the case they where intersect in a single point, the coordinates of the point may be determined by solving the system of equations

$$A_1x + B_1y + C_1 = 0$$
$$A_2x + B_2y + C_2 = 0$$

for the intersection point P(x, y). The resulting formula for the coordinates of point P is

$$\left(\frac{B_1C_2 - B_2C_1}{A_1B_2 - A_2B_1}, \frac{A_1C_2 - A_2C_1}{A_1B_2 - A_2B_1}\right), \ A_1B_2 - A_2B_1 \neq 0.$$

In the case where the denominators are zero, the lines are either parallel or coincident. If the lines are *coincident* the ratio of their corresponding coefficients will be a constant

$$k = \frac{A_1}{A_2} = \frac{B_1}{B_2} = \frac{C_1}{C_2},$$

and the conditions

$$\begin{vmatrix} A_1 & B_1 \\ A_2 & B_2 \end{vmatrix} = 0, \begin{vmatrix} -C_1 & B_1 \\ -C_2 & B_2 \end{vmatrix} = 0 \text{ and } \begin{vmatrix} A_1 & -C_1 \\ A_2 & -C_2 \end{vmatrix} = 0$$

are sufficient to insure the lines are coincident.

\_\_\_\_\_

\_\_\_\_\_

**Example.** Find the intersection point of the two lines whose equations are 2x - 3y + 7 = 0 and 3x + 7y - 2 = 0.

**Solution.** The Descarta2D function Point2D[*line*, *line*] constructs the intersection point of the two lines.

```
In[35]: p12 = Point2D[11 = Line2D[2, -3, 7], 12 = Line2D[3, 7, -2]]Out[35] Point2D[\{-\frac{43}{23}, \frac{25}{23}\}]
```

In[36]: Sketch2D[{11, 12, p12}];



# 5.10 Point Projected Onto a Line

A point  $P_1$  is said to be *projected* onto a point  $P_2$  on a line  $L_2$ , if the line  $P_1P_2$  is perpendicular to  $L_2$ . To determine the coordinates of a point projected onto a line, we can build upon concepts and formulas already established. We construct a line through the point, perpendicular to the given line. This line is then intersected with the given line which yields the desired projected point. Using *Descarta2D*, the sequence of commands to project point  $P_1(x_1, y_1)$  onto the line  $L_2 \equiv A_2x + B_2y + C_2 = 0$  is as follows:

```
In[37]: Clear[x1, y1, A2, B2, C2];
p1 = Point2D[{x1, y1}];

l2 = Line2D[A2, B2, C2];

l1 = Line2D[p1, l2, Perpendicular2D];

p2 = Point2D[l1, l2] // Simplify

Out[37] Point2D[{\frac{B2^2 x1 - A2 (C2 + B2 y1)}{A2^2 + B2^2}}, \frac{-B2 (C2 + A2 x1) + A2^2 y1}{A2^2 + B2^2}}]
```

F

In standard mathematical notation the coordinates of the projected point  $P_2$  are

$$\left(\frac{B_2^2x_1 - A_2(C_2 + B_2y_1)}{A_2^2 + B_2^2}, \frac{A_2^2y_1 - B_2(C_2 + A_2x_1)}{A_2^2 + B_2^2}\right).$$

The coordinates of the projected point can also be written in a somewhat more intuitive form given by

$$(x_1 - ad, y_1 - bd)$$

where

E

L\_\_\_\_\_

$$a = \frac{A}{\sqrt{A^2 + B^2}}, \ b = \frac{B}{\sqrt{A^2 + B^2}} \ \text{and} \ d = \frac{Ax_1 + By_1 + C}{\sqrt{A^2 + B^2}}.$$

Simple algebra confirms that the two forms are equivalent. If the point  $P_1(x_1, y_1)$  is on the line, then it is clear from the second form that the projected point  $P_2$  has coordinates  $(x_1, y_1)$  since d, which is the signed distance from the point to the line, is equal to zero when  $P_1$  is on  $L_2$ . As shown in the next example, *Descarta2D* provides a specific function that projects a point onto a line.

**Example.** Project the point (-3, 2) onto the line 5x - 3y + 4 = 0.

**Solution.** The Descarta2D function Point2D[point, line] projects a point onto a line and returns the projected point.

```
In[38]: p2 = Point2D[p1 = Point2D[{-3, 2}], 12 = Line2D[5, -3, 4]]Out[38] Point2D[\{-\frac{1}{2}, \frac{1}{2}\}]
```

In[39]: Sketch2D[{p1, p2, 12}, PlotRange -> {{-4, 1}, {-1, 4}}];



## 5.11 Line Perpendicular to Line Segment

Given a line segment bounded by the points  $P_0(x_0, y_0)$  and  $P_1(x_1, y_1)$ , we wish to find the line that is the perpendicular bisector of the line segment. Using *Descarta2D* we merely construct the line perpendicular to the line segment through its midpoint.

In standard mathematical notation the equation of the line is

$$2(x_1 - x_0)x + 2(y_1 - y_0)y + x_0^2 - x_1^2 + y_0^2 - y_1^2 = 0.$$

In determinant form the equation is given by

$$\begin{vmatrix} x & y & 1 \\ x_0 + x_1 & y_0 + y_1 & 2 \\ y_0 - y_1 & -(x_0 - x_1) & 0 \end{vmatrix} = 0.$$

**Example.** Find the line that is the perpendicular bisector of the line segment bounded by the points (-3, -1) and (5, 3).

**Solution.** The function Line2D[*lnseg*, Perpendicular2D] constructs the perpendicular bisector of the line segment.

```
In[41]: ls1 = Segment2D[{-3, -1}, {5, 3}];
l1 = Line2D[ls1, Perpendicular2D]
```

```
Out[41] Line2D[16, 8, -24]
```

F

In[42]: Sketch2D[{ls1, l1, Point2D[{-3, -1}], Point2D[{5, 3}]}];



**Descarta2D Hint.** In the previous example, the resulting line 16x + 8y - 24 = 0, can be expressed in a simpler form by dividing the coefficients by 8, resulting in the equation 2x + y - 3 = 0. The Mathematica function Simplify[expr] (or expr //Simplify) can be used to simplify the result of any Descarta2D computation. Be aware, however, that the computation may take a significant amount of time to complete and, sometimes, no simpler expression is found. The Descarta2D Line2D object has a special Simplify function that removes common factors from the coefficients of a line.

```
In[43]: Line2D[16, 8, -24] // Simplify
Out[43] Line2D[2, 1, -3]
```

# 5.12 Angle Bisector Lines

The angle bisectors of two lines  $A_1x + B_1y + C_1 = 0$  and  $A_2x + B_2y + C_2 = 0$  are defined by the locus of points equidistant from the two lines. If P(x, y) is an arbitrary point on the angle bisectors, then using the distance formula for both lines yields

$$d = \pm \frac{A_1 x + B_1 y + C_1}{\sqrt{A_1^2 + B_1^2}}$$
 and  $d = \pm \frac{A_2 x + B_2 y + C_2}{\sqrt{A_2^2 + B_2^2}}$ .

Equating the two yields the equation of the angle bisectors given by

$$\frac{A_1x + B_1y + C_1}{\sqrt{A_1^2 + B_1^2}} = \pm \frac{A_2x + B_2y + C_2}{\sqrt{A_2^2 + B_2^2}}.$$

**Example.** Find and plot the angle bisector lines of the lines x - 3y + 2 = 0 and x + 4y - 2 = 0.

**Solution.** The Descarta2D function MedialLoci2D[{*line*, *line*}] returns a list of two lines that are the angle bisectors of the two given lines.

```
In[44]: 112 = MedialLoci2D[{11 = Line2D[1, -3, 2], 12 = Line2D[1, 4, -2]}]
```

```
\begin{array}{l} \textit{Out[44]} & \left\{ \texttt{Line2D} \left[ \sqrt{10} \ - \sqrt{17} \ , \ 4 \ \sqrt{10} \ + \ 3 \ \sqrt{17} \ , \ - \ 2 \ \sqrt{10} \ - \ 2 \ \sqrt{17} \ \right] \ , \\ & \texttt{Line2D} \left[ \sqrt{10} \ + \sqrt{17} \ , \ 4 \ \sqrt{10} \ - \ 3 \ \sqrt{17} \ , \ - \ 2 \ \sqrt{10} \ + \ 2 \ \sqrt{17} \ \right] \right\} \end{array}
```

In[45]: Sketch2D[{11, 12, 112}];

L\_\_\_\_\_



# 5.13 Concurrent Lines

Three lines that intersect in a single, common point are called *concurrent lines*. Using *Mathematica* we will prove that three lines

$$A_1x + B_1y + C_1 = 0$$
  

$$A_2x + B_2y + C_2 = 0$$
  

$$A_3x + B_3y + C_3 = 0$$

will be *concurrent* when the determinant of their coefficients is zero. The determinant equation is given by

$$\begin{vmatrix} A_1 & B_1 & C_1 \\ A_2 & B_2 & C_2 \\ A_3 & B_3 & C_3 \end{vmatrix} = 0.$$

We create four points,  $P_0$ ,  $P_1$ ,  $P_3$  and  $P_4$ , where  $P_0$  will be the common point of the three lines  $l_1 = P_0P_1$ ,  $l_2 = P_0P_2$  and  $l_3 = P_0P_3$ .

```
In[46]: Clear[x0, y0, x1, y1, x2, y2, x3, y3];
p0 = Point2D[{x0, y0}];
p1 = Point2D[{x1, y1}];
p2 = Point2D[{x2, y2}];
p3 = Point2D[{x2, y2}];
{11, 12, 13} = Map[Line2D[p0, #]&, {p1, p2, p3}]
Out[46] {Line2D[y0 - y1, -x0 + x1, -x1 y0 + x0 y1], Line2D[y0 - y2, -x0 + x2, -x2 y0 + x0 y2],
Line2D[y0 - y3, -x0 + x3, -x3 y0 + x0 y3]}
```

We now extract the coefficients of the lines and apply the postulated determinant.

```
In[47]: {A1, B1, C1} = List @@ 11;
    {A2, B2, C2} = List @@ 12;
    {A3, B3, C3} = List @@ 13;
    Det[{{A1, B1, C1},
        {A2, B2, C2},
        {A3, B3, C3}] // Simplify
Out[47] 0
```

F

F

The condition defined by the determinant is therefore necessary; it is also sufficient provided the slopes of the lines are distinct.

Mathematica Hint. The Mathematica function Apply[f, expr] (or f @@ expr) replaces the head of expr by f. In the Mathematica statements above, the Apply function is used to convert a Line2D object into a list of coefficients.

**Example.** Verify that the three lines given by x - y + 6 = 0, 2x + y - 5 = 0 and -x - 2y + 11 = 0 are concurrent.

**Solution.** The function IsConcurrent2D[*line*, *line*, *line*] returns True if the three lines are concurrent; otherwise, it returns False.

Out[48] True

In[49]: Sketch2D[{11, 12, 13}, CurveLength2D -> 12];



#### 5.14 Pencils of Lines

#### Pencil of Intersecting Lines

Let  $L_1 \equiv A_1x + B_1y + C_1 = 0$  and  $L_2 \equiv A_2x + B_2y + C_2 = 0$  be the equations of two lines in the plane. Consider the equation  $L \equiv (1 - k)L_1 + kL_2 = 0$ , where k is an arbitrary constant. L is clearly an equation of the first-degree as it can be written

$$((1-k)A_1 + kA_2)x + ((1-k)B_1 + kB_2)y + ((1-k)C_1 + kC_2) = 0.$$

Π

Assume that  $L_1$  and  $L_2$  intersect in some point P(x, y); then, by definition,  $L_1(x, y) = 0$  and  $L_2(x, y) = 0$ , since P is on both lines. Furthermore, L now represents a *family*, or *system*, of lines passing through P. Such a family of lines is called a *pencil* of lines. The variable k can be set to a value in a manner that produces a line in the pencil that will satisfy one additional condition. This is a useful technique for solving certain types of geometric problems as demonstrated in the next example.

**Example.** Find the family (pencil) of lines that pass through the intersection point of x - 2y + 4 = 0 and 2x + 3y - 2 = 0. Find the value of k and the associated member of the family that passes through the point (4, 2).

**Solution.** Line2D[*line*, *line*, *k*, Penci12D] constructs a line representing the pencil of lines  $(1 - k)L_1 + kL_2$ . Equation2D[*line*, *coords*] returns a *Mathematica* equation that establishes the condition of the point being on the line.

```
In[50]: Clear[k];
I3 = Line2D[L1 = Line2D[1, -2, 4],
L2 = Line2D[2, 3, -2], k, Pencil2D]
Out[50] Line2D[1 + k, -2 (1 - k) + 3 k, 4 (1 - k) - 2 k]
In[51]: pt = Point2D[{4, 2}];
eqn = Equation2D[L3, Coordinates2D[pt]]
Out[51] 4 (1 - k) - 2 k + 4 (1 + k) + 2 (-2 (1 - k) + 3 k) == 0
In[52]: ans = Solve[eqn]
Out[52] \{\{k \rightarrow -\frac{1}{2}\}\}
In[53]: Sketch2D[{L1, L2, (L3 /. ans[[1]]), pt}];
```



E



Figure 5.7: Pencil of lines through a point.

#### Pencil of Lines Through a Point

F

F

Consider the equation of a line L passing through points  $P_1(x_1, y_1)$  and P(x, y) as shown in Figure 5.7. L is parameterized by the angle  $\theta$  and the coordinates of P are given by

$$x = x_1 + \cos \theta$$
 and  $y = y_1 + \sin \theta$ .

The equation of the line as determined by the two-point form is given by

$$L \equiv -x\sin\theta + y\cos\theta + x_1\sin\theta - y_1\cos\theta = 0$$

Clearly, the point  $P_1(x_1, y_1)$  is on L as the coordinates of  $P_1$  satisfy the equation of L. Also, since the slope m of L is given by

$$m = \frac{-\sin\theta}{-\cos\theta} = \tan\theta$$

we can create a line with any given slope. A vertical line, whose slope is infinite, can be represented using  $\theta = \pi/2$ . A complete pencil of lines, therefore, can be created for values  $0 \le \theta < \pi$ .

**Example.** Find the parametric equation of a pencil of lines passing through the point (2,3).

**Solution.** The function Line2D[*point*,  $\theta$ , Pencil2D] constructs a line parameterized by angle  $\theta$  and passing through the given point.

In[54]: Clear[t]; 11 = Line2D[Point2D[{2, 3}], t, Pencil2D]

Out[54] Line2D[-Sin[t], Cos[t], -3 Cos[t] + 2 Sin[t]]

The following commands plot several members of this pencil of lines.

In[55]: Sketch2D[Map[(11 /. t -> #)&, Pi \* Range[0, 10] / 10]];



A simpler parameterization involves applying the point–slope form of a line and using the slope, m, as the parameter of the pencil. This approach yields the parameterized pencil of lines

 $mx - y + y_1 - mx_1 = 0.$ 

This parameterization, however, cannot represent vertical lines.

#### 5.15 Parametric Equations

We wish to formulate parametric equations for the line  $L_1 \equiv A_1 x + B_1 y + C_1 = 0$ . Since there are an infinite number of valid parameterizations, we will specify that we desire a particular parameterization with the properties that the point nearest the origin will be at parameter value t = 0, and the other points on the line will be parameterized by distance along the line. For example, the parameters  $t = \pm 2$  will generate the pair of points at a distance two from the point on the line nearest the origin.

Using Descarta2D we can determine the point  $P_0$  on  $L_1$  nearest the origin by projecting the origin onto  $L_1$ .

In[56]: Clear[A1, B1, C1];p0 = Point2D[Point2D[{0, 0}], Line2D[A1, B1, C1]]  $Out[56] Point2D[\left\{-\frac{A1 C1}{A1^2 + B1^2}, -\frac{B1 C1}{A1^2 + B1^2}\right\}]$ 

Now consider a right triangle with sides at and bt as shown in Figure 5.8. In this triangle  $a = A_1/\sqrt{A_1^2 + B_1^2}$  and  $b = B_1/\sqrt{A_1^2 + B_1^2}$ . The hypotenuse of this triangle is obviously of length t since

$$(at)^{2} + (bt)^{2} = \left(\frac{A_{1}t}{\sqrt{A_{1}^{2} + B_{1}^{2}}}\right)^{2} + \left(\frac{B_{1}t}{\sqrt{A_{1}^{2} + B_{1}^{2}}}\right)^{2} = t^{2}.$$



Figure 5.8: Parametric equation of a line.

Also the slope is given by

F

F

$$m = \frac{(y_0 - at) - y_0}{(x_0 + bt) - x_0} = -\frac{a}{b} = -\frac{A_1}{B_1}$$

which is the slope of the desired line,  $L_1$ . Therefore, the parametric equations of the line are

$$x = -\frac{A_1C_1}{A_1^2 + B_1^2} + \frac{B_1t}{\sqrt{A_1^2 + B_1^2}}$$
$$y = -\frac{B_1C_1}{A_1^2 + B_1^2} - \frac{A_1t}{\sqrt{A_1^2 + B_1^2}}$$

**Example.** Find the coordinates of the points on the line x - 2y + 3 = 0 for parameter values t = -2, -1, 0, 1, 2. Plot the lines and the points.

**Solution.** The Descarta<sub>2D</sub> function Line<sub>2D</sub>[A, B, C] [t] returns the coordinates of the point at parameter value t on the line.

```
In[58]: Sketch2D[{11, Map[Point2D[#]&, coords]}];
```



Mathematica Hint. The Mathematica function Map[f, expr] (or f / @ expr) applies to f to each element on the first level in expr. In the previous example Map is used to evaluate a line using a list of parameter values.

#### Line Segment

We wish to define the parametric equations for a line segment such that the parameter value t = 0 produces the coordinates of the start point  $P_0$ , t = 1 produces the coordinates of the end point  $P_1$ , and values 0 < t < 1 produce coordinates of points proportionally spaced in between  $P_0$  and  $P_1$ . Let d be the distance from  $P_0$  to a general point P(x, y) on the directed line  $P_0P_1$ . We use Descarta2D to produce the formulas for the coordinates of P:

$$\begin{split} &In[59]: \mbox{Clear[x1, y1, x2, y2, d];} \\ &pt = \mbox{Point2D[Point2D[{x1, y1}], Point2D[{x2, y2}], d]} \\ &Out[59] \mbox{Point2D}\Big[ \Big\{ x1 + \frac{d \ (-x1 + x2)}{\sqrt{(-x1 + x2)^2 + (-y1 + y2)^2}}, \ y1 + \frac{d \ (-y1 + y2)}{\sqrt{(-x1 + x2)^2 + (-y1 + y2)^2}} \Big\} \Big] \end{split}$$

Let t = d/D where  $D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$  is the distance from  $P_0$  to  $P_1$ . Solving for d = tD and substituting d into the Mathematica output above yields the parametric equations

$$x = x_1 + t(x_2 - x_1) y = y_1 + t(y_2 - y_1).$$

**Example.** Find the coordinates of the points at parameter values 0, 1/2 and 1 on the line segment whose start and end points are (-2, 1) and (1, 0), respectively. To what point does the parameter value t = -1 correspond? Plot the objects.

**Solution.** The function Segment2D[ $\{x_0, y_0\}$ ,  $\{x_1, y_1\}$ ][t] returns the coordinates of the point on a line segment at parameter value t.

```
In[60]: 11 = Segment2D[{-2, 1}, {1, 0}];
coords = Map[11[#]&, {-1, 0, 1/2, 1}]
Out[60] {{-5, 2}, {-2, 1}, {-<sup>1</sup>/<sub>2</sub>, <sup>1</sup>/<sub>2</sub>}, {1, 0}}
In[61]: Sketch2D[{11, Map[Point2D, coords]}];
```



The point at parameter value t = -1 is on the line connecting points  $P_0$  and  $P_1$ , at the same distance from  $P_0$  as  $P_1$ , but in the opposite direction.

# 5.16 Explorations

DISTANCE BETWEEN PARALLEL LINES.....lnsdst.nb

Demonstrate that the distance, d, between two parallel lines

$$Ax + By + C_1 = 0$$
 and  $Ax + By + C_2 = 0$ 

is given by

$$d = \sqrt{\frac{(C_2 - C_1)^2}{A^2 + B^2}}$$

INTERSECTION OF LINES IN INTERCEPT FORM......intrsct.nb

Show that the point of intersection of the lines

$$\frac{x}{a} + \frac{y}{b} = 1$$
 and  $\frac{x}{b} + \frac{y}{a} = 1$ 

is

$$\left(\frac{ab}{(a+b)},\frac{ab}{(a+b)}\right).$$

EQUATIONS OF PERPENDICULAR LINES.....lnsperp.nb

Show that the pair of lines ax + by + c = 0 and bx - ay + c' = 0 are perpendicular. Show that the pair

$$ax + by + c = 0$$
 and  $\frac{x}{a} - \frac{y}{b} + c' = 0$ 

is also perpendicular.

VERTICAL/HORIZONTAL DISTANCE TO A LINE.....lndist.nb

Show that the *vertical* distance,  $d_v$ , from a point  $(x_1, y_1)$  to a line whose equation is Ax + By + C = 0 is given by

$$d_v = \left| \frac{(Ax_1 + By_1 + C)}{B} \right|$$

and the *horizontal* distance,  $d_h$ , is given by

$$d_h = \left| \frac{(Ax_1 + By_1 + C)}{A} \right|$$

LINE GENERAL EQUATION DETERMINANT.....lndet.nb Show that the general equation of a line Ax + By + C = 0 is coincident with the line

$$\begin{vmatrix} x & y & 1 \\ -AC & -BC & A^2 + B^2 \\ B & -A & 0 \end{vmatrix} = 0.$$

given in determinant form.

LINE SEGMENT CUT BY TWO LINES.....lnlndist.nb

Let  $L_1$  and  $L_2$  be two *intersecting* lines and  $P_0$  a point. Describe a procedure for finding the lines through  $P_0$  such that  $L_1$  and  $L_2$  cut off a line segment of length S > 0. Implement the solution as a numerical *Mathematica* function.

INTERSECTION POINT OF TWO LINE SEGMENTS.....lnsegpt.nb

Show that the intersection point of the lines underlying two line segments  $P_1P_2$  and  $P_3P_4$ in terms of the coordinates of the four points is given by

$$x = \frac{(x_2 - x_1)(x_3y_4 - x_4y_3) - (x_4 - x_3)(x_1y_2 - x_2y_1)}{(x_4 - x_3)(y_1 - y_2) - (x_2 - x_1)(y_3 - y_4)}$$
  

$$y = \frac{(y_3 - y_4)(x_1y_2 - x_2y_1) - (y_1 - y_2)(x_3y_4 - x_4y_3)}{(x_4 - x_3)(y_1 - y_2) - (x_2 - x_1)(y_3 - y_4)}.$$

INTERSECTION PARAMETERS OF TWO LINE SEGMENTS.....lnsegint.nb

Show that the parameter values,  $t_1$  and  $t_2$ , of the intersection point of two line segments in terms of the end point coordinates is given by

$$t_1 = \frac{x_1(y_3 - y_4) - x_3(y_1 - y_4) + x_4(y_1 - y_2)}{D}$$
  
$$t_2 = \frac{-x_1(y_2 - y_3) + x_2(y_1 - y_3) - x_3(y_1 - y_2)}{D}$$

where

$$D = (x_1 - x_2)(y_3 - y_4) - (x_3 - x_4)(y_1 - y_2)$$

What is the significance of the values of  $t_1$  and  $t_2$  with respect to the standard parameter range for a line segment?

# Chapter 6

# Circles

The circle is the first curve we will study whose equation is of the second degree. Circles have been studied since antiquity and there exists an enormous number of interesting properties, theorems and relationships involving circles. This chapter provides the underlying analytic geometry of a circle and provides a glimpse at some of the catalog of knowledge about circles.

# 6.1 Definitions and Standard Equation

A *circle* is the locus of all points P(x, y) of the plane that have a constant distance r from a fixed point C(h, k); C is called the *center* and r the *radius* of the circle. Using the formula for the distance between two points, we find the equation of a circle in *standard form* to be

$$(x-h)^2 + (y-k)^2 = r^2.$$

Two particular cases of this equation occur frequently and deserve special mention. If the center is at the origin the equation reduces to

$$x^2 + y^2 = r^2.$$

If the x-axis contains a diameter of the circle, and the y-axis touches the circle at its extremity, then the equation becomes

$$x^2 + y^2 = 2rx.$$

**Example.** Write the equation of a circle with center at (-1, 1) and radius 2. Plot the circle.

**Solution.** The equation of the circle is  $(x + 1)^2 + (y - 1)^2 = 4$ . The Descarta2D representation of a circle is Circle2D[{h, k}, r], where {h, k} represents the co-ordinates of the center point, and r is the radius of the circle.

F

-



Figure 6.1: Circle with center at (h, k) and radius r.



**Example.** Determine which of the following points are on the circle centered at (-2, 1) with radius 3: (a) (3, 4), (b) (1, 1), (c) (-2, 4).

Solution. Points whose coordinates satisfy the equation

$$(x+2)^2 + (x-1)^2 = 9$$

are on the circle. The *Descarta2D* function IsOn2D[*point*, *circle*] will return True if the point is on the circle; otherwise, it returns False.

F

F

```
In[2]: c1 = Circle2D[{-2, 1}, 3];
p1 = Point2D[{3, 4}];
p2 = Point2D[{1, 1}];
p3 = Point2D[{-2, 4}];
{IsOn2D[p1, c1], IsOn2D[p2, c1], IsOn2D[p3, c1]}
```

Out[2] {False, True, True}

Therefore, points (b) and (c) are on the circle, and point (a) is not.

```
In[3]: Sketch2D[{c1, p1, p2, p3}];
```



E

F

Two circles are said to be *concentric* if their center points are coincident. Two circles are *coincident* if their center points are coincident and their radii are equal.

**Example.** Show that the two circles whose equations are  $(x-1)^2 + (y-2)^2 = 4$ and  $(x-1)^2 + (y-2)^2 = 9$  are concentric, but not coincident.

**Solution.** The result is obvious by inspection of the equations. The *Descarta2D* function IsConcentric2D[*circle*, *circle*] returns True if the two circles are concentric; otherwise, it returns False. IsCoincident2D[*circle*, *circle*] returns True if the two circles are coincident; otherwise, it returns False.

\_

#### 6.2 General Equation of a Circle

By expanding the standard equation of a circle with center point C(h,k) and radius r the equation may be written as

$$x^{2} + y^{2} - 2hx - 2ky + (h^{2} + k^{2} - r^{2}) = 0$$

which is a special case of the general second-degree equation

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

where the coefficients of  $x^2$  and  $y^2$  are equal and there is no xy term. Therefore, a necessary and sufficient condition that  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$  represent a circle is that A = C and B = 0. It is not necessary that A = C = 1 since the coefficients of  $x^2$  and  $y^2$ being equal can be divided out, reducing them to one. The equation

$$x^2 + y^2 + ax + by + c = 0$$

is the general equation of a circle. It can be reduced to standard form by completing the squares on the  $x^2$ - and x-terms, then on the  $y^2$ - and y-terms yielding

$$x^{2} + ax + \frac{a^{2}}{4} + y^{2} + by + \frac{b^{2}}{4} + c - \frac{a^{2}}{4} - \frac{b^{2}}{4} = 0$$

or

$$\left(x + \frac{a}{2}\right)^2 + \left(y + \frac{b}{2}\right)^2 = \frac{a^2 + b^2 - 4c}{4}.$$

This is the equation of a circle whose center is at (-a/2, -b/2) and whose radius is given by  $r = \frac{1}{2}\sqrt{a^2 + b^2 - 4c}$ . The equation will be a real circle only if  $a^2 + b^2 - 4c > 0$ ; if  $a^2 + b^2 - 4c = 0$  the equation represents a single point (a circle of zero radius); and if  $a^2 + b^2 - 4c < 0$  there are no real points in the locus.

**Example.** Find the center and radius of the circle  $2x^2 + 2y^2 - 5x + 4y - 7 = 0$ .

Solution. Descarta2D represents the quadratic equation

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

as

Quadratic2D[
$$A$$
,  $B$ ,  $C$ ,  $D$ ,  $E$ ,  $F$ ].

The function Loci2D[quad] will convert a quadratic (second-degree) equation to a list of objects represented by the equation. The function Circle2D[quad] will return a circle directly if the quadratic is indeed a circle.

```
In[5]: \{cl = Loci2D[q] = Quadratic2D[2, 0, 2, -5, 4, -7]], Circle2D[q]\}Out[5] \{\{Circle2D[\{\frac{5}{4}, -1\}, \frac{\sqrt{97}}{4}]\}, Circle2D[\{\frac{5}{4}, -1\}, \frac{\sqrt{97}}{4}]\}
```

The center of the circle is (5/4, -1) and the radius is  $\sqrt{97}/4$ . The Descarta2D function Quadratic2D[circle] converts a circle to a quadratic equation.

```
In[6]: Quadratic2D[c1[[1]]] // Simplify
Out[6] Quadratic2D[1, 0, 1, -\frac{5}{2}, 2, -\frac{7}{2}]
```



**Descarta2D Hint.** The Descarta2D function Simplify[quad] simplifies the coefficients of a quadratic by multiplying to remove denominators and factoring to remove common factors. The form quad //Simplify is an equivalent form of the function.

Mathematica Hint. In Mathematica the elements of a list are indicated by double square brackets surrounding the index of the element in the list. In the previous example, c1[[1]] indicates the first element in the list of objects c1.

#### 6.3 Circle from Diameter

Consider two points  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$  defining the end points of a diameter  $P_1P_2$  of a circle, C. Clearly the center of the circle, (h, k), must be the midpoint of  $P_1P_2$  and is given by

$$h = \frac{x_1 + x_2}{2}$$
 and  $k = \frac{y_1 + y_2}{2}$ 

and the radius of the circle, r, must be one-half the distance between  $P_1$  and  $P_2$ :

$$r = \frac{1}{2}\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

One equation of the circle C has a particularly simple form given by

$$(x - x_1)(x - x_2) + (y - y_1)(y - y_2) = 0$$

as can be verified by simplifying the equation of C in standard form.

## 6.4 Circle Through Three Points

Since the equation of a circle has three effective parameters (h, k, r or a, b, c), in general three conditions can be imposed upon the parameters to determine one (or more) circles. In this section we look at the case of a circle passing through three points. In a later chapter we will explore a large number of conditions for constructing circles satisfying three conditions.

We can find the equation of a circle passing through three points  $P_1(x_1, y_1)$ ,  $P_2(x_2, y_2)$ and  $P_3(x_3, y_3)$ , by substituting the coordinates of the points into the standard equation for a circle yielding the three equations

$$(x_1 - h)^2 + (y_1 - k)^2 = r^2$$
  

$$(x_2 - h)^2 + (y_2 - k)^2 = r^2$$
  

$$(x_3 - h)^2 + (y_3 - k)^2 = r^2.$$

This system of equations reduces to three linear equations in three unknowns, h, k and r. Simultaneous solution of the three linear equations gives

$$h = -\frac{H}{2D}, \ k = \frac{K}{2D}, \ \text{and} \ r = \frac{d_{12}d_{13}d_{23}}{2|D|},$$

where

$$H = \begin{vmatrix} 1 & 1 & 1 \\ y_1 & y_2 & y_3 \\ s_1 & s_2 & s_3 \end{vmatrix}, K = \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ s_1 & s_2 & s_3 \end{vmatrix}, D = \begin{vmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix},$$

1

and

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$
 and  $s_i = x_i^2 + y_i^2$ 

If D = 0 the points are collinear and no circle passes through the three points.

ī.

1

**Example.** Find the circle passing through the three points (1, 2), (-3, 1) and (0, -2).

**Solution.** The Descarta<sub>2D</sub> function Circle<sub>2D</sub> [*point*, *point*] returns a circle passing through the three points.

```
In[7]: cl = Circle2D[pl = Point2D[{1, 2}],
p2 = Point2D[{-3, 1}],
p3 = Point2D[{0, -2}]]
Out[7] Circle2D[{-<math>\frac{7}{10}, \frac{3}{10}}, \frac{17}{5\sqrt{2}}]
In[8]: Sketch2D[{pl, p2, p3, cl}];
```
F



The quadratic equation of a circle passing through three points  $P_1(x_1, y_1)$ ,  $P_2(x_2, y_2)$  and  $P_3(x_3, y_3)$  is given by the determinant equation

 $\begin{vmatrix} x^2 + y^2 & x & y & 1 \\ x_1^2 + y_1^2 & x_1 & y_1 & 1 \\ x_2^2 + y_2^2 & x_2 & y_2 & 1 \\ x_3^2 + y_3^2 & x_3 & y_3 & 1 \end{vmatrix} = 0.$ 

**Example.** Find the quadratic equation of the circle passing through the three points (1, 2), (-3, 1) and (0, -2) given in the previous example.

**Solution.** The Descarta2D function Quadratic2D[point, point, point] returns a quadratic representing the circle passing through the three points.

```
In[9]: Quadratic2D[p1, p2, p3]
Out[9] Quadratic2D[15, 0, 15, 21, -9, -78]
```

# 6.5 Intersection of a Line and a Circle

Consider the line Ax + By + C = 0 and the circle  $(x - h)^2 + (y - k)^2 = r^2$ . The points of intersection of the line and circle can be determined by solving the system of these two equations in two unknowns. The coordinates of the points of intersection,  $P_1$  and  $P_2$ , are given by

$$P_{1,2}\left(h - ad \pm b\sqrt{r^2 - d^2}, k - bd \mp a\sqrt{r^2 - d^2}\right)$$

where

$$a = \frac{A}{\sqrt{A^2 + B^2}}, \ b = \frac{B}{\sqrt{A^2 + B^2}} \ \text{and} \ d = \frac{Ah + Bk + C}{\sqrt{A^2 + B^2}}.$$

If  $r^2 - d^2 > 0$  (the radius is greater than the distance from the center point to the line), then there are two distinct intersection points; if  $r^2 - d^2 = 0$ , then the two intersection points are coincident (the line is tangent to the circle); and if  $r^2 - d^2 < 0$ , then the line and the circle do not intersect.

**Example.** Find the two points of intersection between the line and the circle whose equations are 2x - y + 3 = 0 and  $(x - 1)^2 + (y - 2)^2 = 9$ .

**Solution.** The Descarta2D function Points2D[*line*, *circle*] returns a list of the intersection points of the line and the circle.

. 1

### 6.6 Intersection of Two Circles

Consider two circles  $(x - h_1)^2 + (y - k_1)^2 = r_1^2$  and  $(x - h_2)^2 + (y - k_2)^2 = r_2^2$ . The coordinates of the two intersection points,  $P_1$  and  $P_2$ , of these circles can be determined by solving two equations in two unknowns. Alternately, the following geometric approach can be applied. Place the center of a circle with radius  $r_1$  at the origin and place the center of a second circle of radius  $r_2$  at (D,0) as shown in Figure 6.2. The equations of the two circles in standard form are clearly given by  $x^2 + y^2 = r_1^2$  and  $(x - D)^2 + y^2 = r_2^2$ , respectively. Solving the first equation for  $y^2$  yields  $y^2 = r_1^2 - x^2$ . Substituting this value of  $y^2$  into the second equation and solving for x yields

2-1012345

$$x = \frac{D^2 + r_1^2 - r_2^2}{2D}.$$

F



Figure 6.2: Two circles in a special position.

Substituting this value for x back into the first equation and solving for  $y^2$  yields

$$y^{2} = \frac{4D^{2}r_{1}^{2} - (D^{2} + r_{1}^{2} - r_{2}^{2})}{4D^{2}}.$$

Let  $R = D^2 + r_1^2 - r_2^2$  and let  $(x_0, y_0)$  designate the coordinates of the intersection points in this special position. Then

$$x_0 = \frac{R}{2D}$$
 and  $y_0 = \pm \frac{\sqrt{4D^2 r_1^2 - R^2}}{2D}$ .

If the expression under the radical in the expression for  $y_0$  is positive, then there are two distinct intersection points; if it is zero, the two intersection points are coincident (the circles are tangent at this point); and if it is negative, the two circles do not intersect. It is easy to show algebraically that

$$4D^2r_1^2 - R^2 = (D^2 - (r_1 + r_2)^2)((r_1 - r_2)^2 - D^2)$$

which confirms the intuitive insight that the circles do not intersect if either the sum of the radii is greater than the distance between the centers, or the difference of the radii is less than the distance between the centers.

Now consider two circles in arbitrary positions with centers  $C_1(h_1, k_1)$  and  $C_2(h_2, k_2)$  as shown in Figure 6.3. The x- and y-coordinates of the intersection points can be written in terms of the distances  $x_0$  and  $y_0$  determined from the special position shown in Figure 6.2 and are given by

$$x = h_1 + x_1 \pm x_2$$
 and  $y = k_1 + y_1 \mp y_2$ 

where

$$x_1 = x_0 \cos \theta, \ x_2 = y_0 \sin \theta, \ y_1 = x_0 \sin \theta, \ y_2 = y_0 \cos \theta$$

\_



Figure 6.3: Two circles in arbitrary positions.

and

F

F

$$\cos \theta = \frac{h_2 - h_1}{D}$$
 and  $\sin \theta = \frac{k_2 - k_1}{D}$ .

Therefore, the coordinates (x, y) of the intersection points of two circles without reference to trigonometric functions are

$$x = h_1 + x_0 \frac{(h_2 - h_1)}{D} \pm y_0 \frac{(k_2 - k_1)}{D}$$
$$y = k_1 + x_0 \frac{(k_2 - k_1)}{D} \mp y_0 \frac{(h_2 - h_1)}{D}$$

**Example.** Find the points of intersection between the two circles

$$(x-2)^{2} + (y-1)^{2} = 9$$
 and  $(x+2)^{2} + (y+3)^{2} = 16$ 

evaluated numerically.

**Solution.** The Descarta2D function Points2D[circle, circle] returns a list of the intersection points of the two circles.

In[13]: Sketch2D[{c1, c2, pts}];

F

F



# 6.7 Distance from a Point to a Circle

The distance, D, from a point  $P(x_0, y_0)$  to the circle  $(x - h)^2 + (y - k)^2 = r^2$  is given by

$$D = \sqrt{(r - \sqrt{(x_0 - h)^2 + (y_0 - k)^2})^2}.$$

The inner radical represents the distance from point P the center of the circle. The validity of the formula is easily verified by considering separately whether the point is inside, outside or on the circle.

**Example.** Find the distance from (2,3) to the circle  $(x+2)^2 + (y+1)^2 = 1$ .

**Solution.** The function Distance2D[*point*, *circle*] computes the distance between a point and a circle.

```
In[14]: Distance2D[Point2D[{2, 3}],
Circle2D[{-2, -1}, 1]]
Out[14] -1 + 4 √2
```

∃

-

-

### 6.8 Coaxial Circles

Let  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$  be the two points of intersection of the two circles

$$C_1 \equiv (x - h_1)^2 + (y - k_1)^2 = r_1^2$$
 and  
 $C_2 \equiv (x - h_2)^2 + (y - k_2)^2 = r_2^2.$ 

Consider the equation  $C \equiv (1 - \kappa)C_1 + \kappa C_2 = 0$ . This equation represents a circle since it is of the second degree, the coefficients of  $x^2$  and  $y^2$  are the same, and there is no xyterm. Moreover, points  $P_1$  and  $P_2$  are on the circle since both points satisfy the equation C. Therefore, C represents a *family* (or *pencil*) of circles through the points of intersection of the two given circles. A particular member of this family may be determined by specifying that it satisfy one other condition. Inspection of the equation reveals that C has a center (H, K)and radius R, where

$$H = (1 - \kappa)h_1 + kh_2$$

$$K = (1 - \kappa)k_1 + kk_2$$

$$R_1 = h_1^2 + k_1^2 - r_1^2$$

$$R_2 = h_2^2 + k_2^2 - r_2^2$$

$$R = \sqrt{H^2 + K^2 - ((1 - \kappa)R_1 + \kappa R_2)}$$

**Example.** The two circles

$$C_1 \equiv (x-2)^2 + (y-1)^2 = 9$$
 and  
 $C_2 \equiv (x+2)^2 + (y+3)^2 = 16$ 

determine a family of circles  $(1 - \kappa)C_1 + \kappa C_2 = 0$  passing through the points of intersection of  $C_1$  and  $C_2$ . Plot members of the family of circles for values of  $\kappa = \{0, \pm 1, \pm 2, \pm 3 \pm 4, \pm 5\}.$ 

**Solution.** The function Circle2D[circle, circle, k, Pencil2D] returns a circle parameterized by the variable k representing the pencil of circles passing through the intersection points of two circles.

```
In[15]: Clear[k];
c1 = Circle2D[{2, 1}, 3];
c2 = Circle2D[{-2, -3}, 4];
c12 = Circle2D[c1, c2, k, Pencil2D] // Simplify
Out[15] Circle2D[{2 - 4 k, 1 - 4 k}, \sqrt{9 - 25 k + 32 k<sup>2</sup>}]
```

```
In[16]: family = Map[(cl2 /. k -> #)&, Range[-5, 5]]
Out[16] {Circle2D[{22, 21}, \sqrt{934}], Circle2D[{18, 17}, 3\sqrt{69}],
```

```
 \begin{array}{l} \text{Circle2D} \left[ \left\{ 14, 13 \right\}, 2 \sqrt{93} \right], \text{Circle2D} \left[ \left\{ 10, 9 \right\}, \sqrt{187} \right], \text{Circle2D} \left[ \left\{ 6, 5 \right\}, \sqrt{66} \right], \\ \text{Circle2D} \left[ \left\{ 2, 1 \right\}, 3 \right], \text{Circle2D} \left[ \left\{ -2, -3 \right\}, 4 \right], \text{Circle2D} \left[ \left\{ -6, -7 \right\}, \sqrt{87} \right], \\ \text{Circle2D} \left[ \left\{ -10, -11 \right\}, \sqrt{222} \right], \text{Circle2D} \left[ \left\{ -14, -15 \right\}, \sqrt{421} \right], \\ \text{Circle2D} \left[ \left\{ -18, -19 \right\}, 6 \sqrt{19} \right] \right\} \end{array}
```

In[17]: Sketch2D[{c1, c2, family}];



#### 6.9 Radical Axis

Let  $C_1$  and  $C_2$  be the equations of two distinct circles as presented in the previous section. Consider the equation  $L \equiv C_1 - C_2$ . Upon simplification this equation reduces to the linear equation

$$L \equiv 2(h_2 - h_1)x + 2(k_2 - k_1)y + (h_1^2 + k_1^2 - r_1^2) - (h_2^2 + k_2^2 - r_2^2) = 0.$$

This line is called the *radical axis* of the circles  $C_1$  and  $C_2$ . The radical axis possesses the following properties which we state without proof.

- It is the line of the common chord if the two circles intersect in distinct real points.
- It is the common tangent line if the circles intersect in coincident points (are tangent internally or externally).
- It is a real straight line even if the circles do not intersect in real points.
- It is the locus of points from which tangents of equal length can be drawn to the two circles.
- It is perpendicular to the line of centers of the two circles.

- It does not exist (tends to infinity) as the defining circles tend to concentricity.
- The radical axes of three circles, taken in pairs, intersect in a point called the *radical center*.

**Example.** Find the radical axis of the circles  $(x - 4)^2 + (y - 1)^2 = 16$  and  $(x - h)^2 + (y - 1)^2 = 4$  for values of  $h = \{5, 6, 10, 11\}$ .

**Solution.** The Descarta2D function Line2D[circle, circle] returns the radical axis of the two circles.



Mathematica Hint. The Mathematica function Range[1,4] returns the list {1, 2, 3, 4}.

E

**\_\_\_\_\_** 

F

# 6.10 Parametric Equations

A circle may be parameterized in terms of the angle,  $\theta$ , that a ray from the center to the point at the parameter value makes with the +x-axis. The resulting equations are

 $x = h + r \cos \theta$  and  $y = k + r \sin \theta$ 

where (h, k) is the center of the circle and r is the radius of the circle. Values in the range  $0 \le \theta < 2\pi$  generate a complete locus of points on the circle.

**Example.** Generate 12 equally spaced points on the circle  $x^2 + y^2 = 4$  using the parametric equations.

**Solution.** The Descarta2D function Circle2D[ $\{h, k\}, r$ ][t] returns the coordinates of a point at parameter t on the circle.

```
In[22]: cl = Circle2D[{0, 0}, 2];
    pts = Map[Point2D[cl[#]]&, 2 * Pi * Range[0, 12] / 12];
    Sketch2D[{cl, pts}];
```



Alternately, consider the triangle T shown in Figure 6.4. Triangle T is obviously a right triangle since

$$(1-t^2)^2 + (2t)^2 = (1+t^2)^2$$

Therefore, the rational forms of the trigonometric functions for angle  $\theta$  are

$$\sin \theta = \frac{1 - t^2}{1 + t^2} \text{ and}$$
$$\cos \theta = \frac{2t}{1 + t^2}.$$

Substituting these expressions into the parameterization of a circle previously given yields

$$x = h + r \frac{1 - t^2}{1 + t^2}$$
 and  $y = k + r \frac{2t}{1 + t^2}$ .



Figure 6.4: Rational  $\sin \theta$  and  $\cos \theta$ .

These equations are called the *rational parameterization* of the circle and have the advantage that they can be evaluated without using trigonometric functions. Parameter values in the range  $0 \le t \le 1$  produce coordinates of points on the circle in the first quadrant,  $1 \le t < \infty$  the second quadrant,  $-\infty < t \le -1$  the third quadrant, and  $-1 \le t \le 0$  in the fourth quadrant. The point at  $\theta = \pi$  radians cannot be generated using these equations, so they are generally applied only to coordinates in the first quadrant. Also, notice that the points generated by these parametric equations do not produce equally spaced points measured by distance along the circle for equally spaced parameter values.

**Example.** Plot nine points at equal parameter values on the circle  $x^2 + y^2 = 25$  in the first quadrant using the rational parametric equations of the circle.

**Solution.** The points can be generated directly from the equations using parameter values  $0, \frac{1}{8}, \frac{1}{4}, \frac{3}{8}, \frac{1}{2}, \frac{5}{8}, \frac{3}{4}, \frac{7}{8}$  and 1.

```
In[23]: cl = Circle2D[{0, 0}, 5];
    pts = Map[Point2D[5 {1 - #^2, 2 #} / (#^2 + 1)]&, Range[0, 8] / 8];
    Sketch2D[{cl, pts}, PlotRange -> {{-1, 6}}];
```



E

F

# 6.11 Explorations

$$r^{2} + r_{1}^{2} - 2rr_{1}\cos(\theta - \theta_{1}) = R^{2}.$$

CHORD LENGTH OF INTERSECTING CIRCLES.....chdlen.nb



Show that the distance, d, between the intersection points of two circles is given by

$$d = \frac{\sqrt{-(D - r_1 - r_2)(D + r_1 - r_2)(D - r_1 + r_2)(D + r_1 + r_2)}}{D}$$

where D is the distance between the centers of the circles, and  $r_1$  and  $r_2$  are the radii of the two circles.

JOHNSON'S CONGRUENT CIRCLE THEOREM......johnson.nb

Take any three circles  $C_1$ ,  $C_2$  and  $C_3$  which pass through the origin, have equal radii, r, and intersect in pairs in two distinct points (one of the points is, by construction, the origin). Prove that the circle passing through the other three points of intersection between the circles taken in pairs is congruent to the original three circles (that is, this circle has a radius of r).

RADICAL CENTER.....radcntr.nb

Prove that the radical axes of three circles taken in pairs intersect in a common point. This point is called the *radical center* of the three circles.

CIRCLE-POINT MIDPOINT THEOREM.....cirptmid.nb



Show that the locus of midpoints from a fixed point  $P_0$  to a circle  $C_1$  of radius  $r_1$ , is a circle of radius  $\frac{1}{2}r_1$ . Furthermore, show that the center point of the locus is the midpoint of the segment between  $P_0$  and the center of  $C_1$ .

CIRCLE THROUGH THREE POINTS. ..... cir3pts.nb Show that the equation of the circle through the three points (0,0), (a,0) and (0,b) is  $x^2 + y^2 - ax - by = 0$ .

CONSTRUCTION OF TWO RELATED CIRCLES.....tnlncir.nb Prove that if OP and OQ are the tangent lines from (0,0) to the circle

$$x^2 + y^2 + 2gx + 2fy + c = 0,$$

then the equation of circle OPQ is

$$x^2 + y^2 + gx + fy = 0.$$

CIRCLE OF APOLLONIUS......apollon.nb

$$r = \frac{dk}{\sqrt{(k^2 - 1)^2}}$$

Show that the locus of a point P(x, y) that moves so that the ratio of its distance from two fixed points  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$  is a circle with radius

and center

$$\left(\frac{-x_1+k^2x_2}{k^2-1}, \frac{-y_1+k^2y_2}{k^2-1}\right)$$

where  $d = |P_1P_2|$ . The locus is called the Circle of Apollonius for the points  $P_1$  and  $P_2$  and the ratio k.

CASTILLON'S PROBLEM. ..... castill.nb

Let  $P_1$ ,  $P_2$  and  $P_3$  be three points inside the circle  $C_1 \equiv x^2 + y^2 = 1$ . Describe a method for inscribing a triangle inside  $C_1$  such that the sides of the triangle pass through the three given points.

RADICAL AXIS RATIO.....raratio.nb Show that the point of intersection of the radical axis and the line of centers of two circles of radii  $r_1$  and  $r_2$  divides the segment between the two centers into the ratio

$$\frac{d^2 + r_1^2 - r_2^2}{d^2 - r_1^2 + r_2^2},$$

where d is the distance between the centers.

# Chapter 7

# Arcs

We continue our study of circles by focusing on bounded portions of a circle's circumference commonly called *arcs*. Many of the interesting properties of arcs arise when considering how their end points and slopes meet with other curves. For example, many mechanical artifacts use arcs to construct transitions between the primary faces of the object giving a smoother and more durable design.

In addition to the topics presented in this chapter, a subsequent chapter will discuss another interesting use of arcs, the so-called *biarc* configuration of two arcs used to blend curves together smoothly.

### 7.1 Definitions

Consider the parametric equations of a circle

$$x = h + r \cos \theta$$
 and  $y = k + r \sin \theta$ 

where the point C(h,k) is the center of the circle and r is the radius of the circle. A circle is defined to be the set of points P(x, y) for all values of  $\theta$  such that  $0 \le \theta < 2\pi$  (radians). Using the same parametric equations, a *circular arc* may be defined to be the set of points P(x, y) for all values of  $\theta$  such that  $\theta_0 \le \theta \le \theta_1$ , where  $0 \le \theta_0 < 2\pi$  and  $\theta_0 < \theta_1 < (\theta_0 + 2\pi)$ . The point  $P_0(x_0, y_0)$  where  $\theta = \theta_0$  is called the *start point* of the arc, and the point  $P_1(x_1, y_1)$ where  $\theta = \theta_1$  is called the *end point* of the arc. The angle the directed line  $CP_0$  makes with the +x-axis is called the *end angle* of the arc. The center point C(h, k) of the circle is also the *center point* of the arc, and the radius, r, of the circle is the *radius* of the arc.

Let  $CP_0$  and  $CP_1$  be the lines determined by the center point C and the start and end point of the arc, respectively. The angle between lines  $CP_0$  and  $CP_1$  is called the *angular* span of the arc. An arc with an angular span of  $\pi$  radians (180°) is called a *semicircle*. The area bounded by line segments  $CP_0$  and  $CP_1$  and the arc itself is called a *sector*. The area

-



Figure 7.1: Definition of an arc.

bounded by the arc and  $P_0P_1$  is called a *segment* and the line segment  $P_0P_1$  itself is called the *chord* of the arc.

**Example.** Plot the arc centered at the point (-2, 1) with a radius 6 and start angle of  $\pi/6$  radians and end angle of  $\pi/2$  radians. Include the center point of the arc in the plot.

**Solution.** Circle2D[{h, k}, r][{ $\theta_1, \theta_2$ }] represents an arc of a circle between parameters  $\theta_1$  and  $\theta_2$  when plotting.

```
In[1]: p1 = Point2D[{-2, 1}];
a1 = Circle2D[{-2, 1}, 6][{Pi/6, Pi/2}];
Sketch2D[{p1, a1}];
```



-

ŀ



Figure 7.2: Bulge factor arc definition.

# 7.2 Bulge Factor Arc

We now consider an arc representation involving the arc's start and end points, the so-called *bulge factor arc* as illustrated in Figure 7.2. A bulge factor arc is specified by its start and end points plus an additional number specifying the "bulge" (or fatness) of the desired arc. More precisely, if  $P_0$  and  $P_1$  are the start and end points of the arc, and  $P_m$  is the midpoint of the arc, then the bulge factor, B, is defined to be the (non-zero, positive) ratio

$$B = \frac{2H}{D}$$

where D is the distance between  $P_0$  and  $P_1$  and H is the distance from  $P_m$  to the chordal line defined by  $P_0$  and  $P_1$ . Thus, an arc with B = 1 will be a semicircle. Closer examination of the definition of the bulge factor arc reveals that for a given value of B there are two arcs satisfying the definition. These arcs are mirror images of each other (the line passing through  $P_0$  and  $P_1$  is the reflection line). To distinguish between these two arcs we make the arbitrary definition that the arc will be traversed counter-clockwise from  $P_0$  to  $P_1$ . The mirrored arc is represented by interchanging the roles of  $P_0$  and  $P_1$ .

#### **Radius and Center**

In order to determine defining parameters of the circle underlying the bulge factor arc, we need to determine the radius, r, and the center point, C(h, k), in terms of the points,  $P_0(x_0, y_0)$ and  $P_1(x_1, y_1)$ , and the bulge factor, B. Consider the right triangle  $CP_0P_M$  where  $P_M$  is the midpoint of the chord  $P_0P_1$ . By the Pythagorean Theorem

$$|CP_M|^2 + |P_0P_M|^2 = |CP_0|^2$$

or

$$(r-H)^2 + \left(\frac{D}{2}\right)^2 = r^2.$$

Solving for r and substituting H = BD/2 yields

$$r = \frac{D}{4} \left( B + \frac{1}{B} \right)$$

where r > 0, since B > 0 and D > 0.

To find the coordinates C(h, k) of the center point of the arc we note that the center is offset from point  $P_M$  a distance (r - H). The direction of the offset is rotated -90 degrees from the vector  $P_0 - P_1$ . Therefore, the equations for C(h, k) are

$$h = \frac{x_0 + x_1}{2} + \kappa(y_0 - y_1)$$
 and  $k = \frac{y_0 + y_1}{2} - \kappa(x_0 - x_1)$ 

where

F

$$\kappa = \frac{1}{4} \left( \frac{1}{B} - B \right).$$

It is clear from the expressions for r and C that if we replace B with 1/B we get an arc with the same radius and center, whose locus is counter-clockwise from  $P_1$  to  $P_0$ . This arc is the *complement* of the original arc. The reflection of the original arc in the chord may be obtained by reversing the roles of  $P_0$  and  $P_1$  and using the same value, B, as the bulge factor.

**Example.** Plot the arc with end points (1,0) and (0,1) with a bulge factor of 1/2. Find the mirror image of the arc reflected in the chord.

Solution. The standard representation of an arc in Descarta2D is

$$Arc2D[coords_0, coords_1, B]$$

where the start and end point coordinates are given as the first two arguments, respectively, and the bulge factor is the third argument. The arc reflected in the chord is constructed by reversing the roles of the start and end points.

```
In[2]: a1 = Arc2D[{1, 0}, {0, 1}, 1/2];
a2 = Arc2D[{0, 1}, {1, 0}, 1/2];
In[3]: Sketch2D[{a1, a2, Point2D[{1, 0}], Point2D[{0, 1}]}];
```



**Descarta2D Hint.** The Descarta2D function Arc2D[arc, Complement2D] constructs the complement of an arc.

#### Angles

Let  $\theta$  be the angular span of a bulge factor arc defined by points  $P_0$  and  $P_1$  and bulge factor B. Once again examining the right triangle  $CP_0P_M$  reveals that

$$\tan\left(\frac{\theta}{2}\right) = \frac{d/2}{(r-H)}$$
$$= \frac{2B}{(1-B^2)}.$$
(7.1)

Using the trigonometric identity

$$\tan(2A) = \frac{2\tan A}{(1 - \tan^2 A)}$$

we find that

$$B = \tan \frac{\theta}{4}.$$

From this equation it is clear that if B < 1, the arc is a *minor* arc whose angular span is in the range  $0 < \theta < \pi$ ; if B > 1 the arc is a *major* arc with  $\pi < \theta < 2\pi$ .

Let  $\alpha$  denote the angle between the initial tangent vector,  $\mathbf{V}_0$ , and the chord vector,  $\mathbf{P}_1 - \mathbf{P}_0$ , considered positive when  $\mathbf{V}_0$  is clockwise from the chord vector, and negative when  $\mathbf{V}_0$  is counter-clockwise from the chord. Note that  $-\pi < \alpha < \pi$  and  $|\alpha| = \theta/2$ . Therefore,  $B = \tan(\alpha/2)$ .

From Equation (7.1) we can derive an expression for B in terms of  $\sin \alpha$  and  $\cos \alpha$  as follows

$$\tan \alpha = \frac{\sin \alpha}{\cos \alpha} = \frac{k \sin \alpha}{k \cos \alpha} = \frac{s}{c} = \frac{2B}{(1 - B^2)}$$

Solving this quadratic for B in terms of s and c yields

$$B = \frac{s}{c + \sqrt{s^2 + c^2}}.$$
 (7.2)

(The positive root of the quadratic is selected in order to insure that B has the same sign as s. If B turns out to be negative, then the arc's start and end points are interchanged and the absolute value of B is the positive bulge factor.) The constants s and c are some multiple of  $\sin \alpha$  and  $\cos \alpha$  and immediately provide several useful techniques for constructing arcs. These techniques are illustrated in the "Explorations" section at the end of this chapter.

### 7.3 Three–Point Arc

⊢

Let  $P_0$  and  $P_1$  be the start and end points of an arc and let point P be any other point on the arc. One method for constructing the arc through the three points is to first construct the underlying circle through the three points and then compute the limiting angles of the arc from the end points and the center. Alternately, the bulge factor arc form provides an appealing method for computing the arc. Note that  $P_0$  and  $P_1$  are the chord end points required in the bulge factor arc formulation and in order to fully define the arc, we need to determine its bulge factor, B. As the third point P traverses the arc the angle subtended at P by the chord  $P_0P_1$  remains constant at the value  $(\pi - \alpha)$ . Thus, using the simpler vector form,

$$s = |(\mathbf{P} - \mathbf{P}_0) \times (\mathbf{P}_1 - \mathbf{P})|$$
  

$$c = (\mathbf{P} - \mathbf{P}_0) \cdot (\mathbf{P}_1 - \mathbf{P}).$$

From s and c we compute B using Equation (7.2).

**Example.** Find and plot the arc passing through the points (4, 2), (-2, 4), and (0, -6).

**Solution.** The function Arc2D[*point*, *point*, *point*] returns an arc through three points. The first and third points are assumed to be the end points of the arc chord.

```
In[4]: p1 = Point2D[{4, 2}];
    p2 = Point2D[{-2, 4}];
    p3 = Point2D[{0, 6}];
    a1 = Arc2D[p1, p2, p3] // N
Out[4] Arc2D[{0, 6.}, {4., 2.}, 1.61803]
In[5]: Sketch2D[{p1, p2, p3, a1}];
```



# 7.4 Parametric Equations

One possible set of parametric equations for an arc is very similar to those of a circle since they both have the same underlying curve. The parameter, t, can be scaled in a different manner so that parameter value t = 0 produces the start point of the arc, and parameter value t = 1 produces the end point. The resulting parametric equations are

$$x = h + r\cos(\theta_0 + t(\theta_1 - \theta_0))$$
  

$$y = k + r\sin(\theta_0 + t(\theta_1 - \theta_0))$$

where (h, k) is the center of the arc, r is the radius, and  $\theta_0$  and  $\theta_1$  are the start and end angles, respectively, of the arc.

Alternatively, since the standard form of an arc used in Descarta2D is

$$Arc2D[\{x_0, y_0\}, \{x_1, y_1\}, B],$$

we seek parametric equations involving only the start and end point coordinates and the bulge factor. The equations are given by

$$x = h + (x_0 - h)\cos(\beta t) - (y_0 - k)\sin(\beta t)$$
  

$$y = k + (x_0 - h)\sin(\beta t) + (y_0 - k)\cos(\beta t)$$

where  $\beta = 4 \tan^{-1}(B)$  is the span of the arc and C(h, k) is the center point of the arc. Parameter values in the range  $0 \le t \le 1$  generate coordinates covering the span of the arc.

**Example.** Plot eight equally spaced points on the arc between the points (-3, 2) and (2, 1) with a bulge factor of 3/2

**Solution.** The Descarta2D function  $\operatorname{Arc2D}[\{x_0, y_0\}, \{x_0, y_0\}, B][t]$  returns the coordinates at parameter value t on the arc.



# 7.5 Points and Angles at Parameters

Using the parametric equations of an arc defined in the previous section we can find the coordinates of any point on the arc corresponding to an angle  $\theta$  in the range  $\theta_1 \leq \theta \leq \theta_2$ . The parametric equations for an arc as defined in *Descarta2D* are normalized so that the parameter value 0 generates the start point of the arc and the parameter value 1 generates the end point of the arc. Parametric values t, 0 < t < 1, will generate points on the arc between the start and end points.

**Example.** For the arc between the points  $P_0(-2,1)$  and  $P_1(2,2)$  with a bulge factor of 3/2, use the parametric definition of an arc to find and plot the start point, end point and midpoint of the arc.

**Solution.** The function  $\operatorname{Arc2D}[\{x_0, y_0\}, \{x_1, y_1\}, B][t]$  returns a list of coordinates representing the point on the arc at a given parameter value.

```
In[7]: al = Arc2D[{-2, 1}, {2, 2}, 3/2];
coords = {a1[0], a1[1/2], a1[1]} // N
Out[7] {{-2., 1.}, {0.75, -1.5}, {2., 2.}}
In[8]: Sketch2D[{al, Map[Point2D, coords]}];
```

E

⊢



Whereas the Descarta2D function arc[t] generates the coordinates of the point on the arc at parameter t, the function Angle2D[arc, t] returns the angle (in radians) on the arc at parameter t with respect to a horizontal line such as the x-axis.

In[9]: Angle2D[a1, 1/2] // N Out[9] -1.32582

# 7.6 Arcs from Ray Points

F

Sometimes it is more convenient to specify the start and end points of an arc, rather than the start and end angles. One obvious construction method is to specify the center and radius along with the start and end points. Let  $P_0(x_0, y_0)$  and  $P_1(x_1, y_1)$  be the desired start and end points of an arc centered at C(h, k) with radius r. Using simple trigonometry, the start and end angles of the arc are given by

$$\theta_0 = \tan^{-1}\left(\frac{x_0 - h}{y_0 - k}\right) \text{ and } \theta_1 = \tan^{-1}\left(\frac{x_1 - h}{y_1 - k}\right).$$

The arc tangent function used to implement these equations must be sophisticated enough to assign the proper angle based on which quadrant the points are located in. *Mathematica* provides such an arc tangent function that takes the numerator and denominator as separate arguments and computes the angle in the proper quadrant.

**Example.** Plot the arc centered at the point (2,1) with radius 1, and with start and end points of (7,1) and (2,6).

**Solution.** The function Arc2D[*point*, *r*, {*point*, *point*}] returns a bulge factor arc given the center point, radius, and start and end points.

**Descarta2D Hint.** The Arc2D function introduced in the previous example allows more flexible input of the arc start and end points than is obvious from the example. These points may be located at any position on the ray extending from the center point of the arc to the desired arc start and end points. The arc will be bounded by the points of intersection between the circle underlying the arc and the rays defined from the center point to the specified start and end points.

# 7.7 Explorations

ARC FROM BOUNDING POINTS AND ENTRY DIRECTION.....arcentry.nb

Let  $P_0$  and  $P_1$  be the start and end points, respectively, of an arc and P be a third point on the vector tangent to the arc at  $P_0$ . Show that

$$s = |(\mathbf{P} - \mathbf{P}_0) \times (\mathbf{P}_1 - \mathbf{P}_0)|$$
  

$$c = (\mathbf{P} - \mathbf{P}_0) \cdot (\mathbf{P}_1 - \mathbf{P}_0)$$

represent values of s and c useful for computing the bulge factor of the arc.

Arc from Bounding Points and Exit Direction.....arcexit.nb

Let  $P_0$  and  $P_1$  be the start and end points of an arc, respectively, and P be a third point on the vector tangent to the arc at  $P_1$ . Show that

$$s = |(\mathbf{P}_1 - \mathbf{P}_0) \times (\mathbf{P} - \mathbf{P}_1)|$$
  
$$c = (\mathbf{P}_1 - \mathbf{P}_0) \cdot (\mathbf{P} - \mathbf{P}_1)$$

represent values of s and c useful for computing the bulge factor of the arc.

MIDPOINT OF ARC.....arcmidpt.nb



Show that the midpoint, P of a bulge factor arc between points  $P_0$  and  $P_1$  whose bulge factor is B has coordinates

$$P\left(\frac{(x_0+x_1)-B(y_0-y_1)}{2},\frac{(y_0+y_1)+B(x_0-x_1)}{2}\right).$$

CENTROID OF SEMICIRCULAR ARC.....arccent.nb

Show that the centroid of the area bounded by a semicircular arc of radius r and its chord is on the axis of symmetry at a distance

$$H = \frac{4r}{3\pi}$$

from the chord of the arc.

# Chapter 8

# Triangles

Even though a triangle is not easily represented by a single, simple equation, there exist so many interesting properties of triangles that it is worth devoting a special chapter to them. Even today, new relationships involving triangles continue to be discovered. *Descarta2D* implements the triangle as a named object to enable easy study of the mathematical relationships arising from the geometry of a triangle.

# 8.1 Definitions

F

A *triangle* is a composite object consisting of the three line segments connecting three noncollinear points. The line segments are called the *sides* of the triangle, and the points are called the *vertices* of the triangle. The two line segments adjacent to each vertex form an angle inside the triangle called a *vertex angle*.

A triangle is *isosceles* if two sides are equal in length, and the third side is called the *base*. In an *equilateral* triangle all three sides are equal length. A triangle is called *acute* if all the interior angles measure less than 90°. A *right* triangle has one interior angle of 90°, and the side opposite the right angle is called the *hypotenuse*. A triangle with an interior angle greater than 90° is called an *obtuse* triangle. A triangle with three unequal sides is a *scalene* triangle. It is clear from Figure 8.1 that the sum of the interior angles of a triangle is 180 degrees ( $\pi$  radians).

**Example.** Plot the triangle connecting the points (-1, -1), (2, 0) and (-2, 1). Use *Descarta2D* functions to retrieve the vertex points and vertex angles of the triangle (in degrees).

7

Solution. In Descarta2D a triangle is represented as

Triangle2D[coords1, coords2, coords3]



Figure 8.1: Sum of triangle interior angles.

where  $coords_1$ ,  $coords_2$  and  $coords_3$  are the coordinates of the first, second and third vertex points, respectively. The Descarta2D function Point2D[triangle, n] returns a point located at vertex n of the triangle. Angle2D[triangle, n] returns the vertex angle at vertex n of the triangle.

```
In[1]: t1 = Triangle2D[{-1, -1}, {2, 0}, {-2, 1}];
    {Map[Point2D[t1, #]&, {1, 2, 3}],
    Map[Angle2D[t1, #]&, {1, 2, 3}] / Degree // N}
```

```
In[2]: Sketch2D[{t1}];
```

F



**Example.** Given the lines 2x - 3y + 4 = 0, -4x + 2y + 2 = 0 and 4x + 5y - 2 = 0 construct the triangle whose sides lie on the lines.

**Solution.** The Descarta2D function Triangle2D[*line*, *line*] returns a triangle defined by three lines.

```
In[3]: 11 = Line2D[2, -3, 4];
          12 = Line2D[-4, 2, 2];
          13 = Line2D[4, 5, -2];
          t1 = Triangle2D[11, 12, 13]
\textit{Out[3] Triangle2D} \left[ \left\{ \frac{7}{4} \,,\, \frac{5}{2} \right\},\, \left\{ -\frac{7}{11} \,,\, \frac{10}{11} \right\},\, \left\{ \frac{1}{2} \,,\, 0 \right\} \right]
In[4]: pr = PlotRange -> { { -2, 3 } , { -1, 3 } };
          Sketch2D[{11, 12, 13}, pr];
          Sketch2D[{t1}, pr];
                      3
2.5
                                                                      3
2.5
                      1.5
                                                                     2
1.5
                      10.5
                                                                     1
0.5
                           0
                                                                          C
                                                                    -0.5
                    -0.5
                           1
                                                                         - 1
                                 -1
                                        0
                                               1
                                                     2
                                                            3
                                                                                 -1
                                                                                       0
                                                                                              1
```

F

**Example.** Find the line segment and the line associated with the side connecting vertices 2 and 3 of Triangle2D[{2,3}, {-1,2}, {-3,2}].

2 3

**Solution.** The Descarta2D function Segment2D[triangle,  $n_1$ ,  $n_2$ ] returns the line segment connecting two vertices of a triangle. Line2D[triangle,  $n_1$ ,  $n_2$ ] returns the line containing the side of a triangle through two vertices.



Figure 8.2: Centroid of a triangle.

## 8.2 Centroid of a Triangle

The "balance point" of a planar area defined by bounding curves is called the *center of gravity* of the area. When the material covering the area has a *constant* density throughout, and the formulas for the center of gravity depend purely on the size and shape of the area, the center of gravity is called the *centroid* of the area. For symmetric geometric figures such as a square, circle or ellipse, the centroid is obviously the center point of the figure.

Referring to Figure 8.2 it is intuitively obvious that the triangle will "balance" on some horizontal line at ordinate  $\bar{y}$ . The position of this line can be determined by summing the moments,  $\Delta M = D(x_2 - x_1)\Delta y$ , of infinitesimal rectangles on either side of the line. The value of  $\bar{y}$  is the ordinate at which the sum of the moments is equal on both sides of the line. The usual method for determining  $\bar{y}$  is to use integral calculus and the actual derivation is included as the exploration tricent.nb. The derivation shows that the line is one-third of the distance from the base of the triangle to the apex, and the coordinates of the centroid, P, of the triangle are given by

$$P\left(\frac{x_1 + x_2 + x_3}{3}, \frac{y_1 + y_2 + y_3}{3}\right)$$

where  $(x_1, y_1)$ ,  $(x_2, y_2)$  and  $(x_3, y_3)$  are the coordinates of the triangle's vertex points. The centroid point coordinates can be determined by intersecting a pair of lines offset from the sides of the triangle one-third of the distance from the side towards the opposite vertex.

The *medians* of a triangle are the lines connecting the vertices to the midpoints of the opposite sides. The medians taken in pairs intersect in coincident points, and the point is the centroid of the triangle as is shown by the following *Descarta2D* commands:

E

```
In[6]: Clear[x1, y1, x2, y2, x3, y3];
p1 = Point2D[{x1, y1}];
p2 = Point2D[{x2, y2}];
p3 = Point2D[{x3, y3}];
pt1 = Point2D[Line2D[p1, Point2D[p2, p3]],
Line2D[p2, Point2D[p1, p3]]] // Simplify
Out[6] Point2D[{ 1/3 (x1 + x2 + x3), 1/3 (y1 + y2 + y3) }]
```

**Example.** Find the centroid of the triangle whose vertices are (-2, -1), (3, 1), and (0, 2). Show by plotting that the medians intersect at the centroid.

**Solution.** The Descarta2D function Point2D[*triangle*, Centroid2D] returns the centroid point of a triangle.

```
In[7]: t1 = Triangle2D[{-2, -1}, {3, 1}, {0, 2}];
    pt = Point2D[t1, Centroid2D]
Out[7] Point2D[{ 1/3, 2/3 }]
In[8]: {c1, c2, c3} = {{-2, -1}, {3, 1}, {0, 2}};
    s12 = Segment2D[c1, c2];
    s13 = Segment2D[c1, c3];
    s23 = Segment2D[c2, c3];
    m1 = Segment2D[Point2D[c1], Point2D[s23]];
    m2 = Segment2D[Point2D[c2], Point2D[s13]];
    m3 = Segment2D[Point2D[c3], Point2D[s12]];
```

```
In[9]: Sketch2D[{t1, pt, m1, m2, m3}];
```



# 8.3 Circumscribed Circle

A circle passing through the three vertex points of a triangle is said to be *circumscribed* about the triangle. We have already provided in a previous chapter the equation of a circle passing through three points. Since the sides of the triangle are chords of the circle, and the perpendicular bisectors of the chords of a circle intersect at the center point, the center of the circumscribed circle is the intersection point of the perpendicular bisectors of the triangle's sides taken in pairs.

**Example.** Find the circle circumscribing the triangle whose vertices are (1, 2), (-2, 4) and (-3, 1). Show by plotting that the perpendicular bisectors of the sides of the triangle intersect at the center point of the circumscribed circle.

**Solution.** The function Circle2D[*triangle*, Circumscribed2D] returns the circle that circumscribes the triangle.

```
In[10]: pl = Point2D[{1, 2}];
    p2 = Point2D[{-2, 4}];
    p3 = Point2D[{-3, 1}];
    t1 = Triangle2D[p1, p2, p3];
    c1 = Circle2D[t1, Circumscribed2D] // N
Out[10] Circle2D[{-1.13636, 2.04545}, 2.13685]
In[11]: Sketch2D[{p1, p2, p3, t1, c1, Point2D[c1],
    Line2D[p1, p3, Perpendicular2D],
    Line2D[p1, p3, Perpendicular2D],
    Line2D[p2, p3, Perpendicular2D]}];
```

0

-1

-3 -2 -1 0 1

2

F

**Descarta2D Hint.** Point2D[*triangle*, Circumscribed2D] directly returns the center point of the circumscribed circle of a triangle.

The radius, R, of the circle circumscribing a triangle whose sides are of length  $s_1$ ,  $s_2$  and  $s_3$  is given by

$$R = \frac{s_1 s_2 s_3}{\sqrt{PS}}$$

where  $S = s_1 + s_2 + s_3$  and  $P = (-s_1 + s_2 + s_3)(s_1 - s_2 + s_3)(s_1 + s_2 - s_3)$ . This formula is derived in the exploration trirad.nb.

# 8.4 Inscribed Circle

A circle inside a triangle that is tangent to all three of the triangle's sides is said to be *inscribed* in the triangle. The center of the inscribed circle must lie on the angle bisectors of the triangle's sides because the center must be equidistant from the sides. Therefore, the point of intersection of a pair of angle bisectors of a triangle is the center of the inscribed circle. The radius of the inscribed circle is the distance from the center point to any one of the triangle's sides. The center (h, k) and radius r of the inscribed circle derived from this construction yields

$$h = (s_1x_1 + s_2x_2 + s_3x_3)/2s$$
  

$$k = (s_1y_1 + s_2y_2 + s_3y_3)/2s$$
  

$$r = \sqrt{(s - s_1)(s - s_2)(s - s_3)/s}$$

where

$$s_1 = \sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2}$$
  

$$s_2 = \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2}$$
  

$$s_3 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$
  

$$s = (s_1 + s_2 + s_3)/2.$$

**Example.** Find the circle inscribed in the triangle whose vertices are (-3,3), (3,3) and (1,-3). Show by plotting that the angle bisectors of the sides of the triangle intersect at the center of the inscribed circle.

**Solution.** The function Circle2D[*triangle*, Inscribed2D] returns the circle inscribed in the triangle.

```
In[12]: p1 = Point2D[{-3, 3}];
       p2 = Point2D[{3, 3}];
       p3 = Point2D[{1, -3}];
       t1 = Triangle2D[p1, p2, p3];
       c1 = Circle2D[t1, Inscribed2D] // N
Out[12] Circle2D[{0.443274, 1.15722}, 1.84278]
In[13]: Sketch2D[{p1, p2, p3, t1, c1, Point2D[c1],
         MedialEquations2D[{Line2D[p1, p2], Line2D[p1, p3]}][[2]],
         MedialEquations2D[{Line2D[p2, p3], Line2D[p2, p1]}][[2]],
         \label{eq:medialEquations2D[{Line2D[p3, p1], Line2D[p3, p2]}][[2]]};
                             4
                             2
                            0
                           -2
                           -4
                                                   2
                                                         4
                               -4
                                      -2
                                             0
```

**Descarta2D Hint.** The function Point2D[*triangle*, Inscribed2D] directly returns the center point of the inscribed circle of a triangle.

The radius, r, of a circle inscribed in a triangle whose sides are of length  $s_1$ ,  $s_2$  and  $s_3$  is given by

$$r = \frac{1}{2}\sqrt{\frac{P}{S}}$$

where  $S = s_1 + s_2 + s_3$  and  $P = (-s_1 + s_2 + s_3)(s_1 - s_2 + s_3)(s_1 + s_2 - s_3)$ . This formula is derived in the exploration trirad.nb.

# 8.5 Solving Triangles

Clearly, the shape of a triangle, independent of its position and orientation, is determined by its side lengths and vertex angle magnitudes. Labeling the sides and angles as shown in Figure 8.3 relative to the vertices, we pose the problem of determining all of the configuration parameters  $(s_n \text{ and } a_n)$  given a subset of them. The configuration parameters are always assumed to be



Figure 8.3: Standard labeling of a triangle's sides and angles.

positive and the angles less than  $\pi$  radians. Generally, a unique triangle is determined by specifying three of the six configuration parameters, although in two cases (AAA and SSA), as outlined below, the configuration is ambiguous. The configurations requiring consideration are enumerated as

- **AAA** Angle-Angle-Angle: Specifying three angles of a triangle determines the shape of a family of similar triangles, but is ambiguous as to the lengths of the sides.
- **AAS** Angle-Angle-Side: The AAS configuration is specified by two angles and a side *not* between them. For example, the configuration parameters  $a_1$ ,  $a_2$  and  $s_1$  specify an AAS configuration. In a valid configuration the sum of the two given angles must be less than  $\pi$  radians, and such configurations admit a unique solution.
- **ASA** Angle-Side-Angle: The ASA configuration is specified by two angles and the side between the angles. The configuration parameters  $a_1$ ,  $s_3$  and  $a_2$ , for example, illustrate an ASA configuration. The ASA configuration allows a unique solution if the sum of the two angles is less than  $\pi$  radians.
- **SAS** Side-Angle-Side: The SAS configuration involves two sides and the angle between them. The configuration given by  $s_1$ ,  $a_3$  and  $s_2$  is an example of an SAS configuration. The SAS configuration specifies a unique solution for all values of the configuration parameters.
- **SSA** Side-Side-Angle: SSA configurations  $(s_1, s_2 \text{ and } a_1, \text{ for example})$  are referred to as the *ambiguous* case, because two valid solutions may exist. That is, two different sets of configuration parameters representing two different triangles may satisfy the configuration. In some cases (right triangles) only one solution may exist, and in other cases the configuration may be inconsistent allowing no solutions.
- **SSS** Side-Side-Side: Specifying three sides of a triangle determines a unique triangle, or may be inconsistent if the length of one side is greater than or equal to the sum of the lengths of the other two sides.

The process of determining the full set of configuration parameters given one of the cases above is called *solving* the triangle. Solving triangles involves the application of three fundamental principles. In terms of the configuration parameters these three principles lead to the following equations:

- Sum of the angles of a triangle is  $\pi$  radians:  $a_1 + a_2 + a_3 = \pi$ .
- The Law of Sines (three equations):

$$\frac{s_1}{\sin(a_1)} = \frac{s_2}{\sin(a_2)} = \frac{s_3}{\sin(a_3)}.$$

• The Law of Cosines (three equations):

$$\begin{aligned} s_1^2 &= s_2^2 + s_3^2 - 2s_2s_3\cos(a_1), \\ s_2^2 &= s_1^2 + s_3^2 - 2s_1s_3\cos(a_2), \\ s_3^2 &= s_1^2 + s_2^2 - 2s_1s_2\cos(a_3). \end{aligned}$$

When applying the Law of Sines or the Law of Cosines, care must be taken to properly handle the ambiguous cases noting that  $\sin(a) = \sin(\pi - a)$ , for example, when applying the Law of Sines. The *Descarta2D* package D2DTriangle2D provides details illustrating how to solve all the triangle configuration cases using these principles.

**Example.** Find all the configuration parameters for a triangle with  $s_1 = 1$ ,  $s_2 = 2$  and  $a_3 = \pi/6$  radians. Construct a triangle satisfying this SAS configuration.

**Solution.** SolveTriangle2D[{ $\{s_1, s_2, s_3\}, \{a_1, a_2, a_3\}$ }] returns a complete specification a triangle configuration, in the form { $\{s_1, s_2, s_3\}, \{a_1, a_2, a_3\}$ }, given three of the six configuration parameters. The three parameters to be found must be omitted (i.e. entered as Null in the configuration). The function Triangle2D[*config*] returns the triangle satisfying the configuration. The first vertex of the triangle will be the origin and the second vertex will be on the +*x*-axis.

$$Out [14] \left\{ \left\{ 1, 2, \sqrt{5 - 2\sqrt{3}} \right\}, \left\{ \arccos\left[\frac{4 - \sqrt{3}}{2\sqrt{5 - 2\sqrt{3}}} \right], \operatorname{ArcCos}\left[\frac{1 - \sqrt{3}}{\sqrt{5 - 2\sqrt{3}}} \right], \frac{\pi}{6} \right\} \right\}$$

In[15]: Triangle2D[sa] // Simplify

 $Out[15] \text{ Triangle2D}[\{0, 0\}, \{\sqrt{5-2\sqrt{3}}, 0\}, \{\frac{4-\sqrt{3}}{\sqrt{5-2\sqrt{3}}}, \frac{1}{\sqrt{5-2\sqrt{3}}}\}]$ 

E

F


F

**Descarta2D Hint.** Triangle2D[ $\{s_1, s_2, s_3\}$ ] constructs a triangle given the lengths of the sides only.

**Example.** Find all configuration parameters for a triangle with  $s_1 = 1$ ,  $s_2 = 1.5$  and  $a_1 = \pi/6$  radians (an SSA case).

**Solution.** The function SolveTriangle2D[config] returns a complete triangle configuration given a partial configuration. Since this is an SSA case, there is a possibility of two solutions. SolveTriangle2D[config, True] will return the alternate triangle configuration, if one exists.



\_

## 8.6 Cevian Lengths

A *cevian* of a triangle is a line segment connecting a vertex to a point on the line containing the side of the triangle opposite the vertex. Therefore a cevian may be inside the triangle (if the point is on the opposite side) or it may be outside the triangle (if the point is on the extension of the line which contains the opposite side). Common cevians include the *altitude* of a triangle which is the cevian perpendicular to the opposite side, the *median* which connects the vertex to the midpoint of the opposite side and the *angle bisector* which bisects the angle at the vertex.

If a triangle has sides whose lengths are  $s_1$ ,  $s_2$  and  $s_3$  opposite vertices  $V_1$ ,  $V_2$  and  $V_3$ , then the length of the altitude,  $h_1$ , from  $V_1$  is given by

$$h_1 = \frac{\sqrt{PS}}{2s_1}$$

where  $S = s_1 + s_2 + s_3$  and  $P = (-s_1 + s_2 + s_3)(s_1 - s_2 + s_3)(s_1 + s_2 - s_3)$ . The length of the median,  $m_1$ , from vertex  $V_1$  is given by

$$m_1 = \frac{1}{2}\sqrt{-s_1^2 + 2(s_2^2 + s_3^2)}.$$

The length of the angle bisector,  $b_1$ , from  $V_1$  is given by

$$b_1 = \frac{\sqrt{Ss_2s_3(-s_1+s_2+s_3)}}{s_2+s_3}.$$

The formulas for the lengths of the cevians from vertices  $V_2$  and  $V_3$  can be found by cyclic permutation of the subscripts given in the formulas above. The derivations of these formulas are provided in the exploration tricev.nb.

### 8.7 Explorations

CIRCLE INSCRIBED IN A RIGHT TRIANGLE.....rttricir.nb

Show that if r is the radius of a circle inscribed in a right triangle with sides a and b and hypotenuse c, then  $r = \frac{1}{2}(a+b-c)$ .

EULER'S TRIANGLE FORMULA..... trieuler.nb If T is a triangle, and P and r are the center and radius of the circle inscribed in T, and Q and R are the center and radius of the circle circumscribing T, show that

$$d^2 = R^2 - 2rR$$

where d is the distance from P to Q.

GERGONNE POINT OF A TRIANGLE..... gergonne.nb Let  $Q_{12}$ ,  $Q_{13}$  and  $Q_{23}$  be the points of contact of the inscribed circle of triangle  $P_1P_2P_3$  with sides  $L_{12}$ ,  $L_{13}$  and  $L_{23}$ , respectively. Show that lines  $P_1Q_{23}$ ,  $P_2Q_{13}$  and  $P_3Q_{12}$  are concurrent. The point of concurrency is called the Gergonne Point of the triangle after J.D. Gergonne (1771–1859), founder-editor of the mathematics journal Annales de Mathematiques.

CENTROID OF A TRIANGLE.....tricent.nb



Show that the centroid of a triangle, as illustrated in the figure, is on a line at a distance  $\bar{y} = b/3$  from the base of the triangle.

ALTITUDE OF A TRIANGLE.....trialt.nb

The *altitude* from vertex A of  $\triangle ABC$  is a line segment from A perpendicular to the side BC (or its extension). Show that the equation of the line containing the altitude from A is

$$(x_3 - x_2)x + (y_3 - y_2)y - x_1(x_3 - x_2) - y_1(y_3 - y_2) = 0$$

where the coordinates of the vertices are  $A(x_1, y_1)$ ,  $B(x_2, y_2)$  and  $C(x_3, y_3)$ .

TRIANGLE ALTITUDE LENGTH. ..... triallen.nb Show that the length, L, of a triangle's altitude (from vertex  $V_3$  to side  $s_1$ ) is given by

$$L = \frac{\sqrt{(s_1 + s_2 - s_3)(s_1 - s_2 + s_3)(-s_1 + s_2 + s_3)(s_1 + s_2 + s_3)}}{2s_3}$$

where  $s_1$ ,  $s_2$  and  $s_3$  are the lengths of the triangle's sides.

CONCURRENT TRIANGLE ALTITUDES......triconn.nb

Show that the three altitudes of any  $\triangle ABC$  are concurrent in a single point (x, y) whose coordinates are given by

$$x = \frac{x'}{D}$$
 and  $y = \frac{y'}{D}$ 

where

$$\begin{aligned} x' &= -(y_1 - y_2)(x_1x_2 + y_3^2) + (y_1 - y_3)(x_1x_3 + y_2^2) - (y_2 - y_3)(x_2x_3 + y_1^2) \\ y' &= +(x_1 - x_2)(y_1y_2 + x_3^2) - (x_1 - x_3)(y_1y_3 + x_2^2) + (x_2 - x_3)(y_2y_3 + x_1^2) \end{aligned}$$

and

$$D = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

and the coordinates of the vertices are  $A(x_1, y_1)$ ,  $B(x_2, y_2)$  and  $C(x_3, y_3)$ . This point is called the *orthocenter* of the triangle.

TRIANGLE SIDE LENGTHS FROM ALTITUDES.....trisides.nb

Prove that the lengths of a triangle's sides whose altitudes are of lengths  $h_1$ ,  $h_2$  and  $h_3$  are given by

$$s_1 = \frac{2h_1H_1}{H}, \ s_2 = \frac{2h_2H_2}{H} \text{ and } s_3 = \frac{2h_3H_3}{H}$$

where  $H_1 = h_2 h_3$ ,  $H_2 = h_1 h_3$  and  $H_3 = h_1 h_2$ , and

$$H = \sqrt{(H_1 + H_2 - H_3)(H_1 - H_2 + H_3)(-H_1 + H_2 + H_3)(H_1 + H_2 + H_3)}.$$

TRIANGLE RADII.....trirad.nb

Prove that the radius, r, of a circle inscribed in a triangle is given by

$$r = \frac{1}{2}\sqrt{\frac{P}{S}}$$

where  $S = s_1 + s_2 + s_3$ ,  $P = (-s_1 + s_2 + s_3)(s_1 - s_2 + s_3)(s_1 + s_2 - s_3)$  and  $s_1$ ,  $s_2$  and  $s_3$  are the lengths of the triangle's sides. Furthermore, prove that the radius, R, of the circle circumscribing the triangle is given by

$$R = \frac{s_1 s_2 s_3}{\sqrt{PS}}.$$

TRIANGLE CEVIAN LENGTHS.....tricev.nb

Prove that the length of the altitude,  $h_1$ , from vertex  $V_1$  of a triangle to the opposite side of the triangle (whose length is  $s_1$ ) is given by

$$h_1 = \frac{\sqrt{PS}}{2s_1}$$

where  $S = s_1 + s_2 + s_3$  and  $P = (-s_1 + s_2 + s_3)(s_1 - s_2 + s_3)(s_1 + s_2 - s_3)$ . Prove that the length of the median,  $m_1$ , from vertex  $V_1$  is given by

$$m_1 = \frac{1}{2}\sqrt{-s_1^2 + 2(s_2^2 + s_3^2)}.$$

Prove that the length of the angle bisector,  $b_1$ , from  $V_1$  is given by

$$b_1 = \frac{\sqrt{Ss_2s_3(-s_1+s_2+s_3)}}{s_2+s_3}.$$

Also show that the formulas for the lengths of the cevians from vertices  $V_2$  and  $V_3$  can be found by cyclic permutation of the subscripts given in the formulas above.

Part III

# Conics

## Chapter 9

## Parabolas

In the branch of mathematics known as *celestial mechanics* it is shown that an object, such as a comet, that falls toward the sun "from infinity" would, if not deflected by the gravitational attraction of other bodies, travel in a path whose shape is a parabola with the sun at its focus. Projectiles in a vacuum on the surface of the earth travel in paths which are nearly parabolic, and projectiles in the air approximate this path with greater or less precision according to their speed, shape and weight. Humans also take advantage of the focusing properties of a parabolic shape in the design of such artifacts as headlights, searchlights and various listening and broadcasting devices. This chapter develops the underlying mathematics of a parabola.

### 9.1 Definitions

A parabola is the locus of a point that moves so that the ratio of its distance from a fixed point and from a fixed line is one. The fixed point, F, is called the *focus* and the fixed line, D, the *directrix*. By definition, the distance from any point P on the parabola to the focus is equal to its distance to the directrix. The ratio PF/PD is called the *eccentricity* e and e = 1. The line FD through the focus perpendicular to the directrix is called the *axis* of the parabola. The midpoint V of the segment FD, obviously a point on the locus, is called the *vertex* of the parabola. The focal chord perpendicular to the axis is called the *latus rectum*.

## 9.2 General Equation of a Parabola

We choose any point  $F(x_1, y_1)$  as the focus and any line  $D \equiv A_1x + B_1y + C_1 = 0$ , where  $A_1^2 + B_1^2 = 1$ , as the directrix. The normalized form of the line is used to simplify the derivation. With reference to these defining elements the equation of the parabola becomes

$$\sqrt{(x-x_1)^2 + (y-y_1)^2} = \pm (A_1x + B_1y + C_1)$$



Figure 9.1: Definition of a parabola.

which can be written as

$$-B_1^2 x^2 + 2A_1 B_1 x y - A_1^2 y^2 + 2(x_1 + A_1 C_1) x + 2(y_1 + B_1 C_1) y + (C_1 - x_1^2 - y_1^2) = 0.$$

This equation is of the form  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$ , an equation of the second degree. One characteristic of the equation is that the second-degree terms in x and y form a perfect square, so the equation may also be written

$$-(B_1x - A_1y)^2 + 2(x_1 + A_1C_1)x + 2(y_1 + B_1C_1)y + (C_1 - x_1^2 - y_1^2) = 0.$$

Moreover, it can be verified that  $B^2 - 4AC = 0$ . Therefore a necessary condition that the equation  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$  represent a parabola is that  $B^2 - 4AC = 0$ . The general equation of a parabola reveals that if the directrix line is parallel to one of the coordinate axes then B = 0 since either  $A_1$  or  $B_1$  will be zero. The equation of a parabola in this position will have no xy term.

## 9.3 Standard Forms of a Parabola

The definition of a parabola makes the shape of the curve depend only upon the distance from the focus to the directrix and not essentially upon the coordinate system. The general equation is complicated because of the choice of a general point and a general line. By an appropriate choice of axes this equation can be simplified; but it will then represent only parabolas in special positions. For example, if axes are chosen so that the focus has coordinates (f, 0) and the directrix the equation x = -f, then the locus definition yields

$$x + f = \sqrt{(x - f)^2 + y^2}$$

which reduces to  $y^2 = 4fx$ . This is one of the *standard forms* of the equation of a parabola. It has a vertex V(0,0). If f is positive the parabola opens to the right; if f is negative it opens to the left. The distance f is called the *focal length* of the parabola and is the distance between the focus and the vertex of the parabola.

Generalizing the location of the vertex point to V(h, k) gives a parabola whose equation is

$$(y-k)^2 = 4f(x-h).$$

This equation is the standard form used in *Descarta2D* as illustrated in the following example.

**Example.** Plot the parabola whose vertex is (2, 1), focal length is 1/2, and opens to the right.

**Solution.** Parabola2D[ $\{h, k\}$ ,  $f, \theta$ ] is the standard representation of a parabola in *Descarta2D* where the coordinates of the vertex are (h, k), the focal length is f and the rotation angle (in radians) about the vertex is  $\theta$ .

In[1]: Sketch2D[{Parabola2D[{2,1},1/2,0]}];



E

L\_\_\_\_\_

The axis of the parabola may also be parallel to the y-axis in which case the equation is

$$(x-h)^2 = 4f(y-k).$$

Descarta2D does not directly use this form of the parabola, but instead simply rotates the  $y^2$  form by the appropriate angle.

**Example.** Plot the parabola whose vertex is (1, -1), focal length is 1/3, and opens upward.

**Solution.** Use the same command as in the previous example with a rotation angle of  $\pi/2$  radians.

In[2]: Sketch2D[{Parabola2D[{1, -1}, 1/3, Pi/2]}];



L

**Example.** Plot the four parabolas whose vertices are (1, 1), (-1, 1), (-1, -1) and (1, -1), focal length 1/3, and axes aligned with the lines x - y = 0 and x + y = 0.

**Solution.** The Descarta2D command Parabola2D[ $\{h, k\}$ ,  $f, \theta$ ] returns the desired parabolas using the Angle2D[*line*] command to find the required values for  $\theta$ .

```
In[3]: axis1 = Line2D[1, -1, 0]; axis2 = Line2D[1, 1, 0];
theta = {Angle2D[axis1], Angle2D[axis2],
Angle2D[axis1] + Pi, Angle2D[axis2] + Pi};
pts = {{1, 1}, {-1, 1}, {-1, -1}, {1, -1}};
Sketch2D[{axis1, axis2,
Map[Parabola2D[pts[[#]], 1/3, theta[[#]]]&,
Range[1, 4]]}];
```



## 9.4 Reduction to Standard Form

The most general equation of a parabola with no xy term present (and hence one whose axis is parallel to one of the coordinate axes) is one of the two forms

(1)  $Ax^2 + Dx + Ey + F = 0$ , axis parallel to the y-axis; (2)  $Cy^2 + Dx + Ey + F = 0$ , axis parallel to the x-axis.

In either case it is easy to reduce this general equation to the corresponding standard form by the process of completing the square.

**Example.** Reduce  $x^2 + 4x + 4y - 8 = 0$  to the equation of a parabola in standard form.

**Solution.** The *Descarta2D* function Loci2D[*quadratic*] returns a list of geometric objects represented by a quadratic equation.

```
In[4]: crv1 = Loci2D[Quadratic2D[1, 0, 0, 4, 4, -8]]
Out[4] {Parabola2D[{-2, 3}, 1, \frac{3\pi}{2}]}
```

The equation in standard form is  $(y-3)^2 = 4(x+2)$ , rotated 270° ( $3\pi/2$  radians) about the point (-2, 3).

```
In[5]: Sketch2D[{crv1}];
```

F

E

F



The following example shows how *Descarta2D* may be used to find the various geometric objects associated with a parabola.

**Example.** Find the focus, directrix, vertex, axis and eccentricity of the parabola represented by the equation  $x^2 - 2x - 8y - 15 = 0$ . Plot the geometric objects.

**Solution.** The function Foci2D[*parabola*] returns a list of one point which is the focus of the parabola; Directrices2D[*parabola*] returns a list of one line which is the directrix of the parabola; Line2D[*parabola*] returns the axis line of the parabola; and the function Eccentricity2D[*parabola*] returns the eccentricity of a parabola (always 1).

 $\textit{Out[7]} \ \{1, \ \texttt{Point2D[} \{1, \ 0\}\}, \ \texttt{Line2D[} 0, \ 1, \ 4\}, \ \texttt{Point2D[} \{1, \ -2\}\}, \ \texttt{Line2D[} -1, \ 0, \ 1]\}$ 

```
In[8]: Sketch2D[{pl, geom}];
```



## 9.5 Parabola from Focus and Directrix

A parabola may be defined in terms of a focus point  $F(x_1, y_1)$  and a directrix line given by  $L \equiv A_2x + B_2y + C_2 = 0$ . Given these two defining elements the parabola's parameters (vertex point V(h, k), focal length f and angle of rotation  $\theta$ ) can be determined. Let

$$d = \frac{A_2 x_1 + B_2 y_1 + C_2}{\sqrt{A_2^2 + B_2^2}}$$

be the signed distance from the focus F to the directrix L, D = |d|, and F' be the projection of F on L. From a previous chapter the coordinates of F' are given by  $(x_1 - ad, y_1 - bd)$ , where  $a = A_2/\sqrt{A_2^2 + B_2^2}$  and  $b = B_2/\sqrt{A_2^2 + B_2^2}$ . The vertex point V is obviously the midpoint of the line segment FF' and has coordinates  $V(h, k) = (x_1 - ad/2, y_1 - bd/2)$ . The focal length f is half of D, f = D/2. The rotation angle  $\theta$  is the angle of the line FF'. L\_\_\_\_\_

F

**Example.** Determine the parabola in standard form defined by the focus point F(1,1) and the directrix line x + y = 0.

**Solution.** The Descarta2D function Parabola2D[point, line] returns the parabola defined by a focus point and a directrix line.

```
In[9]: pl = Parabola2D[Point2D[{1, 1}], Line2D[1, 1, 0]]
Out[9] Parabola2D[\{\frac{1}{2}, \frac{1}{2}\}, \frac{1}{\sqrt{2}}, \frac{\pi}{4}]
In[10]: \{Foci2D[pl], Directrices2D[pl]\} // Simplify
Out[10] \{\{Point2D[\{1, 1\}]\}, \{Line2D[1, 1, 0]\}\}
```

## 9.6 Parametric Equations

The standard form of a parabola used in Descarta2D has the equation

$$(y-k)^2 = 4f(x-h)$$

where (h, k) is the vertex of the parabola and f is the focal length. The axis of this parabola is parallel to the x-axis and the parabola opens to the right (when f > 0). Parabolas in other orientations are obtained by applying a rotation,  $\theta$ , to the standard parabola. Since only the y term is quadratic, it is easy to find one set of parametric equations for a parabola. Let y = k + 2ft be one of the equations; then, solving for t yields

$$t = \frac{(y-k)}{2f}.$$

Substituting this into the equation of the parabola and solving for x yields the two parametric equations

$$\begin{array}{rcl} x &=& h + ft^2 \\ y &=& k + 2ft. \end{array}$$

The parameter value t = 0 produces the vertex point (h, k). Increasing values of t produce points above and to the right of the vertex. Negative values of t produce points that correspond to positive t values reflected in the axis of the parabola. Parameter values  $t = \pm 1$  produce the end points of the focal chord of the parabola. **Example.** Generate seven points on the parabola  $(y + 1)^2 = 2(x - 1)$  at equally spaced parameter values. Plot the curve using a curve length of 20. Generate a second plot of the points on the reflected branch of the parabola.

**Solution.** The Descarta2D command Parabola2D[ $\{h, k\}, f, \theta$ ][t] returns the coordinates at parameter t on the parabola. The option CurveLength2D->n sets the approximate length of unbounded curves plotted by Descarta2D.

```
In[11]: p1 = Parabola2D[{1, -1}, 1/2, 0];
    pts1 = Map[Point2D[p1[#/2]]&, Range[0, 6]];
    pts2 = Map[Point2D[p1[#/2]]&, Range[-6, 0]];
    Sketch2D[{p1, pts1}, CurveLength2D -> 20];
    Sketch2D[{p1, pts2}, CurveLength2D -> 20];
```



Mathematica Hint. Using the CurveLength2D option as part of the Sketch2D command sets the length of all unbounded curves being plotted. If this option is not specified, then a default is used. The initial default set by Descarta2D is 10. To change the default to a new value, n, use the Mathematica command

SetOptions[Sketch2D, CurveLength2D->n].

## 9.7 Explorations

LENGTH OF PARABOLA FOCAL CHORD......pbfocchd.nb

Prove that the length of the focal chord of a parabola is 4f, where f is the focal length.

F

PARABOLA THROUGH THREE POINTS......pb3pts.nb

Show that the parabola passing through the points (0,0), (a,b) and (b,a) whose axis is parallel to the x-axis has vertex (h, k) and focal length f given by

$$h = \frac{(a^2 + ab + b^2)^2}{4ab(a+b)}, \ k = \frac{a^2 + ab + b^2}{2(a+b)} \text{ and } f = -\frac{ab}{4(a+b)}.$$

Furthermore, show that the quadratic representing the parabola is

$$(a+b)y^{2} + abx - (a^{2} + ab + b^{2})y = 0.$$

PARABOLIC Arch.....pbarch.nb



Find the equation of the parabolic arch of base b and height h as shown in the figure. Assume that b and h are positive.

PARABOLA DETERMINANT.....pbdet.nb

Show that the determinant

| y     | $x^2$   | x     | 1 |     |
|-------|---------|-------|---|-----|
| $y_1$ | $x_1^2$ | $x_1$ | 1 | - 0 |
| $y_2$ | $x_2^2$ | $x_2$ | 1 | - 0 |
| $y_3$ | $x_3^2$ | $x_3$ | 1 |     |

represents a parabola  $Ax^2 + Dx + Ey + F = 0$  passing through the points  $(x_1, y_1)$ ,  $(x_2, y_2)$  and  $(x_3, y_3)$ .

PARABOLA INTERSECTION ANGLES......pbang.nb Show that the parabolas  $y^2 = ax$  and  $x^2 = by$  will cut each other at an angle  $\theta$  given by

$$\theta = -\tan^{-1}\frac{a^{\frac{1}{3}}}{2b^{\frac{1}{3}}} + \tan^{-1}\frac{2b^{\frac{1}{3}}}{a^{\frac{1}{3}}}.$$

CIRCLE TANGENT TO A PARABOLA......pbtancir.nb Any line through the point (-3a, 0) cuts the parabola  $y^2 = 4ax$  in the points P and Q. Prove that the circle through P, Q and the focus is tangent to the parabola.

POLAR EQUATION OF A PARABOLA.....polarpb.nb

Show that the polar equation of a parabola opening to the right with vertex at (0,0) is given by

$$r = \frac{4f\cos\theta}{\sin^2\theta}$$

where f is the focal length of the parabola.

## Chapter 10

# Ellipses

The visible universe is filled with ellipses, or near ellipses, traced by celestial bodies revolving around each other, such as planets and the sun. The fact that the angle formed by two focal radii through a point on an ellipse is bisected by the normal to the curve may be used in a device for re-concentrating sound waves, at illustrated in the acoustics of the Mormon Tabernacle in Salt Lake City, Utah. Various types of rotating machinery use elliptical components to generate special types of linear and rotational motions. This chapter develops the mathematics of an ellipse.

## 10.1 Definitions

An *ellipse* is the locus of a point that moves so that the ratio of its distance from a fixed point and from a fixed line is a positive constant less than one. As with the parabola, a focus, directrix and eccentricity are associated with the curve as described in Table 10.1.

Consider the line through the focus perpendicular to the directrix. From the definition PF/PD = e/1 there are obviously two points V and V' which divide the (undirected) segment

| Element            | DESCRIPTION          |
|--------------------|----------------------|
| P(x,y)             | Point on locus       |
| Fixed point $F$    | Focus                |
| Fixed line $D$     | Directrix            |
| Fixed constant $e$ | Eccentricity         |
| e = PF/PD < 1      | Ellipse relationship |

Table 10.1: Definition of an ellipse.



Figure 10.1: Ellipse definition.

FD, internally and externally respectively, into the ratio of e/1. Therefore V and V' are points (on the same side of D) on the ellipse; they are called the *vertices*. The segment VV' is called the *major axis*. By symmetry there is another point F' and another line D' such that F' and D' would serve as the definition of this curve. Thus an ellipse has two foci and two directrices associated in pairs F, D and F', D'. The midpoint of FF', which is also the midpoint of VV', is called the *center* C. It is evident that the locus is contained between the vertices, that it is bounded in all directions and that it is symmetric both with respect to the major axis and to a line perpendicular to it through C.

The focal chord perpendicular to the major axis is called the *latus rectum*. The length of the central chord perpendicular to the major axis is called the *minor axis*.

**Example.** Plot the ellipse with center at coordinates (2, 1), major axis length of 6, minor axis length of 2, and rotated  $30^{\circ}$  ( $\pi/6$  radians) about the center point.

**Solution.** Ellipse2D[ $\{h, k\}$ ,  $a, b, \theta$ ] is the standard representation of an ellipse in Descarta2D. The ellipse is centered at coordinates  $\{h, k\}$ , has semi-major axis of a, semi-minor axis of b and is rotated about the center point by an angle  $\theta$  (the *semi-major axis* is half the length of the major axis; the *semi-minor axis* is half the length of the minor axis).

In[1]: Sketch2D[{Ellipse2D[{2, 1}, 3, 1, Pi / 6]}];

E



10.2 General Equation of an Ellipse

Take any point  $F(x_1, y_1)$  as focus and any line,  $D \equiv A_1x + B_1y + C_1 = 0$  as directrix, where  $A_1^2 + B_1^2 = 1$ . The normalized form of the line is used to simplify the derivation. By definition the equation of the ellipse is

$$\sqrt{(x-x_1)^2 + (y-y_1)^2} = \pm e(A_1x + B_1y + C_1)$$

which may be expanded to

$$(e^2 A_1^2 - 1)x^2 + 2e^2 A_1 B_1 xy + (e^2 B_1^2 - 1)y^2 + 2(x_1 + e^2 A_1 C_1)x + 2(y_1 + e^2 B_1 C_1)y + (e^2 C_1^2 - x_1^2 - y_1^2) = 0.$$

This is of the form  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$ , an equation of the second degree. Moreover, it can be verified that  $B^2 - 4AC = 4(e^2 - 1) < 0$  (when e < 1).

Therefore, a necessary condition that  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$  represent an ellipse is that  $B^2 - 4AC < 0$ . The general equation reveals that if the defining directrix line is parallel to one of the coordinate axes then B = 0, since either  $A_1$  or  $B_1$  will be zero. The equation of an ellipse in this position will have no xy term.

## **10.3 Standard Forms of an Ellipse**

By an appropriate choice of coordinate axes the general equation of an ellipse can be reduced to one of the following *standard forms*.

#### Major Axis Parallel to the x-Axis

The equation of an ellipse in standard position whose major axis is parallel to the x-axis and whose center is at the origin is

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$



Figure 10.2: Ellipse in standard position (x-axis).



Figure 10.3: Ellipse in standard position (y-axis).

|                    | x-axis  | <i>y</i> -axis                                  |
|--------------------|---|---|
| Equation           | $\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$ | $\frac{(x-h)^2}{b^2} + \frac{(y-k)^2}{a^2} = 1$ |
| Center             | C(h,k)  | C(h,k)  |
| Semi-major axis    | a   | a   |
| Semi-minor axis    | b   | b   |
| Vertices           | $V(h \pm a, k)$                                 | $V(h,k\pm a)$                                   |
| Foci               | $F(h\pm ae,k)$                                  | $F(h, k \pm ae)$                                |
| Directrices        | $x = h \pm a/e$                                 | $y = k \pm a/e$                                 |
| Focal chord length | $2b^2/a$  | $2b^2/a$  |
| Eccentricity       | $e = \frac{\sqrt{a^2 - b^2}}{a} < 1$            | $e = \frac{\sqrt{a^2 - b^2}}{a} < 1$            |

Table 10.2: Ellipse equations (x - and y - axis).

where a and b are the lengths of the semi-major and semi-minor axes, respectively. If the ellipse is centered at (h, k), then the equation is

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$$

as shown in Figure 10.2. When an ellipse is in this special position, the formulas for the important points, lines and constants associated with the ellipse are simply determined and are summarized in Table 10.2.

#### Major Axis Parallel to the *y*-axis

The equation of an ellipse in standard position whose major axis is parallel to the y-axis and whose center is at the origin is

$$\frac{x^2}{b^2} + \frac{y^2}{a^2} = 1$$

where a and b are the lengths of the semi-major and semi-minor axes, respectively. If the ellipse is centered at (h, k), then the equation is

$$\frac{(x-h)^2}{b^2} + \frac{(y-k)^2}{a^2} = 1$$

as shown in Figure 10.3. When an ellipse is in this special position, the formulas for the important points, lines and constants associated with the ellipse are simply determined and are summarized in Table 10.2.

## 10.4 Reduction to Standard Form

The most general equation of an ellipse with no xy term (and hence one whose axes are parallel to the coordinate axes) is of the form

 $Ax^{2} + Cy^{2} + Dx + Ey + F = 0, \quad AC > 0.$ 

The condition  $B^2 - 4AC < 0$  reduces to AC > 0 which implies that A and C are of like sign. This equation can be reduced to one of the standard forms by completing the square.

**Example.** Reduce  $x^2 + 4y^2 + 4x = 0$  to standard form and plot.

**Solution.** The Descarta2D function Loci2D[quad] reduces a quadratic equation to a standard form.

```
In[2]: crv1 = Loci2D[Quadratic2D[1, 0, 4, 4, 0, 0]]
Out[2] {Ellipse2D[{-2, 0}, 2, 1, 0]}
```

The equation in standard form is  $\frac{(x+2)^2}{4} + \frac{y^2}{1} = 1.$ 

In[3]: Sketch2D[{crv1}];



F

**Example.** Reduce  $5x^2 + 9y^2 - 10x - 54y + 41 = 0$  to standard form. Find the center, foci, vertices, directrices, the lengths of the semi-major and semi-minor axes and the eccentricity. Plot the geometric objects.

L

**Solution.** The function Loci2D[quad] reduces a quadratic equation to a standard form. The function Point2D[ellipse] returns the center point of an ellipse; the function Foci2D[ellipse] returns a list of the two foci of an ellipse; the function Vertices2D[ellipse] returns a list of the two vertex points of an ellipse; the function Directrices2D[ellipse] returns a list of the two directrix lines of an ellipse; SemiMajorAxis2D[ellipse] and SemiMinorAxis2D[ellipse] return the lengths of the semi-major and semi-minor axes of an ellipse, respectively.

```
In[4]: crv1 = Loci2D[Quadratic2D[5, 0, 9, -10, -54, 41]]
Out[4] \{ Ellipse2D[\{1, 3\}, 3, \sqrt{5}, 0] \}
The standard form of the equation is \frac{(x-1)^2}{9} + \frac{(y-3)^2}{5} = 1.
In[5]: objs = Map[(#[crv1[[1]]])&,
             {Point2D, Foci2D, Vertices2D, Directrices2D,
             SemiMajorAxis2D, SemiMinorAxis2D,
             Eccentricity2D}]
Out[5] {Point2D[{1, 3}], {Point2D[{3, 3}], Point2D[{-1, 3}]},
         \{\text{Point2D}[\{4, 3\}], \text{Point2D}[\{-2, 3\}]\}, \{\text{Line2D}[1, 0, -\frac{11}{2}], \text{Line2D}[1, 0, \frac{7}{2}]\},
        3, \sqrt{5}, \frac{2}{3}
In[6]: Sketch2D[{crv1, Drop[objs, -3]},
            PlotRange -> { {-5, 7}, {-1, 6} },
            CurveLength2D -> 15];
                                6
                               5
                               4
                               3
                               2
                               1
                               0
                              -1
                                   -4
                                         -2
                                                0
                                                       2
                                                             4
                                                                    б
```

## 10.5 Ellipse from Vertices and Eccentricity

Suppose we are given the two vertices,  $V_1(x_1, y_1)$  and  $V_2(x_2, y_2)$ , and the eccentricity, e, of an ellipse and we wish to find the standard equation of the ellipse. The center point (h, k) of the ellipse is clearly the midpoint between the vertices and is given by

$$\left(\frac{x_1+x_2}{2},\frac{y_1+y_2}{2}\right).$$

The length of the semi-major axis, a, is one-half the distance between the vertices, yielding  $a = |V_1V_2|/2$ . The eccentricity is given by

$$e = \frac{\sqrt{a^2 - b^2}}{a},$$

so, solving for b gives the length of the semi-minor axis as

$$b = a\sqrt{1 - e^2}.$$

The line through the two vertex points determines the rotation angle of the ellipse as

$$\theta = \tan^{-1}(x_2 - x_1, y_2 - y_1).$$

**Example.** Find the ellipse whose vertices are (4, 2) and (-2, 1), and whose eccentricity is 7/8.

**Solution.** The Descarta2D function Ellipse2D[{point, point}, e] returns the ellipse whose vertices are the given points with the specified eccentricity.

```
In[7]: p1 = Point2D[{4, 2}];
    p2 = Point2D[{-2, 1}];
    e1 = Ellipse2D[{p1, p2}, 7/8] //N
Out[7] Ellipse2D[{1., 1.5}, 3.04138, 1.4724, 0.165149]
```

```
In[8]: Sketch2D[{p1, p2, e1}];
```



E

L\_\_\_\_\_

### **10.6** Ellipse from Foci and Eccentricity

It is evident from Table 10.2 that the distance between the foci of an ellipse is  $|F_1F_2| = 2ae$ and that the distance between the vertices is  $|V_1V_2| = 2a$ . Therefore, the eccentricity, e, given by

$$e = \frac{2ae}{2a} = \frac{|F_1F_2|}{|V_1V_2|}$$

is the ratio of the distance between the foci to the distance between the vertices. This relationship allows us to construct an ellipse by specifying the two foci and the eccentricity. The semi-major axis length, a, is given by  $a = |F_1F_2|/2e$  and the semi-minor axis length is  $b = a\sqrt{1-e^2}$ . The center point of the ellipse is clearly the midpoint of the two foci and the angle of rotation is

$$\theta = \tan^{-1}(x_2 - x_1, y_2 - y_1),$$

where  $F_1(x_1, y_1)$  and  $F_2(x_2, y_2)$  are the coordinates of the foci.

**Example.** Find the ellipse whose foci are (-1, -1) and (1, 1) and whose eccentricity is 1/2.

**Solution.** The Descarta2D function Ellipse2D[point, point, e] constructs an ellipse given the two foci points and the eccentricity.

```
In[9]: el = Ellipse2D[Point2D[\{-1, -1\}], Point2D[\{1, 1\}], 1/2]
out[9] Ellipse2D[\{0, 0\}, 2\sqrt{2}, \sqrt{6}, \frac{\pi}{4}]
In[10]: \{Foci2D[el], Eccentricity2D[el]\}
out[10] \{\{Point2D[\{1, 1\}], Point2D[\{-1, -1\}]\}, \frac{1}{2}\}
```

## 10.7 Ellipse from Focus and Directrix

Given the focus point  $F(x_1, y_1)$ , the directrix line  $L \equiv px + qy + r = 0$ , and the eccentricity, 0 < e < 1, of an ellipse we wish to determine the standard equation of the ellipse. The rotation angle of the ellipse is the angle the line perpendicular to L makes with the +x-axis and is given by  $\theta = \tan^{-1}(p, q)$ . The distance, d, from F to L is given by

$$d = \sqrt{\frac{(px_1 + qy_1 + r)^2}{p^2 + q^2}}.$$

It is clear from Table 10.2 that the distance from F to L is also given by d = a/e - ae. Solving for a (the length of the semi-major axis) yields

$$a = d \frac{e}{(1 - e^2)}.$$

Table 10.2 shows that the eccentricity, e, is related to the lengths of the semi-major and semi-minor axes, a and b, respectively, by

$$e = \frac{\sqrt{a^2 - b^2}}{a}.$$

Solving this equation for b yields

\_\_\_\_\_

$$b = a\sqrt{1 - e^2}.$$

Table 10.2 reveals that the distance from the focus F to the center C(h, k) is given by ae. If F' is the projection of F onto L, then we can find the center point C of the ellipse by offsetting F in the direction from F to F' a distance -ae. This computation is easily accomplished using *Descarta2D* and is provided in the exploration ellfd.nb. The resulting defining constants of the ellipse are given by

$$h = x_1 + \frac{paeD}{d}, \ k = y_1 + \frac{qaeD}{d},$$
  
 $a = d\frac{e}{(1 - e^2)}, \ b = a\sqrt{1 - e^2},$ 

where

$$d = \sqrt{\frac{(px_1 + qy_1 + r)^2}{p^2 + q^2}}$$
 and  $D = \frac{px_1 + qy_1 + r}{p^2 + q^2}$ .

**Example.** Find the ellipse whose focus point is (3, 2), directrix line x - y + 2 = 0 and eccentricity is 1/4.

**Solution.** The Descarta2D function Ellipse2D[point, line, e] constructs an ellipse for the focus, directrix and eccentricity.

In[11]: e1 = Ellipse2D[p1 = Point2D[{3, 2}], 12 = Line2D[1, -1, 2], 1/4]

Out[11] Ellipse2D[
$$\{\frac{31}{10}, \frac{19}{10}\}, \frac{2\sqrt{2}}{5}, \sqrt{\frac{3}{10}}, \frac{3\pi}{4}$$
]  
In[12]: {Foci2D[e1],  
Directrices2D[e1],  
Eccentricity2D[e1]} // Simplify  
Out[12] {{Point2D[{3, 2}], Point2D[ $\{\frac{16}{5}, \frac{9}{5}\}$ ]},  
{Line2D[-1, 1, -2], Line2D[-5, 5, 22]},  $\frac{1}{4}$ }

F

## 10.8 Parametric Equations

The parametric equations for a standard ellipse

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$$

are very similar to those of a circle, with the exception that the radius is replaced by either the length of the semi-major axis, a, or the semi-minor axis, b. The appropriate equations are

 $x = h + a\cos\theta$  and  $y = k + b\sin\theta$ 

where (h, k) is the center of the ellipse, a and b are the lengths of the semi-major and semiminor axes, respectively, and parameter values in the range  $0 \le t < 2\pi$  generate a complete curve. The validity of these equations can be verified by direct substitution.

Example. Plot 16 points on the ellipse

$$\frac{x^2}{16} + \frac{y^2}{4} = 1$$

at equally spaced parameter values.

**Solution.** The Descarta2D function Ellipse2D[ $\{h, k\}$ ,  $a, b, \theta$ ][t] returns the coordinates of a point at parameter value t on the ellipse.

As with the circle, a pair of rational equations may be used as the parametric equations for an ellipse. The ellipse

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

٦

has the parametric equations

$$x = a\left(\frac{1-t^2}{1+t^2}\right)$$
 and  $y = b\left(\frac{2t}{1+t^2}\right)$ .

Values of t in the range  $0 \le t \le 1$  generate coordinates on the ellipse in the first quadrant. The point (-a, 0), which is on the ellipse, cannot be generated using these equations.

**Example.** Plot the ellipse  $x^2/4 + y^2 = 1$  using the rational parametric equations in the parameter range  $-10 \le t \le 10$ .

**Solution.** The *Mathematica* function ParametricPlot plots curves defined by parametric equations.

## 10.9 Explorations

LENGTH OF ELLIPSE FOCAL CHORD..... elllen.nb Prove that the length of the focal chord of an ellipse is  $2b^2/a$ , where *a* is the length of the semi-major axis and *b* is the length of the semi-minor axis.

SUM OF FOCAL DISTANCES OF AN ELLIPSE..... ellips2a.nb Show that the sum of the distances from the two foci to any point on an ellipse is 2a, where a is the length of the semi-major axis.

F

F

ELLIPSE FROM FOCUS AND DIRECTRIX.....ellfd.nb

Show that the ellipse with focus  $F(x_1, y_1)$ , directrix line  $L \equiv px+qy+r = 0$  and eccentricity, 0 < e < 1, is defined by the constants

$$h = x_1 + \frac{paeD}{d}, \quad k = y_1 + \frac{qaeD}{d},$$
  
 $a = d\frac{e}{(1-e^2)}, \quad b = a\sqrt{1-e^2}, \quad \theta = \tan^{-1}(p,q),$ 

where

$$d = \sqrt{\frac{(px_1 + qy_1 + r)^2}{p^2 + q^2}}$$
 and  $D = \frac{px_1 + qy_1 + r}{p^2 + q^2}$ 

FOCUS OF ELLIPSE IS POLE OF DIRECTRIX.....elfocdir.nb Show that the focus of an ellipse is the pole of the corresponding directrix.

ELLIPSE LOCUS, DISTANCE FROM TWO LINES.....elldist.nb A point moves so that the sum of the squares of its distances from two intersecting straight lines is a constant. Prove that its locus is an ellipse.

SIMILAR ELLIPSES.....ellsim.nb All ellipses of equal eccentricity are essentially *similar* in that by a proper choice of scales (and axes) they can be made to coincide. Show this property is true for two ellipses of equal eccentricity centered at the origin.

POLAR EQUATION OF AN ELLIPSE.....polarell.nb

Show that the polar equation of an ellipse with a horizontal major axis and centered at (0,0) is given by

$$r = \frac{ab}{\sqrt{a^2 \sin^2 \theta + b^2 \cos^2 \theta}}$$

where a and b are the lengths of the semi-major and semi-minor axes, respectively.

APOAPSIS AND PERIAPSIS OF AN ELLIPSE.....ellrad.nb

Show that the greatest (*apoapsis*) and least (*periapsis*) radial distance of a point on an ellipse as measured from a focus point is given by r = a(1 + e) and r = a(1 - e), respectively, where e is the eccentricity and a is the length of the semi-major axis of the ellipse.

## Chapter 11

# Hyperbolas

The equations of a hyperbola are in many ways similar to those of an ellipse, the forms often only differing by a + or - sign. The properties and characteristics of a hyperbola, however, are somewhat less intuitive than an ellipse, possibly because the curve has two disjoint branches or because it extends to infinity. This chapter describes the detailed mathematics of a hyperbola.

## 11.1 Definitions

A hyperbola is the locus of a point that moves so that the ratio of its distance from a fixed point and from a fixed line is a constant greater than one. As with the parabola and ellipse, a focus, directrix and eccentricity are associated with the curve as shown in Table 11.1.

Consider the line through the focus perpendicular to the directrix. From the definition PF/PD = e/1 there are obviously two points V and V' which divide the (undirected) segment FD, internally and externally respectively, in the ratio of e/1. Therefore, V and V' are points (on *opposite* sides of D) on the hyperbola; they are called the *vertices*. The segment VV' is called the *transverse axis*. By symmetry, there is another point F' and another line D' such

 Table 11.1: Hyperbola definition.

| Element            | DESCRIPTION            |  |
|--------------------|------------------------|--|
| P(x,y)             | Point on locus         |  |
| Fixed point $F$    | Focus                  |  |
| Fixed line $D$     | Directrix              |  |
| Fixed constant $e$ | Eccentricity           |  |
| e = PF/PD > 1      | Hyperbola relationship |  |



Figure 11.1: Hyperbola definition.

that F' and D' would serve in the definition of this curve. Thus, a hyperbola has two foci and two directrices associated in pairs F, D and F', D'. The midpoint of FF', which is also the midpoint of VV', is called the *center* C. There are two tangent lines through C whose points of contact are at an infinite distance from C. These are called the *asymptotes* of the hyperbola. The focal chord perpendicular to the transverse axis is called the *latus rectum*.

A line through C perpendicular to the transverse axis does not intersect the hyperbola in real points. But the portion of it, bisected by C, which is equal in length to the parallel segment through V contained between the asymptotes is called the *conjugate axis*.

**Example.** Plot the hyperbola with center at coordinates (2, 1), transverse axis length of 1, conjugate axis length of 3/4 and rotated  $30^{\circ}$  ( $\pi/6$  radians) about the center point.

**Solution.** Hyperbola2D[ $\{h, k\}$ ,  $a, b, \theta$ ] is the standard representation of a hyperbola in *Descarta2D*. The hyperbola is centered at coordinates (h, k), has semi-transverse axis of a, semi-conjugate axis of b and is rotated about the center point by an angle  $\theta$  (the *semi-transverse axis* is half the length of the transverse axis; the *semi-conjugate axis* is half the length of the conjugate axis).

In[1]: Sketch2D[{Hyperbola2D[{2,1},1,3/4,Pi/6]}];

E

L



## 11.2 General Equation of a Hyperbola

Take any point  $F(x_1, y_1)$  as focus and any line,  $D \equiv A_1x + B_1y + C_1 = 0$  as directrix, where  $A_1^2 + B_1^2 = 1$ . The normalized form of the line is used to simplify the derivation. By definition the equation of the hyperbola is

$$\sqrt{(x-x_1)^2 + (y-y_1)^2} = \pm e(A_1x + B_1y + C_1)$$

which may be expanded to

$$(e^2 A_1^2 - 1)x^2 + 2e^2 A_1 B_1 xy + (e^2 B_1^2 - 1)y^2 + 2(x_1 + e^2 A_1 C_1)x + 2(y_1 + e^2 B_1 C_1)y + (e^2 C_1^2 - x_1^2 - y_1^2) = 0.$$

This is of the form  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$ , an equation of the second degree. Moreover, it can be verified that  $B^2 - 4AC = 4(e^2 - 1) > 0$  (when e > 1).

Therefore, a necessary condition that  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$  represent a hyperbola is that  $B^2 - 4AC > 0$ . The general equation reveals that if the defining directrix line is parallel to one of the coordinate axes then B = 0, since either  $A_1$  or  $B_1$  will be zero. The equation of a hyperbola in this position will have no xy term.

## 11.3 Standard Forms of a Hyperbola

By an appropriate choice of coordinate axes the general equation of a hyperbola can be reduced to one of the following *standard forms*.

#### Transverse Axis Parallel to the *x*-Axis

The equation of a hyperbola in standard position whose transverse axis is parallel to the *x*-axis and whose center is at the origin is

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$$



Figure 11.2: Hyperbola in standard position (x-axis).



Figure 11.3: Hyperbola in standard position (y-axis).
|                      | x-axis  | <i>y</i> -axis                                   |
|----------------------|---|--|
| Equation             | $\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1$ | $-\frac{(x-h)^2}{b^2} + \frac{(y-k)^2}{a^2} = 1$ |
| Center               | C(h,k)  | C(h,k)   |
| Semi-transverse axis | a   | a  |
| Semi-conjugate axis  | b   | b  |
| Vertices             | $V(h\pm a,k)$                                   | $V(h,k\pm a)$                                    |
| Foci                 | $F(h \pm ae, k)$                                | $F(h,k\pm ae)$                                   |
| Directrices          | $x = h \pm a/e$                                 | $y = k \pm a/e$                                  |
| Asymptotes           | $bx \pm ay - (bh \pm ak) = 0$                   | $ax \pm by - (ah \pm bk) = 0$                    |
| Focal chord length   | $2b^2/a$  | $2b^2/a$   |
| Eccentricity         | $e = \frac{\sqrt{a^2 + b^2}}{a} > 1$            | $e = \frac{\sqrt{a^2 + b^2}}{a} > 1$             |

Table 11.2: Hyperbola definition (x- and y-axis).

where a and b are the lengths of the semi-transverse and semi-conjugate axes, respectively. If the hyperbola is centered at (h, k), then the equation is

$$\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1$$

as shown in Figure 11.2. When a hyperbola is in this special position, the formulas for the important points, lines and constants associated with the hyperbola are simply determined and are summarized in Table 11.2.

The lengths of the transverse axis, conjugate axis, focal chord and the value of the eccentricity are independent of the origin and are also given in Table 11.2. Note that the equations of the asymptotes can be obtained directly from the equation of the hyperbola in standard form by replacing the one on the right-hand side of the equation with a zero. The left-hand side of the equation will then factor into two linear terms which are the asymptotes of the hyperbola.

#### Transverse Axis Parallel to the *y*-Axis

The equation of a hyperbola in standard position whose transverse axis is parallel to the y-axis and whose center is at the origin is

$$-\frac{x^2}{b^2} + \frac{y^2}{a^2} = 1$$

|    | TRANSVERSE AXIS       | Center at $(h,k)$                                |
|----|-----------------------|--|
| H  | parallel to $x$ -axis | $\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1$  |
| H' | parallel to $y$ -axis | $-\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$ |

Table 11.3: Conjugate hyperbolas.

where a and b are the lengths of the semi-transverse and semi-conjugate axes, respectively. If the hyperbola is centered at (h, k), then the equation is

$$-\frac{(x-h)^2}{b^2} + \frac{(y-k)^2}{a^2} = 1$$

as shown in Figure 11.3. When a hyperbola is in this special position, the formulas for the important points, lines and constants associated with the hyperbola are simply determined and are summarized in Table 11.2

The lengths of the semi-transverse axis, semi-conjugate axis, focal chord and the value of the eccentricity are independent of the origin and are also shown in Table 11.2. These constants have the same values as a hyperbola whose transverse axis is parallel to the x-axis.

#### Conjugate and Rectangular Hyperbolas

Two hyperbolas with the same center are *conjugate hyperbolas* if the transverse axis of one coincides with the conjugate axis of the other. The equations of two conjugate hyperbolas H and H' in standard form are shown in Table 11.3. It is evident that if a is the semi-transverse axis of H, then a is the semi-conjugate axis of H', and vice versa. Conjugate hyperbolas have the same asymptotes and their foci lie on a circle with center at the center of the hyperbolas.

**Example.** Write the equation of the hyperbola whose center is (-2, 1), transverse axis length 6 (parallel to the x-axis), and conjugate axis length 8. Determine its eccentricity, foci and vertices. Find the equations of its directrices and asymptotes. Plot the geometric objects.

Solution. The equation can be written directly using the standard form as

$$\frac{(x+2)^2}{9} - \frac{(y-1)^2}{16} = 1$$

In Descarta2D this hyperbola is written as Hyperbola2D[{-2,1},3,4,0].

```
In[2]: h1 = Hyperbola2D[{-2, 1}, 3, 4, 0];
```

The Descarta2D function Eccentricity2D[hyperbola] returns the eccentricity of the hyperbola.

In[3]: Eccentricity2D[h1] Out[3]  $\frac{5}{3}$ 

The Descarta2D function Foci2D[hyperbola] returns a list of the two focus points; the function Vertices2D[hyperbola] returns a list of the two vertex points; the function Directrices2D[hyperbola] returns a list of the two directrix lines; the function Asymptotes2D[hyperbola] returns a list of the two asymptote lines.

```
In[4]: objs = Map[(#[h1])&,
            {Foci2D, Vertices2D, Directrices2D, Asymptotes2D}]
Out[4] {{Point2D[{3,1}], Point2D[{-7,1}]}, {Point2D[{1,1}], Point2D[{-5,1}]},
          \left\{\text{Line2D}\left[1,\,0,\,\frac{1}{5}\right],\,\text{Line2D}\left[1,\,0,\,\frac{19}{5}\right]\right\},\,\left\{\text{Line2D}\left[4,\,3,\,5\right],\,\text{Line2D}\left[4,\,-3,\,11\right]\right\}\right\}
In[5]: Sketch2D[{h1, objs},
              CurveLength2D -> 40,
              PlotRange -> {{-14, 10}, {-9, 11}}];
                                        10
                                      7.5
                                          5
                                      2.5
                                          0
                                    -2.5
                                        -5
                                    -7.5
                                                                      5
                                               -10
                                                       -5
                                                               0
                                                                            10
```

**Example.** Plot the hyperbola whose equation is  $4x^2 - y^2 + 36 = 0$  along with its conjugate.

**Solution.** The function Loci2D[quad] constructs a list containing the objects represented by a quadratic equation; Hyperbola2D[hyperbola, Conjugate2D] constructs the conjugate of a hyperbola.

\_\_\_\_

-

```
In[6]: {h1} = Loci2D[Quadratic2D[4, 0, -1, 0, 0, 36]]
Out[6] {Hyperbola2D[{0, 0}, 6, 3, π/2]}
In[7]: h2 = Hyperbola2D[h1, Conjugate2D]
Out[7] Hyperbola2D[{0, 0}, 3, 6, 0]
In[8]: f = {{fla, f1b} = Foci2D[h1], {f2a, f2b} = Foci2D[h2]};
c1 = Circle2D[f1a, f1b, f2a]; IsOn2D[f2b, c1]
Out[8] True
```

The statement IsOn2D[f2b, c1], by returning True, shows that the foci of both hyperbolas are on a common circle.



A rectangular (or equilateral) hyperbola is one in which the transverse and conjugate axes are equal in length, in which case the asymptotes are at right angles to each other.

## 11.4 Reduction to Standard Form

F

F

The most general equation of a hyperbola with no xy term (and hence one whose axes are parallel to the coordinate axes) is of the form

$$Ax^{2} + Cy^{2} + Dx + Ey + F = 0, \quad AC < 0.$$

The condition  $B^2 - 4AC > 0$  reduces to AC < 0 which implies that A and C are of *opposite* sign. This equation can be reduced to one of the standard forms by completing the square.

**Example.** Reduce  $x^2 - y^2 - 2x - y + 1 = 0$  to standard form and plot.

**Solution.** The Descarta2D function Loci2D[quad] constructs a list containing the objects represented by the quadratic.

```
In[10]: h1 = Loci2D[Quadratic2D[1, 0, -1, -2, -1, 1]]Out[10] \{Hyperbola2D[\{1, -\frac{1}{2}\}, \frac{1}{2}, \frac{1}{2}, \frac{\pi}{2}]\}
```

The equation in standard form is

$$-\frac{(x-1)^2}{\frac{1}{4}} + \frac{(y+\frac{1}{2})^2}{\frac{1}{4}} = 1.$$

This is a rectangular hyperbola with  $a = b = \frac{1}{2}$ .

In[11]: Sketch2D[{h1}];



## 11.5 Hyperbola from Vertices and Eccentricity

Suppose we are given the two vertices,  $V_1(x_1, y_1)$  and  $V_2(x_2, y_2)$  and the eccentricity, e, of a hyperbola and we wish to find the standard equation of the hyperbola. The center point (h, k) of the hyperbola is clearly the midpoint between the vertices and is given by

$$\left(\frac{x_1+x_2}{2},\frac{y_1+y_2}{2}\right).$$

The length of the semi-transverse axis, a, is one-half the distance between the vertices, yielding  $a = |V_1V_2|/2$ . The eccentricity is given by

$$e = \frac{\sqrt{a^2 + b^2}}{a},$$

-

so, solving for b gives the length of the semi-conjugate axis as

$$b = a\sqrt{e^2 - 1}.$$

The line through the two vertex points determines the rotation angle of the hyperbola as

 $\theta = \tan^{-1}(x_2 - x_1, y_2 - y_1).$ 

**Example.** Find the hyperbola whose vertices are (4, 2) and (-2, 1), and whose eccentricity is 3/2.

**Solution.** The Descarta2D function Hyperbola2D[{point, point}, e] returns the hyperbola whose vertices are the given points with the specified eccentricity.

```
In[12]: pl = Point2D[{4, 2}];
    p2 = Point2D[{-2, 1}];
    h1 = Hyperbola2D[{p1, p2}, 3/2] // N
Out[12] Hyperbola2D[{1., 1.5}, 3.04138, 3.40037, 0.165149]
```

```
In[13]: Sketch2D[{p1, p2, h1}];
```



## 11.6 Hyperbola from Foci and Eccentricity

It is evident from Table 11.2 that the distance between the foci of a hyperbola is given by  $|F_1F_2| = 2ae$  and that the distance between the vertices is  $|V_1V_2| = 2a$ . Therefore, the eccentricity, e, given by

$$e = \frac{2ae}{2a} = \frac{|F_1F_2|}{|V_1V_2|}$$

F

is the ratio of the distance between the foci to the distance between the vertices. This relationship allows us to construct a hyperbola by specifying the two foci and the eccentricity. The semi-transverse axis length, a, is given by  $a = |F_1F_2|/2e$  and the semi-conjugate axis length is  $b = a\sqrt{e^2 - 1}$ . The center point of the hyperbola is clearly the midpoint of the two foci and the angle of rotation is  $\arctan(x_2 - x_1, y_2 - y_1)$ , where  $F_1(x_1, y_1)$  and  $F_2(x_2, y_2)$  are the coordinates of the foci.

**Example.** Find the hyperbola whose foci are (-1, -1) and (1, 1) and whose eccentricity is 3/2.

**Solution.** The function Hyperbola2D[*point*, *point*, *e*] constructs a hyperbola given the two foci points and the eccentricity.

```
In[14]: h1 = Hyperbola2D[Point2D[\{-1, -1\}], Point2D[\{1, 1\}], 3/2]
Out[14] Hyperbola2D[\{0, 0\}, \frac{2\sqrt{2}}{3}, \frac{\sqrt{10}}{3}, \frac{\pi}{4}]
In[15]: \{Foci2D[h1], Eccentricity2D[h1]\}
Out[15] \{ \{Point2D[\{1, 1\}], Point2D[\{-1, -1\}]\}, \frac{3}{2} \}
```

## 11.7 Hyperbola from Focus and Directrix

Given the focus point  $F(x_1, y_1)$ , the directrix line  $L \equiv px + qy + r = 0$  and the eccentricity, e > 1, of a hyperbola we wish to determine the standard equation of the hyperbola. The rotation angle of the hyperbola is the angle the line perpendicular to L makes with the +x-axis and is given by  $\theta = \tan^{-1}(p,q)$ . The distance, d, from F to L is given by

$$d = \sqrt{\frac{(px_1 + qy_1 + r)^2}{p^2 + q^2}}$$

It is clear from Table 11.2 that the distance from F to L is also given by d = ae - a/e. Solving for a (the length of the semi-transverse axis) yields

$$a = d\frac{e}{(e^2 - 1)}.$$

Table 11.2 shows that the eccentricity, e, is related to the lengths of the semi-transverse and semi-conjugate axes, a and b, respectively, by

$$e = \frac{\sqrt{a^2 + b^2}}{a}.$$

Solving this equation for b yields

$$b = a\sqrt{e^2 - 1}.$$

Table 11.2 reveals that the distance from the focus F to the center C(h, k) is given by ae. If F' is the projection of F onto L, then we can find the center point C(h, k) of the hyperbola by offsetting F in the direction from F to F' a distance ae. This computation is easily accomplished using *Descarta2D* and is provided in the exploration hypfd.nb. The defining constants of the hyperbola so computed are

$$h = x_1 - \frac{paeD}{d}, \quad k = y_1 - \frac{qaeD}{d},$$
  
 $a = d\frac{e}{(e^2 - 1)}, \quad b = a\sqrt{e^2 - 1},$ 

where

F

$$d = \sqrt{\frac{(px_1 + qy_1 + r)^2}{p^2 + q^2}}$$
 and  $D = \frac{px_1 + qy_1 + r}{p^2 + q^2}$ .

**Example.** Find the hyperbola whose focus is (3, 2), directrix line is x - y + 2 = 0 and eccentricity is 5.

**Solution.** The Descarta2D function Hyperbola2D [point, line, e] constructs a hyperbola from the focus, directrix and eccentricity.

```
In[16]: h1 = Hyperbola2D[p1 = Point2D[{3, 2}], 12 = Line2D[1, -1, 2], 5]
Out[16] Hyperbola2D[{\frac{23}{16}, \frac{57}{16}}, \frac{5}{8\sqrt{2}}, \frac{5\sqrt{3}}{4}, \frac{3\pi}{4}]
In[17]: \{Foci2D[h1], \\ Directrices2D[h1], \\ Eccentricity2D[h1] \} // Simplify
Out[17] \{\{Point2D[\{-\frac{1}{8}, \frac{41}{8}\}], Point2D[{3, 2}]\}, \\ \{Line2D[-4, 4, -9], Line2D[-1, 1, -2]\}, 5\}
```

## 11.8 Parametric Equations

The standard form of a hyperbola used in Descarta2D has the equation

$$\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1$$

where (h, k) is the center of the hyperbola, and a and b are the lengths of the semi-transverse and semi-conjugate axes, respectively. The axis of this hyperbola is parallel to the x-axis and the hyperbola opens to the right and left. Hyperbolas in other orientations are obtained by applying a rotation,  $\theta$ , to the standard hyperbola. The parametric equations for a hyperbola are similar to those of an ellipse, except hyperbolic functions are used instead of standard trigonometric functions. The parametric equations are

$$x = h + \cosh t$$
 and  $y = k + \sinh t$ .

The parameter value t = 0 produces the vertex point on the right branch of the hyperbola. Increasing values of t produce points above and to the right of this vertex. Negative values of t produce points that correspond to positive t values reflected in the transverse axis of the hyperbola. All of the points on the right branch need to be reflected in the conjugate axis of the hyperbola to produce the left branch of the curve.

In Descarta2D the parametric equations of a hyperbola are scaled by a factor s so that the end points of the focal chord are at the parameter values -1 and +1. Specifically, the equations used in Descarta2D are

$$x = h + a \cosh(st)$$
 and  $y = k + b \sinh(st)$ 

where

F

$$s = \cosh^{-1} e$$

and e is the eccentricity of the hyperbola. The validity of these equations can be verified by direct substitution.

**Example.** Plot eight points at equal parameter values on the upper and lower portions of the right branch of the hyperbola  $x^2/4 - y^2/2 = 1$ .

**Solution.** The command Hyperbola2D[ $\{h, k\}$ ,  $a, b, \theta$ ][t] returns the coordinates at parameter t on the hyperbola.

```
In[18]: h1 = Hyperbola2D[{0, 0}, 2, Sqrt[2], 0];
    pts1 = Map[Point2D[h1[#/3]]&, Range[0, 7]];
    pts2 = Map[Point2D[h1[#/3]]&, Range[-7, 0]];
    pr = PlotRange -> {{-6, 10}, {-5, 5}};
    Sketch2D[{h1, pts1}, pr];
    Sketch2D[{h1, pts2}, pr];
```

-1



As with the ellipse, a pair of rational equations may be used as the parametric equations for a hyperbola. The hyperbola

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$$

has the parametric equations

L\_\_\_\_\_

$$x = a\left(\frac{1+t^2}{1-t^2}\right)$$
 and  $y = b\left(\frac{2t}{1-t^2}\right)$ .

Values of t in the range  $0 \le t < 1$  generate coordinates on the hyperbola in the first quadrant. The other portions of the curve can be generated by reflecting the coordinates generated by these equations.

**Example.** Plot the hyperbola  $x^2/25 - y^2 = 1$  using the rational parametric equations in the parameter range  $-1/2 \le t \le 1/2$ .

**Solution.** The Mathematica function ParametricPlot plots curves defined by parametric equations.

## 11.9 Explorations

LENGTH OF HYPERBOLA FOCAL CHORD......hyplen.nb

Prove that the length of the focal chord of a hyperbola is  $2b^2/a$ , where a is the length of the semi-transverse axis and b is the length of the semi-conjugate axis.

FOCAL DISTANCES OF A HYPERBOLA.....hyp2a.nb

Show that the difference of the distances from the two foci to any point on a hyperbola is 2a, where a is the length of the semi-transverse axis.

HYPERBOLA FROM FOCUS AND DIRECTRIX......hypfd.nb

Show that the hyperbola with focus  $F(x_1, y_1)$ , directrix  $L \equiv px + qy + r = 0$  and eccentricity, e > 1 is defined by the constants

$$h = x_1 - \frac{paeD}{d}, \quad k = y_1 - \frac{qaeD}{d},$$
$$a = d\frac{e}{(e^2 - 1)}, \quad b = a\sqrt{e^2 - 1}, \quad \theta = \tan^{-1}(p, q),$$

where

$$d = \sqrt{\frac{(px_1 + qy_1 + r)^2}{p^2 + q^2}}$$
 and  $D = \frac{px_1 + qy_1 + r}{p^2 + q^2}$ .

RECTANGULAR HYPERBOLA DISTANCES.....hypinv.nb

Show that the distance of any point on a rectangular hyperbola from its center varies inversely as the perpendicular distance from its polar to the center.

 $1/(e_1^2) + 1/(e_2^2) = 1.$ 

POLAR EQUATION OF A HYPERBOLA ......polarhyp.nb

Show that the polar equation of a hyperbola with a horizontal transverse axis and centered at (0,0) is given by

$$r = \frac{ab}{\sqrt{b^2 \cos^2 \theta - a^2 \sin^2 \theta}}$$

where a and b are the lengths of the semi-transverse and semi-conjugate axes, respectively.

TRIGONOMETRIC PARAMETRIC EQUATIONS......hyptrig.nb Show that the parametric equations

$$x = a \sec \theta$$
 and  $y = b \tan \theta$ 

represent the hyperbola

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1.$$

\_\_\_\_\_

## Chapter 12

# **General Conics**

In previous chapters we have examined specific forms of an equation of the second degree resulting in a detailed understanding of circles, parabolas, ellipses and hyperbolas. In this chapter we will study the general second-degree equation itself resulting in a more complete understanding of the equation.

## 12.1 Conic from Quadratic Equation

In this section we will present a method for converting a general quadratic equation of the form  $Q \equiv Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$  to a conic curve in a standard form. The method involves examining the coefficients of the equation and applying algebraic operations to the equation which successively simplify the equation until a standard form can be recognized by inspection. The general approach involves the following steps:

- If the quadratic equation is one of several special forms, then the standard form of the curve can be determined by inspection. The following curves have a quadratic form that can be directly recognized: (1) a single point, (2) a single line, (3) two lines (parallel, co-incident or intersecting), (4) a circle, parabola, ellipse or hyperbola in standard position and (5) several forms with no real locus (imaginary).
- If the quadratic equation has first-degree terms  $(D \neq 0 \text{ or } E \neq 0)$ , translate the equation to a coordinate system that eliminates the x or y terms. Once the curve is identified, translate the standard curve back to the original position.
- If there is an xy cross-product term in the quadratic equation  $(B \neq 0)$ , eliminate it by applying an appropriate rotation. After the standard curve is identified, rotate it back to the original position.

The following subsections describe each of these reduction steps in more detail.

#### Linear Polynomial

 $\vdash$ 

Form:  $Q \equiv Dx + Ey + F = 0$ , D and E not both zero.

If the first three coefficients of Q are equal to zero, and coefficients D and E are not both zero, then the equation Q represents a single straight line Dx + Ey + F = 0.

**Example.** Show that Descarta2D will detect a quadratic equation as a line if the first three coefficients are zero. Use the line x - 2y + 4 = 0 as an example.

Solution. Use the Descarta2D function Loci2D[quad].

```
In[1]: Clear[x, y];
Loci2D[Quadratic2D[x - 2y + 4 == 0, {x, y}]]
Out[1] {Line2D[1, -2, 4]}
```

#### **Pair of Vertical Lines**

Form:  $Q \equiv Ax^2 + Dx + F = 0, A \neq 0.$ 

If Q takes the form  $Ax^2 + Dx + F = 0$  and  $A \neq 0$  then Q can be factored into two linear terms using the quadratic formula. This yields the two equations

$$x = \frac{-D \pm \sqrt{D^2 - 4AF}}{2A}.$$

If the discriminant of this equation,  $d = D^2 - 4AF$ , is less than zero, then there are no real points in the locus represented by Q. Otherwise, Q represents a pair of vertical lines whose equations are

$$2Ax + (D + \sqrt{d}) = 0$$
 and  $2Ax + (D - \sqrt{d}) = 0$ .

The two lines are coincident if d = 0.

E

**Example.** Show that the equation  $x^2 + x - 6 = 0$  represents two vertical lines.

**Solution.** Use the Descarta2D function Loci2D[quad].

```
In[2]: Clear[x,y];
Loci2D[Quadratic2D[x<sup>2</sup>+x-6==0, {x, y}]]
Out[2] {Line2D[2, 0, -4], Line2D[2, 0, 6]}
```

#### Pair of Horizontal Lines

Form:  $Q \equiv Cy^2 + Ey + F = 0, C \neq 0.$ 

If Q takes the form  $Cy^2 + Ey + F = 0$  and  $C \neq 0$  then Q can be factored into two linear terms using the quadratic formula. This yields the two equations

$$y = \frac{-E \pm \sqrt{E^2 - 4CF}}{2C}$$

If the discriminant of this equation,  $d = E^2 - 4CF$ , is less than zero, then there are no real points in the locus represented by Q. Otherwise, Q represents a pair of horizontal lines whose equations are

$$2Cy + (E + \sqrt{d}) = 0$$
 and  $2Cy + (E - \sqrt{d}) = 0.$ 

The two lines are coincident if d = 0.

**Example.** Show that the equation  $2y^2 - 11y + 12 = 0$  represents two horizontal lines.

**Solution.** Use the Descarta2D function Loci2D[quad].

```
In[3]: Clear[x, y];
Loci2D[Quadratic2D[2y^2 - 11y + 12 == 0, {x, y}]]
Out[3] {Line2D[0, 4, -16], Line2D[0, 4, -6]}
```

#### 

E

#### Intersecting Lines (or a Single Point)

Form:  $Q \equiv Ax^2 + Cy^2 = 0$ ,  $A \neq 0$  and  $C \neq 0$ .

If Q consists of  $x^2$  and  $y^2$  terms only its locus is either a single point or a pair of intersecting lines. If AC > 0 then the locus is the single point (0,0). If A < 0 and C > 0, then the equation factors into the two linear terms

$$\left(\sqrt{-A}x - \sqrt{C}y\right)\left(\sqrt{-A}x + \sqrt{C}y\right) = 0.$$

If A > 0 and C < 0, then the equation factors into the two linear terms given by

$$\left(\sqrt{A}x - \sqrt{-C}y\right)\left(\sqrt{A}x + \sqrt{-C}y\right) = 0.$$

Both of these equations represent a pair of lines that intersect at the origin.

∃

**Example.** Show that the equation  $9x^2 - 4y^2 = 0$  represents a pair of intersecting lines.

**Solution.** Use the Descarta2D function Loci2D[quad].

```
In[4]: Clear[x, y];
Loci2D[Quadratic2D[9x^2 - 4y^2 == 0, {x, y}]]
Out[4] {Line2D[3, -2, 0], Line2D[3, 2, 0]}
```

### Circle

F

Form:  $Q \equiv Ax^2 + Cy^2 + F = 0, A = C, A \neq 0, C \neq 0, F \neq 0.$ 

When the coefficients of the  $x^2$  and  $y^2$  terms of Q are equal and none of the coefficients A, C, or F are equal to zero, the equation has no locus if F > 0; otherwise, when F < 0, the locus is a circle centered at the origin with radius  $\sqrt{-F}$ .

**Example.** Show that the equation  $3x^2 + 3y^2 - 12 = 0$  is the equation of a circle.

Solution. Use the Descarta2D function Loci2D[quad].

```
In[5]: Clear[x, y];
Loci2D[Quadratic2D[3 x^2 + 3 y^2 - 12 == 0, {x, y}]]
Out[5] {Circle2D[{0, 0}, 2]}
```

#### Parabola (Horizontal Axis)

Form:  $Q \equiv Cy^2 + Dx + Ey + F = 0$ ,  $C \neq 0$  and  $D \neq 0$ .

When Q has a  $y^2$  term and an x term, and the  $x^2$  and xy terms are missing, Q represents a parabola whose axis is horizontal. The vertex, (h, k), and the focal length, f, may be determined by completing the square and forming the equation

$$(y-k)^2 = 4f(x-h)$$

where

$$h = \frac{E^2 - 4CF}{4CD}, \ k = -\frac{E}{2C}$$
 and  $f = -\frac{D}{4C}$ 

which is clearly a parabola. The parabola will open to the right if f is positive and it will open to the left if f is negative.

F

н

```
Example. Find and plot the parabola whose equation is y^2 - 8x = 0.
```

**Solution.** Use the Descarta2D function Loci2D[quad].

```
In[7]: Sketch2D[crv, CurveLength2D-> 60];
```



#### Parabola (Vertical Axis)

Form:  $Q \equiv Ax^2 + Dx + Ey + F = 0$ ,  $A \neq 0$  and  $E \neq 0$ .

When Q has an  $x^2$  term and a y term, and the  $y^2$  and xy terms are missing, Q represents a parabola whose axis is vertical. The vertex, (h, k), and the focal length, f, may be determined by completing the square and forming the equation

$$(x-h)^2 = 4f(y-k)$$

where

$$h = -\frac{D}{2A}, \ k = \frac{D^2 - 4AF}{4AE}, \ \text{and} \ f = -\frac{E}{4A}$$

which is clearly a parabola. The parabola will open upward if f is positive and it will open downward if f is negative.

**Example.** Find the parabola whose equation is  $2x^2 - 8x + 4y - 1 = 0$ .

**Solution.** Use the Descarta2D function Loci2D[quad].

```
In [8]: Clear [x, y];

crv = Loci2D[Quadratic2D[2x^2 - 8x + 4y - 1 == 0, {x, y}]]

Out [8] {Parabola2D[{2, \frac{9}{4}}, \frac{1}{2}, \frac{3\pi}{2}]}
```

#### Central Conic (Ellipse or Hyperbola)

Form:  $Q \equiv Ax^2 + Cy^2 + F = 0$ ,  $A \neq 0$ ,  $C \neq 0$ ,  $F \neq 0$ , and  $A \neq C$ .

If Q has non-zero coefficients on the  $x^2$ ,  $y^2$ , and constant terms,  $A \neq C$ , and all the other coefficients are zero, then Q can be written in the form

$$\frac{x^2}{\left(-\frac{F}{A}\right)} + \frac{y^2}{\left(-\frac{F}{C}\right)} = 1.$$

This equation represents an ellipse, a hyperbola or no real locus depending of the values of -F/A and -F/C. The real loci (ellipses and hyperbolas) are centered at the origin (0,0) and have sizes and orientations as shown in the following table:

| Condition  | Locus     | a                     | b                     | $\theta$        |
|--|-----------|-----------------------|-----------------------|-----------------|
| $\left(-\frac{F}{A}\right) < 0 \text{ and } \left(-\frac{F}{C}\right) < 0$ | no locus  | -                     | -                     | -               |
| $\left(-\frac{F}{A}\right) > 0 \text{ and } \left(-\frac{F}{C}\right) < 0$ | hyperbola | $\sqrt{-\frac{F}{A}}$ | $\sqrt{\frac{F}{C}}$  | 0               |
| $\left(-\frac{F}{A}\right) < 0 \text{ and } \left(-\frac{F}{C}\right) > 0$ | hyperbola | $\sqrt{-\frac{F}{C}}$ | $\sqrt{\frac{F}{A}}$  | $\frac{\pi}{2}$ |
| $\left(-\frac{F}{A}\right) > \left(-\frac{F}{C}\right) > 0$                | ellipse   | $\sqrt{-\frac{F}{A}}$ | $\sqrt{-\frac{F}{C}}$ | 0               |
| $\left(-\frac{F}{C}\right) > \left(-\frac{F}{A}\right) > 0$                | ellipse   | $\sqrt{-\frac{F}{A}}$ | $\sqrt{-\frac{F}{C}}$ | $\frac{\pi}{2}$ |

**Example.** Find and plot the conic curve whose equation is  $-x^2 - 4y^2 + 1 = 0$ .

**Solution.** Use the Descarta2D function Loci2D[quad].

F

In[9]: Clear[x, y]; crv = Loci2D[Quadratic2D[-x^2 - 4y^2 + 1 == 0, {x, y}]]

- *Out[9]*  $\{ Ellipse2D[\{0, 0\}, 1, \frac{1}{2}, 0] \}$
- In[10]: Sketch2D[crv];



**Remove the First-Degree Terms** 

Form:  $Q \equiv Ax^2 + Cy^2 + Dx + Ey + F = 0, A \neq 0, C \neq 0, D$  or E non-zero.

If both the  $x^2$  and  $y^2$  terms are present in Q along with at least one of the x or y terms, then Q can be simplified by introducing a change in variables. Specifically, if the substitutions

$$x' = x - \frac{D}{2A}$$
 and  $y' = y - \frac{E}{2C}$ 

are made in Q a new equation

$$Q' \equiv A'x'^2 + C'y'^2 + F' = 0$$

will result where

F

F

$$\begin{array}{rcl}
A' &=& 4A^2C, \\
C' &=& 4AC^2 \ \text{and} \\
F' &=& -CD^2 - AE^2 + 4ACF.
\end{array}$$
(12.1)

Q' is now in a form that can be recognized by inspection. The change in variables translates the origin of the conic. To restore it to its original position we apply the inverse translation to the standard form of the conic.

**Example.** Find the conic whose equation is  $-x^2 + 9y^2 + 4x - 18y - 4 = 0$ . Plot the conic.

-

Solution. Use the Descarta2D function Loci2D[quad].

```
In [11]: Clear [x, y];

crv = Loci2D [Quadratic2D[-x^2 + 9 y^2 + 4 x - 18 y - 4 == 0, {x, y}]]

Out [11] {Hyperbola2D [{2, 1}, 1, 3, \frac{\pi}{2}]}
```

```
In[12]: Sketch2D[crv];
```



#### Eliminate the *xy* Term

Form:  $Q \equiv Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0, B \neq 0.$ 

All quadratic equations with a non-zero xy term coefficient are standard conics in a rotated position. It can be shown that rotating such an equation by the angle  $\theta$ , where

$$\tan\left(2\theta\right) = \frac{B}{C-A}$$

will produce a new quadratic equation, Q', whose x'y' coefficient B' will be zero (see exploration elimxy1.nb).

The coefficient of the xy term can also be removed by making the substitutions

$$x' = kx + y$$
 and  $y' = ky - x$ 

where

$$k = \frac{(C-A)}{B} + \sqrt{\left(\frac{C-A}{B}\right)^2 + 1}$$

(see exploration elimxy2.nb). These substitutions are equivalent to a rotation  $\theta$  where

$$\tan \theta = \frac{1}{k}$$

and a scaling by the factor

$$s = \frac{1}{\sqrt{1+k^2}}.$$

The equation for Q' resulting from the substitution is given by

$$A'x'^{2} + C'y'^{2} + D'x' + E'y' + F' = 0$$

where

E

$$A' = Ak^{2} - Bk + C,$$
  

$$C' = Ck^{2} + Bk + A,$$
  

$$D' = Dk - E,$$
  

$$E' = Ek + D \text{ and}$$
  

$$F' = F$$

as shown in explorations elimxy2.nb and elimxy3.nb. Q' is then a quadratic equation without an xy term that can be recognized by the previously presented techniques. A scaling and rotation is applied to the resulting conic returning it to its original position. This approach is the one implemented in *Descarta2D* since no trigonometric functions are involved in the process, except for the final rotation.

**Example.** Find the conic curve represented by the equation

$$-4x^2 + 10xy - 4y^2 - 12x + 6y - 9 = 0$$

and plot the curve.

Solution. Use the Descarta2D function Loci2D[quad].

```
In [13]: Clear [x, y];

crv = Loci2D [Quadratic2D[-4 x^2 + 10 x * y - 4 y^2 - 12 x + 6 y - 9 == 0, {x, y}]]
```

*Out[13]* {Hyperbola2D[{1, 2}, 3, 1,  $\frac{\pi}{4}$ ]}

In[14]: Sketch2D[crv];



|       | Degenerate Conic, $D = 0$   | Proper Conic, $D \neq 0$        |  |
|-------|-----------------------------|---------------------------------|--|
| K < 0 | two intersecting lines      | hyperbola                       |  |
| K = 0 | J < 0, two parallel lines   | parabola                        |  |
|       | J = 0, two coincident lines |                                 |  |
|       | J > 0, no real locus        |                                 |  |
| K > 0 |                             | ID < 0, circle $(a = b, h = 0)$ |  |
|       | single point                | ID < 0, ellipse                 |  |
|       |                             | ID > 0, no real locus           |  |

Table 12.1: Classification of conics.

## 12.2 Classification of Conics

We may desire to determine the type of a conic from the general equation without computing the defining numerical parameters. This can be accomplished by examining the values of a set of *invariant* expressions. For simplicity of the invariant expressions we choose to write the quadratic equation in the form

$$ax^2 + 2hxy + by^2 + 2gx + 2fy + c = 0$$

where the factor 2 is inserted in the xy, x and y terms. For this form of the equation we define

$$I = a + b, J = ab + ac + bc - f^2 - g^2 - h^2, K = ab - h^2$$

and

$$D = \left| \begin{array}{ccc} a & h & g \\ h & b & f \\ g & f & c \end{array} \right|.$$

Each of the four expressions is invariant under rotation of the coordinate axes; that is, they are equal respectively to the corresponding expressions after a rotation is performed. The invariants are useful in the classification of conics as shown in Table 12.1.

## 12.3 Center Point of a Conic

The center point of a central conic (a circle, ellipse or hyperbola) can be determined directly from its equation. The center point, (h, k), of  $ax^2 + bxy + cy^2 + dx + ey + f = 0$  has a rela-

tively simple form given by

$$h = \frac{2cd - be}{b^2 - 4ac}$$
 and  $k = \frac{2ae - bd}{b^2 - 4ac}$ .

If  $b^2 - 4ac = 0$  then the conic is a parabola and has no center.

**Example.** Find the center point of  $5x^2 - 6xy + 5y^2 - 14x + 2y + 5 = 0$ .

**Solution.** The Descarta2D function Point2D[quad] returns the center point of a central conic.

```
In[15]: Clear[x, y];
Point2D[Quadratic2D[5x^2 - 6x*y+5y^2 - 14x+2y+5 == 0, {x, y}]]
Out[15] Point2D[{2, 1}]
```

F

#### 12.4 Conic from Point, Line and Eccentricity

Conic curves may be defined as the locus of a point that moves so that the ratio of its distance from a fixed point and from a fixed line is a constant. The fixed point is called the *focus*, the fixed line the *directrix* and the constant ratio the *eccentricity*. In previous chapters is has been shown that if the eccentricity, e, is a positive number less than one, then the conic curve is an ellipse, if e = 1 a parabola and if e > 1 the curve is a hyperbola.

Consider a focus point  $F(x_1, y_1)$  and a (normalized) directrix line  $\lambda x + \mu y - \rho = 0$  (where  $\lambda^2 + \mu^2 = 1$ ). The distance,  $d_1$ , from a point P(x, y) on the locus to the focus F is given by

$$d_1 = \sqrt{(x - x_1)^2 + (y - y_1)^2}$$

and the distance,  $d_2$ , from point P to the directrix line is given by

$$d_2 = \pm (\lambda x + \mu y - \rho).$$

By definition, the equation of the conic curve is

$$e = \frac{d_1}{d_2}$$

or

$$\sqrt{(x-x_1)^2 + (y-y_1)^2} = \pm e(\lambda x + \mu y - \rho).$$

Squaring both sides and rearranging yields

$$(e^2\lambda^2 - 1)x^2 + 2e^2\lambda\mu xy + (e^2u^2 - 1)y^2 + 2(x_1 - e^2\lambda\rho)x + 2(y_1 - e^2\mu\rho)y + (e^2\rho^2 - x_1^2 - y_1^2) = 0.$$

This equation is of the form  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$  and is, therefore, a conic curve of the second degree. The equation reveals that if the defining directrix line is parallel to one of the coordinate axes, then B = 0, since either  $\lambda$  or  $\mu$  will be zero and the equation will have no xy term.

**Example.** Find the quadratic equation of the curve whose focus is the point (2, 1), directrix is x - 3y + 3 = 0 and eccentricity is 2. Plot the conic curve.

**Solution.** The Descarta2D function Quadratic2D[point, line, e] returns a quadratic representing the equation of the conic with the given point as a focus, the line as a directrix and the given eccentricity.

In[17]: Sketch2D[{pt1, ln1, Loci2D[q1]}];



## 12.5 Common Vertex Equation

All of the proper conics (circles, ellipses, hyperbolas and parabolas) can be represented by an equation involving the vertex of the conic. The expression 2p in the equation of the parabola  $y^2 = 2px$  is the length of the chord of the parabola perpendicular to the x-axis through the

F

Table 12.2: Parameter of a conic.

| Curve     | PARAMETER     | VERTEX EQUATION        |
|-----------|---------------|------------------------|
| parabola  | 2p            | $y^2 = 2px$            |
| ellipse   | $2p = 2b^2/a$ | $y^2 = 2px - (p/a)x^2$ |
| hyperbola | $2p = 2b^2/a$ | $y^2 = 2px + (p/a)x^2$ |

focus and represents a measure of the *width* of the parabola. The expression 2p is called the *parameter* of the parabola. This definition can be generalized to the other conics: the *parameter* of a conic is defined as the length of the chord perpendicular to the principal axis through the focus. The length of this chord for each conic is shown in Table 12.2 and is quite easy to determine from the standard form of each conic.

Consider the equation of an ellipse centered at the origin in standard position

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1.$$

Transforming the origin to the vertex V(-a, 0) yields the equation

$$\frac{(x-a)^2}{a^2} + \frac{y^2}{b^2} = 1$$

which can be rearranged into  $y^2 = 2b^2x/a - b^2x^2/a^2$ , or, by using the semi-parameter  $p = b^2/a$  of the ellipse, into

$$y^2 = 2px - (p/a)x^2.$$

The relation to the vertex equation of the parabola  $y^2 = 2px$  is obvious. The term  $(p/a)x^2$  is subtracted from the term 2px to obtain the ellipse. This explains the name *ellipse*: it is derived from the Greek term *elleipsis* meaning a *deficiency* compared with a parabola.

Similarly, the equation of a hyperbola

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$$

referred to by its vertex can be shown to be

$$y^2 = 2px + (p/a)x^2$$

where  $p = b^2/a$  is the semi-parameter of the hyperbola. Compared with the parabola  $y^2 = 2px$ , there is a term  $(p/a)x^2$  in *excess* of the term 2px. This explains the name hyperbola from the Greek hyperbole meaning the excess.

By introducing the eccentricity e of the conic, all of the vertex equations can be represented by the *common vertex equation* 

$$(y-k)^{2} = 2p(x-h) - (1-e^{2})(x-h)^{2}$$

where (h, k) is the vertex point of the conic, 2p is the parameter of the conic and e is the eccentricity. The vertex equation also includes the case of a circle by using e = 0 as the eccentricity.

**Example.** Plot the four curves represented by the vertex equation

$$y^{2} = 2(x-1) - (1-e^{2})(x-1)^{2}$$

for the eccentricities  $e = \{0, 3/4, 1, 3/2\}.$ 

**Solution.** The Descarta2D function Loci2D[point, len,  $e, \theta$ ] constructs a conic (circle, ellipse, hyperbola or parabola) from the vertex point, focal chord length, eccentricity and rotation angle.

$$\begin{split} &In[18]: \mbox{ conl = Map[Loci2D[Point2D[{1, 0}], 1, #, 0]&, {0, 3/4, 1, 3/2}] \\ &Out[18] \ \left\{ \left\{ \mbox{Circle2D}\left[ \left\{ \frac{3}{2}, 0 \right\}, \frac{1}{2} \right] \right\}, \ \left\{ \mbox{Ellipse2D}\left[ \left\{ \frac{15}{7}, 0 \right\}, \frac{8}{7}, \frac{2}{\sqrt{7}}, 0 \right] \right\}, \\ & \left\{ \mbox{Parabola2D}\left[ \left\{ 1, 0 \right\}, \frac{1}{4}, 0 \right] \right\}, \ \left\{ \mbox{Hyperbola2D}\left[ \left\{ \frac{3}{5}, 0 \right\}, \frac{2}{5}, \frac{1}{\sqrt{5}}, 0 \right] \right\} \right\} \end{split}$$

In[19]: Sketch2D[con1, PlotRange -> {{1/2, 5}, {-2, 2}}];



**Descarta2D Hint.** The Descarta2D function Quadratic2D[point, len,  $e, \theta$ ] returns a quadratic given the vertex point, focal chord length, eccentricity and rotation angle.

F

F

#### 12.6 Conic Intersections

Intersecting two curves is most easily accomplished if we can obtain parametric equations for one of them and an implicit equation for the other. Specifically, suppose that the first curve has parametric equations x = x(t) and y = y(t) and the second curve has an implicit equation f(x, y) = 0. By substitution these two curves intersect at values of t satisfying f(x(t), y(t)) = 0. Once the values for t are known they can be substituted into the parametric equations to find the (x, y) coordinates of the intersection points.

As a specific application of this technique, suppose we wish to find the intersection points of a line px + qy + r = 0 and a conic curve  $ax^2 + bxy + cy^2 + dx + ey + f = 0$ . We can take either x or y as the parameter of the equation px + qy + r = 0; suppose we select x (assuming  $q \neq 0$ ), yielding the parametric equations

$$x = x$$
 and  $y = -\frac{px+r}{q}$ .

Substituting these values into the equation for the conic curve yields a quadratic equation in the variable x given by

$$ax^{2} + bx\left(-\frac{px+r}{q}\right) + c\left(-\frac{px+r}{q}\right)^{2} + dx + e\left(-\frac{px+r}{q}\right) + f = 0$$

which is easy to solve using the quadratic formula. Once the two values for x are known, the corresponding values for y can be determined using the parametric equations of the line. So, in the general case, a line and a conic will intersect in two points, the points being real and distinct, real and coincident (the line being tangent to the conic) or imaginary (the line does not intersect the conic).

Now consider the case of two intersecting conic curves whose equations are given by

$$a_1x^2 + b_1xy + c_1y^2 + d_1x + e_1y + f_1 = 0$$
  
$$a_2x^2 + b_2xy + c_2y^2 + d_2x + e_2y + f_2 = 0.$$

Depending on the values of the coefficients it may or may not be easy to express one of the equations with a pair of parametric equations; therefore, we look for alternative techniques for finding the points of intersection. The brute force approach to the problem is to simply regard it as a problem of solving two non-linear equations in two unknowns. *Mathematica* can solve such systems of equations both numerically and symbolically, and this is, in fact, the method implemented in *Descarta2D*.

Alternatively, the method of *pencils* can be used. Suppose we have two curves f(x, y) and g(x, y). For any given value of  $\lambda$ , we can form the equation  $f(x, y) + \lambda g(x, y) = 0$  which obviously passes through all the points of intersection of the original two curves. As  $\lambda$  varies, an entire family of curves, called a *pencil*, will be produced. By selecting an appropriate value for  $\lambda$  we can hope to produce an equation  $f(x, y) + \lambda g(x, y) = 0$  that is particularly simple. We can then intersect the simpler curve with one of the original curves. This approach works well for conic curves because there always exists a value for  $\lambda$  such that  $f(x, y) + \lambda g(x, y) = 0$ 

represents two straight lines. Intersecting these two lines with either of the original conics produces the four intersection points (which may be real and distinct, real and coincident or imaginary). So there may be up to four points of intersection between two conic curves. Since there exist three pairs of lines passing through four points, there are three values for  $\lambda$ that represent two lines in the equation of the pencil. Finding the three values for  $\lambda$  involves solving a cubic equation, which appears to be easier than solving two non-linear equations in two unknowns. (Since solving two non-linear equations in two unknowns is equivalent to solving a fourth-degree equation, and solving a fourth-degree equation reduces to solving a cubic equation, the two techniques are mathematically similar in complexity.)

**Example.** Find the points of intersection of the line x - 2y + 2 = 0 with (a) the circle  $x^2 + y^2 = 4$  and (b) the ellipse  $x^2/9 + y^2 = 1$ .

**Solution.** The Descarta2D function Points2D[curve, curve] returns a list of points that are the intersection of the two curves.

```
In[20]: 11 = Line2D[1, -2, 2];
c1 = Circle2D[{0, 0}, 2];
e1 = Ellipse2D[{0, 0}, 3, 1, 0];
pts = {Points2D[11, e1], Points2D[c1, e1]} // N
Out[20] {{Point2D[{-2.76923, -0.384615}], Point2D[{0., 1.}]},
{Point2D[{-1.83712, -0.790569}], Point2D[{-1.83712, 0.790569}],
Point2D[{1.83712, -0.790569}], Point2D[{1.83712, 0.790569}]}}
```

```
In[21]: Sketch2D[{l1, c1, e1, pts}];
```



## 12.7 Explorations

ELIMINATE CROSS-TERM BY ROTATION.....elimxy1.nb Show that rotating a quadratic  $ax^2 + bxy + cy^2 + dx + ey + f = 0$  through an angle  $\theta$  given by

$$\tan(2\theta) = \frac{b}{c-a}$$

will cause the xy term to vanish.

ELIMINATE CROSS-TERM BY CHANGE IN VARIABLES. ..... elimxy2.nb Show that applying the change in variables x' = kx + y and y' = ky - x, where

$$k = \frac{(c-a)}{b} + \sqrt{\left(\frac{c-a}{b}\right)^2 + 1},$$

to the equation  $ax^2 + bxy + cy^2 + dx + ey + f = 0$  will cause the xy term to vanish and a new quadratic with the following coefficients will be formed:

$$\begin{array}{rcl}
a' &=& ak^2 - bk + c, \\
b' &=& 0, \\
c' &=& ck^2 + bk + a, \\
d' &=& dk - e, \\
e' &=& ek + d \text{ and} \\
f' &=& f.
\end{array}$$

ELIMINATE CROSS-TERM BY CHANGE IN VARIABLES. ..... elimxy3.nb Show that applying the change in variables x' = kx + y and y' = ky - x, where

$$k = \frac{(c-a)}{b} + \sqrt{\left(\frac{c-a}{b}\right)^2 + 1},$$

to the equation  $ax^2 + bxy + cy^2 + dx + ey + f = 0$  is equivalent to rotating the quadratic by an angle  $\theta$  given by

$$\tan \theta = \frac{1}{k}$$

and scaling the quadratic by a scale factor

$$s = \frac{1}{\sqrt{1+k^2}}.$$

ELIMINATE LINEAR TERMS.....elimlin.nb

Show that applying the change in variables

$$x' = x - \frac{d}{2a}$$
 and  $y' = y - \frac{e}{2c}$ 

to the quadratic equation  $ax^2 + cy^2 + dx + ey + f = 0$  yields the quadratic

$$ax'^{2} + cy'^{2} - \frac{d^{2}}{4a} - \frac{e^{2}}{4c} + f = 0$$

whose linear terms have vanished.

CENTER OF A QUADRATIC.....center.nb Show that applying the change in variables

$$x = x' + \frac{2cd - be}{b^2 - 4ac}$$
 and  $y = y' + \frac{2ae - bd}{b^2 - 4ac}$ 

to the quadratic  $ax^2 + bxy + cy^2 + dx + ey + f = 0$  causes the linear terms to vanish, implying that the center of the conic is

$$h = \frac{2cd - be}{b^2 - 4ac}, \ k = \frac{2ae - bd}{b^2 - 4ac}.$$

POLAR EQUATION OF A CONIC......polarcon.nb



Let the focus F of a conic be at the pole of a polar coordinate system and the directrix D be perpendicular to the polar axis at a distance  $\rho$  to the left of the pole as shown in the figure. Show that the polar equation of the conic is

$$r = \frac{e\rho}{1 - e\cos\theta}$$

where e is the eccentricity of the conic.

Show that the quadratic  $Q \equiv ax^2 + bxy + cy^2 + dx + ey = 0$ , that passes through the origin, can be parameterized by the equations

$$x(t) = -\frac{d+et}{a+t(b+ct)}$$
 and  $y(t) = -\frac{t(d+et)}{a+t(b+ct)}$ 

where  $-\infty < t < +\infty$ .

## Chapter 13

# **Conic Arcs**

In previous chapters we introduced *line segments* and *circular arcs* which are pieces of more complete curves. In this chapter we introduce a *conic arc* which is a piece of a conic curve. As with circular arcs, conic arcs are useful for constructing smoothly connected sequences of curves as well as pleasing aesthetic shapes.

## 13.1 Definition of a Conic Arc

Let points  $P_0(x_0, y_0)$  and  $P_1(x_1, y_1)$  be the start and end points, respectively, of a segment of a conic curve, Q, and let  $P_A(x_A, y_A)$  be the point of intersection, or *apex*, of the two tangent lines to the curve at  $P_0$  and  $P_1$  as shown in Figure 13.1. Furthermore, let h equal the maximum height of the segment measured from the chord  $P_0P_1$  and k be the distance from  $P_A$  to the chord  $P_0P_1$ . The points  $P_0$ ,  $P_1$ ,  $P_A$  and the ratio  $\rho$ , given by  $\rho = h/k$ , define a *conic arc*. The ratio  $\rho$  is called the *projective discriminant* of the conic arc, and the point at the maximum height on the curve is called the *shoulder point*. The points  $P_0$ ,  $P_1$  and  $P_A$  are called *control points* of the conic arc.



Figure 13.1: Definition of a conic arc.

**Example.** Plot the conic arc with start and end points (-2, 1) and (3, 0), respectively, apex point (1, 2) and projective discriminant  $\rho = 0.45$ .

Solution. Descarta2D represents a conic arc as

ConicArc2D[ $\{x_0, y_0\}, \{x_A, y_A\}, \{x_1, y_1\}, \rho$ ]

where  $(x_0, y_0)$  and  $(x_1, y_1)$  are the coordinates of the start and end points, respectively,  $(x_A, y_A)$  is the apex point and  $\rho$  is the projective discriminant.



There are several functions provided by Descarta2D to query conic arcs. The function Rho2D[cnarc] returns the  $\rho$  value of the conic arc. Point2D[cnarc, Apex2D] returns the apex control point of a conic arc. The coordinates of points on a conic arc at a parameter value t are returned by the function cnarc[t], t = 0 returns the start point coordinates, t = 1 the end point coordinates and t = 1/2 the shoulder point coordinates.

```
In[2]: {Rho2D[c1],
        Point2D[c1, Apex2D],
        Map[c1[#]&, {0, 1/2, 1}]}
Out[2] {0.45, Point2D[{1, 2}], {{-2., 1.}, {0.725, 1.175}, {3., 0}}}
```

## 13.2 Equation of a Conic Arc

The curve underlying the conic arc is clearly a conic curve since there are five conditions imposed on the curve (two points, two tangents and the projective discriminant,  $\rho$ ). The projective discriminant,  $\rho$ , can be interpreted as defining a third line tangent to the curve,

parallel to the line  $P_0P_1$  at a distance h from  $P_0P_1$ , where h is given by  $h = \rho k$  and k is the distance from  $P_A$  to line  $P_0P_1$ .

In a subsequent chapter we will describe a general procedure for finding the quadratic equation of a conic constrained by two points and three tangent lines, and we will show that when the two points are on two of the tangent lines, there is only one quadratic satisfying the conditions. Specifically, the equation is given by

 $\alpha\beta = k(1 - \alpha - \beta)^2$ 

where,

$$k = \frac{(1-\rho)^2}{4\rho^2}$$
  

$$\alpha = \frac{(y-y_A)(x_1-x_A) - (x-x_A)(y_1-y_A)}{(y_0-y_A)(x_1-x_A) - (x_0-x_A)(y_1-y_A)}$$
  

$$\beta = \frac{(y-y_A)(x_0-x_A) - (x-x_A)(y_0-y_A)}{(y_1-y_A)(x_0-x_A) - (x_1-x_A)(y_0-y_A)}.$$

**Example.** Find the quadratic associated with the conic arc with start and end points (0,0) and (3,0), respectively, apex point (1,2) and projective discriminant  $\rho = 1/4$ . Find the conic curve associated with the conic arc.

**Solution.** The function Quadratic2D[*cnarc*] returns the quadratic associated with a conic arc. The function Loci2D[*cnarc*] returns a list containing the conic curve associated with a conic arc.

```
In[3]: cal = ConicArc2D[{0, 0}, {1, 2}, {3, 0}, 1/4];
        {q1 = Quadratic2D[cal] // Simplify,
        c1 = Loci2D[cal] // N}
```

```
Out[3] {Quadratic2D[-16, -8, -73, 48, -24, 0],
{Ellipse2D[{1.5625, -0.25}, 1.60506, 0.743424, 3.07187]}}
```



### 13.3 Projective Discriminant

In this section we will examine the significance of the value of the projective discriminant,  $\rho$ . By definition,  $\rho$  may take on values in the range

 $0 < \rho < 1.$ 

Consider the conic arc, S, with start and end points (0,0) and (1,0), respectively, apex point  $P_A(x_A, y_A)$  and projective discriminant  $\rho$ . Clearly, any arbitrary conic arc can be transformed to coincide with S by applying a proper sequence of translations, rotations and scaling transformations. Such transformations do not change the type of conic curve associated with the conic arc. Using *Mathematica* we can find the quadratic equation underlying S as shown by the following commands.

```
In[5]: Clear[xA, yA, p];
S = ConicArc2D[{0, 0}, {xA, yA}, {1, 0}, p];
Q = Quadratic2D[S] // Simplify
Out[5] Quadratic2D[-4 p<sup>2</sup> yA<sup>2</sup>, 4 p<sup>2</sup> (-1 + 2 xA) yA, -1 + 2 p - p<sup>2</sup> (1 - 2 xA)<sup>2</sup>, 4 p<sup>2</sup> yA<sup>2</sup>,
-4 p<sup>2</sup> xA yA, 0]
```

As has already been shown in a previous chapter, the specific conic type of the quadratic equation

$$ax^{2} + bxy + cy^{2} + dx + ey + f = 0$$

is determined by the discriminant,  $D = b^2 - 4ac$ . For an ellipse D < 0, for a parabola D = 0, and for a hyperbola D > 0. For the quadratic, Q, representing the conic arc S defined above,  $D = 16\rho^2(-1+2\rho)y_1^2$ . It is clear by inspection, that if  $0 < \rho < 1/2$  the conic is an ellipse; if  $\rho = 1/2$  the conic is a parabola; and for  $1/2 < \rho < 1$  the conic is a hyperbola.

### 13.4 Conic Characteristics

In Section 13.2 we showed that the quadratic equation associated with a conic arc is given by

$$\alpha\beta = k(1 - \alpha - \beta)^2$$

where,

$$k = \frac{(1-\rho)^2}{4\rho^2}$$

$$\alpha = \frac{(y-y_A)(x_1-x_A) - (x-x_A)(y_1-y_A)}{(y_0-y_A)(x_1-x_A) - (x_0-x_A)(y_1-y_A)}$$

$$\beta = \frac{(y-y_A)(x_0-x_A) - (x-x_A)(y_0-y_A)}{(y_1-y_A)(x_0-x_A) - (x_1-x_A)(y_0-y_A)}$$

Therefore, since we know its quadratic equation, all the geometric characteristics of the conic curve associated with the conic arc can be expressed in terms of the defining elements of the

conic arc,  $P_0(x_0, y_0)$ ,  $P_A(x_A, y_A)$ ,  $P_1(x_1, y_1)$  and  $\rho$ . Of particular interest is the formula for the center of a central conic (circle, ellipse or hyperbola), since this formula is used in the next section to convert a conic into a conic arc. The center point (H, K) is given by

$$H = \frac{-\rho^2 x_A + (\rho - 1)^2 x_M}{1 - 2\rho}$$

$$K = \frac{-\rho^2 y_A + (\rho - 1)^2 y_M}{1 - 2\rho}$$
(13.1)

where  $P_M(x_M, y_M)$  is the midpoint of the conic arc's chord and has coordinates

 $x_M = (x_0 + x_1)/2$  and  $y_M = (y_0 + y_1)/2$ .

This formula is derived in the exploration cacenter.nb.

**Example.** Find the center of the conic arc with control points (0,0), (2,1) and (3,0) and  $\rho = 1/4$ .

**Solution.** The Descarta2D function Point2D[cnarc] returns the center point of a conic arc (the underlying conic cannot be a parabola).

Let Q be a conic and L be a line that intersects Q in two distinct points. We wish to determine the conic arc, S cut by L through Q. Clearly, the intersection points of the line Lwith Q are the start and end points of S. Also, the line passing through the intersection points is the polar (line) of the apex point,  $P_A$ , of S. To complete the definition of the conic arc, we need to determine  $\rho$ . If the conic is a parabola, then  $\rho = 1/2$ ; otherwise, we can assume that the conic is a central conic. Assume the center of the conic is (h, k). Then, using the formula for the x-coordinate of the center of a conic arc given in Equation (13.1), we solve to find the value of  $\rho$  to be

$$\rho = \frac{1}{1 \pm \sqrt{(h - x_A)/(h - x_M)}}$$

where  $P_M(x_M, y_M)$  is the midpoint of  $P_0P_1$ . We choose the plus sign in the denominator because  $\rho$  has to be less than one and the radical produces a positive number. In certain configurations, this formula will be indeterminate and we instead use the *y*-coordinate of the center of the conic arc yielding

$$\rho = \frac{1}{1 \pm \sqrt{(k - y_A)/(k - y_M)}}$$

again choosing the plus sign in the denominator.

7

**Example.** Find the conic arc cut by the line 2x - 4y = 0 through the ellipse

$$\frac{(x-1)^2}{9} + \frac{(y+1)^2}{4} = 1.$$

Plot the original curves and the conic arc separately.

**Solution.** The Descarta2D function ConicArc2D[*line*, *conic*] returns a conic arc defined by a line cutting a conic curve.

```
In[7]: 11 = Line2D[2, -4, 0];
e1 = Ellipse2D[{1, -1}, 3, 2, 0];
ca1 = ConicArc2D[11, e1]
Out[7] ConicArc2D[{-2, -1}, {-2, 5/3}, {46/25, 23/25}, 3/8]
In[8]: Map[Sketch2D[{#}, PlotRange -> {{-5, 5}, {-3, 3}}]&,
{{11, e1}, ca1}];
```



## 13.5 Parametric Equations

The conic arc defined in this chapter is a special case of a more general curve called a *rational quadratic Bézier*. The parametric equations of this simplified formulation are given by

$$x = \frac{b_0(1-\rho)x_0 + b_1\rho x_A + b_2(1-\rho)x_1}{b_0(1-\rho) + b_1\rho + b_2(1-\rho)}$$
$$y = \frac{b_0(1-\rho)y_0 + b_1\rho y_A + b_2(1-\rho)y_1}{b_0(1-\rho) + b_1\rho + b_2(1-\rho)}$$

F

\_\_\_\_\_
F

F

where  $\rho = h/k$  is the projective discriminant and

$$b_0 = (1-t)^2$$
  
 $b_1 = 2t(1-t)$   
 $b_2 = t^2$ .

It is clear from direct substitution that  $P_0$  is the point whose coordinates correspond to t = 0, and  $P_1$  corresponds to t = 1. The point where the curve intersects the line through the midpoint of  $P_0P_1$  and  $P_A$  is called the *shoulder* point of the conic arc. The shoulder point corresponds to the parameter value t = 1/2.

**Example.** Plot nine points at equal parameter values on the conic arc with (-2, 1) and (1, 2) as start and end point, (0, 3) as the apex point and  $\rho = 0.45$ .

**Solution.** The Descarta2D function cnarc[t] returns the coordinates of a point on a conic arc at a parameter t.

```
In[9]: cal = ConicArc2D[{-2, 1}, {0, 3}, {1, 2}, 0.45];
Sketch2D[{cal, Map[Point2D[cal[#]]&, Range[0, 8] / 8]}];
```



# 13.6 Explorations

CIRCULAR CONIC ARC.....cacircle.nb Show that the conic arc with control points (0,0), (a,b) and (2a,0) will be a circular arc if

$$\rho = \frac{a(-a + \sqrt{a^2 + b^2})}{b^2}.$$

CENTER OF A CONIC ARC.....cacenter.nb

Show that the center point (H, K) of a conic arc whose control points are  $P_0(x_0, y_0)$ ,  $P_A(x_A, y_A)$  and  $P_1(x_1, y_1)$  and projective discriminant  $\rho$  is

$$H = \frac{-\rho^2 x_A + (\rho - 1)^2 x_M}{1 - 2\rho}$$
$$K = \frac{-\rho^2 y_A + (\rho - 1)^2 y_M}{1 - 2\rho}$$

where  $P_M(x_M, y_M)$  is the midpoint of the conic arc's chord and has coordinates

$$x_M = \frac{x_0 + x_1}{2}$$
 and  $y_M = \frac{y_0 + y_1}{2}$ 

TANGENT LINE AT SHOULDER POINT.....catnln.nb

Let P be the point at parameter value t = 1/2 on a unit conic arc, C, whose control points are  $P_0(0,0)$ ,  $P_A(a,b)$  and  $P_1(1,0)$  and whose projective discriminant is  $\rho$ . Let L be the line tangent to C at t. Show that L is parallel to the chord  $P_0P_1$  at a distance  $b\rho$  from  $P_0P_1$ . The point P is called the *shoulder point* of the conic arc.

$$(x_M + \rho(x_A - x_M), y_M + \rho(y_A - y_M))$$

where  $P_M(x_M, y_M)$  is the midpoint of the conic arc's chord and has coordinates

$$x_M = \frac{x_0 + x_1}{2}$$
 and  $y_M = \frac{y_0 + y_1}{2}$ 

 $P_0P_AP_1$  such that  $|PP_M|/|P_AP_M| = \rho (P_M(x_M, y_M))$  is the midpoint of  $P_0P_1$ ). Show that P is coincident with the shoulder point of C, having coordinates

$$(x_M + \rho(x_A - x_M), y_M + \rho(y_A - y_M))$$
.

PARAMETRIC EQUATIONS OF A CONIC ARC.....caparam.nb

SHOULDER POINT ON MEDIAN..... camedian.nb Let C be a conic arc with control points  $P_0(x_0, y_0)$ ,  $P_A(x_A, y_A)$  and  $P_1(x_1, y_1)$  and projective discriminant  $\rho$ . Let P be the point on the median  $P_A P_M$  associated with vertex  $P_A$  of triangle

Show that the parametric equations of a unit conic arc represent the same implicit quadratic equation as the one underlying the conic as derived from the control points  $P_0(0,0)$ ,  $P_A(a,b)$  and  $P_1(1,0)$  and  $\rho$ .

# Chapter 14

# **Medial Curves**

A *medial curve* is the locus of points equidistant from two loci of points. In this chapter we will derive the equations of medial curves that are equidistant from two points, a point and a curve (line or circle) and two curves (lines or circles).

### 14.1 Point-Point

E

F

Consider two distinct points  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$  and a point P(x, y). The distance,  $d_1$ , from P to  $P_1$  is given by

$$d_1 = \sqrt{(x - x_1)^2 + (y - y_1)^2}.$$

Likewise, the distance,  $d_2$ , from P to  $P_2$  is given by

$$d_2 = \sqrt{(x - x_2)^2 + (y - y_2)^2}.$$

If point P is on the medial curve defined by  $P_1$  and  $P_2$ , then  $d_1 = d_2$  and

$$\sqrt{(x-x_1)^2 + (y-y_1)^2} = \sqrt{(x-x_2)^2 + (y-y_2)^2}.$$

Squaring both sides of this equation and rearranging yields

$$2(x_2 - x_1)x + 2(y_2 - y_1)y + (x_1^2 + y_1^2) - (x_2^2 + y_2^2) = 0$$

which is easily recognized in this form as the general equation of a line. The medial line is the perpendicular bisector of the line segment joining  $P_1$  and  $P_2$ . The derivation is provided in the exploration mdptpt.nb.

**Example.** Find the equation of the medial line determined by the two points (1,2) and (-1,-1). Plot the points and the medial line.

**Solution.** The function MedialLoci2D[{*point*, *point*}] returns a list of one line that is the medial line determined by the two points.

```
Out[1] {Line2D[-4, -6, 3]}
```

In[2]: Sketch2D[{p1, p2, 112}];



**Descarta2D Hint.** The function Point2D[point, point, Perpendicular2D] returns the perpendicular bisector of the line segment joining two points. This function may also be used.

# 14.2 Point-Line

Consider the point  $P_1(x_1, y_1)$  and the line  $L_2 \equiv A_2 x + B_2 y + C_2 = 0$ , where  $A_2^2 + B_2^2 = 1$  (to simplify the derivation, the coefficients of the line are normalized because distance is involved). The distance,  $d_1$ , from a point P(x, y) to  $P_1$  is given by

$$d_1 = \sqrt{(x - x_1)^2 + (y - y_1)^2}.$$

The distance,  $d_2$ , from a point P(x, y) to the normalized line  $L_2$  is given by

$$d_2 = \pm (A_2 x + B_2 y + C_2)$$

Since P is the locus of points on the medial curve,  $d_1 = d_2$ , and by squaring and rearranging we obtain the quadratic equation

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0.$$

F

where

$$A = B_2^2,$$
  

$$B = -2A_2B_2,$$
  

$$C = A_2^2,$$
  

$$D = -2(x_1 + A_2C_2),$$
  

$$E = -2(y_1 + B_2C_2) \text{ and }$$
  

$$F = x_1^2 + y_1^2 - C_2^2.$$

These equations are derived in the exploration mdptln.nb.

The definition of a parabola is the locus of points equidistant from a point and a line, so it is obvious that in the general case the medial curve of a point and a line will be a parabola.

**Example.** Find the medial curve of the point (-1, -1) and -x - y + 1 = 0 and plot.

**Solution.** The function MedialLoci2D[{*point, line*}] returns a list of one curve that is the medial curve of the point and the line.

 $In[3]: \text{ crvl = MedialLoci2D[{pl = Point2D[{-1, -1}], 12 = Line2D[-1, -1, 1]}]}$  $Out[3] \ \left\{ Parabola2D\left[ \left\{ -\frac{1}{4}, -\frac{1}{4} \right\}, \frac{3}{2\sqrt{2}}, \frac{5\pi}{4} \right] \right\}$ 

In[4]: Sketch2D[{p1, l2, crv1}];



If the point  $P_1$  is on line  $L_2$ , then the medial curve will be a line perpendicular to the defining line.

**Example.** Find the medial curve of the point (1,0) and the line -x - y + 1 = 0 and plot. Notice that the point is on the line.

**Solution.** The same function, MedialLoci2D[{*point, line*}], introduced in the previous example will return a list containing the medial curve, which is a line in this case.

```
In[6]: Sketch2D[{p1, l2, crvl}];
```



# 14.3 Point-Circle

Consider a point  $P_1(x_1, y_1)$  and a circle  $C_2$  with center  $(h_2, k_2)$  and radius  $r_2$ . The distance,  $d_1$ , from a point P(x, y) to  $P_1$  is given by

$$d_1 = \sqrt{(x - x_1)^2 + (y - y_1)^2}.$$

The distance,  $d_2$ , from a point P(x, y) to the circle  $C_2$  is given by

$$d_2 = \sqrt{(x - h_2)^2 + (y - k_2)^2} - r_2$$

when P is outside of circle  $C_2$ . When P is inside  $C_2$  the distance,  $d_2$ , is given by

$$d_2 = r_2 - \sqrt{(x - h_2)^2 + (y - k_2)^2}.$$

F

If P is the locus of points equidistant from  $P_1$  and  $C_2$ , then  $d_1 = d_2$ . Squaring both sides of this equation eliminates the distinction between points P inside the circle and outside the circle. Rearranging the resulting equation yields the quadratic equation

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

where

$$\begin{array}{rcl} A &=& 4((x_1-h_2)^2-r_2^2),\\ B &=& 8(x_1-h_2)(y_1-k_2),\\ C &=& 4((y_1-k_2)^2-r_2^2),\\ D &=& 4(R(x_1-h_2)+2r_2^2x_1),\\ E &=& 4(R(y_1-k_2)+2r_2^2y_1),\\ F &=& R^2-4r_2^2(x_1^2+y_1^2) \ \text{and}\\ R &=& (h_2^2+k_2^2)-(x_1^2+y_1^2)-r_2^2. \end{array}$$

This derivation is included in the exploration mdptcir.nb.

If the point  $P_1$  is outside circle  $C_2$ , the medial curve will be a hyperbola. If  $P_1$  is inside  $C_2$ , the medial curve will be an ellipse. In the special case that  $P_1$  is on  $C_2$ , the medial curve will be a line containing the center point of  $C_2$ . If  $P_1$  is coincident with the center of  $C_2$ , then the medial curve will be a circle centered at  $P_1$  with a radius of  $r_2/2$ .

**Example.** Find the medial curves of four points (-8, 1), (-4, 1), (-2, 1) and (0, 1) with the circle  $x^2 + (y - 1)^2 = 4$ . Plot each of the curves separately.

**Solution.** The Descarta2D function MedialLoci2D[{point, circle}] returns a list of one object equidistant from the point and the circle.

In[8]: Map[Sketch2D[{pts[[#]], c2, crvs[[#]]}]&, {1, 2, 3, 4}];

٦



F

F

# 14.4 Line–Line

The locus of points equidistant from two lines

$$L_1 \equiv A_1 x + B_1 y + C_1 = 0$$
 and  
 $L_2 \equiv A_2 x + B_2 y + C_2 = 0$ 

are the two angle bisector lines. The equations of these two lines are

$$\frac{A_1x + B_1y + C_1}{\sqrt{A_1^2 + B_1^2}} = \pm \frac{A_2x + B_2y + C_2}{\sqrt{A_2^2 + B_2^2}}$$

as shown in the exploration mdlnln.nb.

**Example.** Find the medial lines for 3x - 4y + 1 = 0 and 2x + 2y - 3 = 0 and plot.

**Solution.** The function MedialLoci2D[{*line*, *line*}] returns a list of lines that are the medial lines of the two given lines. If the lines are parallel, then the list will contain one line; otherwise, it will contain two lines.

```
\begin{array}{l} \textit{Out[9]} & \left\{ \texttt{Line2D} \left[ -10 + 6 \ \sqrt{2} \ , \ -10 - 8 \ \sqrt{2} \ , \ 15 + 2 \ \sqrt{2} \ \right] , \\ & \texttt{Line2D} \left[ 10 + 6 \ \sqrt{2} \ , \ 10 - 8 \ \sqrt{2} \ , \ -15 + 2 \ \sqrt{2} \ \right] \right\} \end{array}
```

In[10]: Sketch2D[{11, 12, lns}];



# 14.5 Line–Circle

Consider a line  $L_1 \equiv A_1x + B_1y + C_1 = 0$ , where  $A_1^2 + B_1^2 = 1$  (to simplify the derivation, the coefficients of the line are normalized because distance is involved), and a circle  $C_2$  with center at  $(h_2, k_2)$  and radius  $r_2$ . The distance,  $d_1$ , from a point P(x, y) to line  $L_1$  is given by

$$d_1 = \pm (A_1 x + B_1 y + C_1).$$

The distance,  $d_2$ , from point P(x, y) to circle  $C_2$  is given by

$$d_2 = \sqrt{(x - h_2)^2 + (y - k_2)^2} - r_2$$

when P is outside of circle  $C_2$ . When P is inside  $C_2$  the distance,  $d_2$ , is given by

$$d_2 = -\sqrt{(x-h_2)^2 + (y-k_2)^2} + r_2.$$

We introduce a sign constant, s, which takes on the values  $\pm 1$ , so that we can combine the two equations for  $d_2$  yielding

$$d_2 = s \left( \sqrt{(x - h_2)^2 + (y - k_2)^2} - r_2 \right).$$

If P is the locus of points equidistant from  $P_1$  and  $C_2$ , then  $d_1 = d_2$ . Rearranging the resulting equation yields the quadratic equation

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

where

F

F

$$A = B_{1}^{2},$$
  

$$B = -2A_{1}B_{1},$$
  

$$C = A_{1}^{2},$$
  

$$D = -2(h_{2} + A_{1}(C_{1} + sr_{2})),$$
  

$$E = -2(k_{2} + B_{1}(C_{1} + sr_{2})) \text{ and}$$
  

$$F = h_{2}^{2} + k_{2}^{2} - r_{2}^{2} - C_{1}(C_{1} + 2sr_{2})$$

This derivation is included in mdlncir.nb. If the line intersects the circle in two distinct points, then the medial curves will be two parabolas, each passing through the points of intersection of the line and the circle.

**Example.** Find the curves that are equidistant from the line y = 1 and the circle  $x^2 + (y - 1)^2 = 4$ . Plot the curves.

**Solution.** The function MedialLoci2D[{*line*, *circle*}] returns a list of curves equidistant from a line and a circle.

```
In[11]: l1 = Line2D[0, 1, -1];
c2 = Circle2D[{0, 1}, 2];
crvs = MedialLoci2D[{11, c2}]
```

Out[11] {Parabola2D[{0,0},1, $\frac{\pi}{2}$ ], Parabola2D[{0,2},1, $\frac{3\pi}{2}$ ]}

```
In[12]: Sketch2D[{11, c2, crvs}];
```



If the line is tangent to the circle then one of the medial curves will be a parabola, and the other will be a line passing through the tangency point and the center point of the circle. Strictly speaking, not all of the points on the line are equidistant from the line and the circle, unless we consider the distance to be measured both from the closest point on the circle *and* the farthest point on the circle.

**Example.** Find the curves that are equidistant from the line y = 3 and the circle  $x^2 + (y-1)^2 = 4$  and plot. Notice that the line is tangent to the circle.

**Solution.** Use the function MedialLoci2D[{*line*, *circle*}] introduced in the previous example.

```
In[13]: 11 = Line2D[0, 1, -3];
c2 = Circle2D[{0, 1}, 2];
crvs = MedialLoci2D[{11, c2}]
Out[13] {Line2D[2, 0, 0], Parabola2D[{0, 3}, 2, 3π/2]}
```

In[14]: Sketch2D[{l1, c2, crvs}];



If the line and the circle do not intersect, then the two medial curves will be parabolas. Strictly speaking, only one of these parabolas is equidistant from the circle and the line, unless we consider the distance to be measured both from the closest point on the circle *and* the farthest point on the circle.

**Example.** Find the curves that are equidistant from the line y = 5 and the circle  $x^2 + (y - 1)^2 = 4$ . Plot the curves.

**Solution.** Use the function MedialLoci2D[{*line*, *circle*}] as described in the previous examples.

=

# 14.6 Circle–Circle

Consider two distinct circles

$$C_1 \equiv (x - h_1)^2 + (y - k_1)^2 = r_1^2$$
 and  $C_2 \equiv (x - h_2)^2 + (y - k_2)^2 = r_2^2$ 

Using the same distance equating techniques outlined in previous sections, and introducing a sign constant  $s = \pm 1$ , we can obtain the quadratic equation of the curves equidistant from the two circles

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

where

$$A = 4((h_1 - h_2)^2 - R),$$
  

$$B = 8(h_1 - h_2)(k_1 - k_2),$$
  

$$C = 4((k_1 - k_2)^2 - R),$$
  

$$D = 4(h_1(-D_1 + D_2 + R) + h_2(D_1 - D_2 + R)),$$
  

$$E = 4(k_1(-D_1 + D_2 + R) + k_2(D_1 - D_2 + R)) \text{ and }$$
  

$$F = (D_1 - D_2)^2 - 2(D_1 + D_2)R + R^2$$

and

$$R = (r_1 - sr_2)^2,$$
  

$$D_1 = h_1^2 + k_1^2,$$
  

$$D_2 = h_2^2 + k_2^2 \text{ and }$$
  

$$s = \pm 1.$$

| Configuration, $C_1/C_2$          | $r_1 \neq r_2$    | $r_1 = r_2$    |
|-----------------------------------|-------------------|----------------|
| externally disjoint               | two hyperbolas    | line/hyperbola |
| externally tangent                | line/ellipse      | line/line      |
| intersecting $(2 \text{ points})$ | ellipse/hyperbola | line/ellipse   |
| internally tangent                | line/ellipse      | (impossible)   |
| internally disjoint               | ellipse/ellipse   | (impossible)   |
| concentric                        | circle/circle     | (all points)   |

Table 14.1: Medial curves for two circles.

This derivation is included in the exploration mdcircir.nb. Table 14.1 summarizes the medial curves associated with a pair of circles in several configurations taking into consideration differing radii and equal radii. Strictly speaking, some of the branches of these curves are not equidistant from the two circles, unless we consider the distance to be measured both from the closest *and* the farthest point on the circles.

**Example.** Find and plot the curves equidistant from the two circles  $x^2 + y^2 = 9$ and  $x^2 + (y-2)^2 = 4$ .

Solution. Use the function MedialLoci2D[{circle, circle}].

```
In[17]: c1 = Circle2D[\{0, 0\}, 3];
c2 = Circle2D[\{0, 2\}, 2];
crvs = MedialLoci2D[\{c1, c2\}]
Out[17] \{Ellipse2D[\{0, 1\}, \frac{5}{2}, \frac{\sqrt{21}}{2}, \frac{\pi}{2}], Hyperbola2D[\{0, 1\}, \frac{1}{2}, \frac{\sqrt{3}}{2}, \frac{\pi}{2}]\}
In[18]: Sketch2D[\{c1, c2, crvs\}];
4[
```



F

F

**Descarta2D Hint.** The function MedialLoci2D[{ $obj_1$ ,  $obj_2$ }] produces the same result as MedialLoci2D[{ $obj_2$ ,  $obj_1$ }], that is, the objects may be provided in any order in the list. In addition, MedialEquations2D[{ $obj_1$ ,  $obj_2$ }] will return a list of lines and/or quadratics representing the medial curves.

# 14.7 Explorations

MEDIAL CURVE, POINT-POINT.....mdptpt.nb Show that the line  $2(x_2 - x_1)x + 2(y_2 - y_1)y + (x_1^2 + y_1^2) - (x_2^2 + y_2^2) = 0$  is equidistant from the points  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$ .

MEDIAL CURVE, POINT-LINE......mdptln.nb Show that the quadratic equation

$$Ax^{2} + Bxy + Cy^{2} + Dx + Ey + F = 0.$$

where

$$A = B_2^2,$$
  

$$B = -2A_2B_2,$$
  

$$C = A_2^2,$$
  

$$D = -2(x_1 + A_2C_2),$$
  

$$E = -2(y_1 + B_2C_2) \text{ and }$$
  

$$F = x_1^2 + y_1^2 - C_2^2$$

is equidistant from the point  $P_1(x_1, y_1)$  and the line  $L \equiv A_2x + B_2y + C_2 = 0$ , assuming that L is normalized  $(A_2^2 + B_2^2 = 1)$ .

MEDIAL CURVE, POINT-CIRCLE.....mdptcir.nb Show that the quadratic equation

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

where

$$\begin{array}{rcl} A &=& 4((x_1-h_2)^2-r_2^2),\\ B &=& 8(x_1-h_2)(y_1-k_2),\\ C &=& 4((y_1-k_2)^2-r_2^2),\\ D &=& 4(R(x_1-h_2)+2r_2^2x_1),\\ E &=& 4(R(y_1-k_2)+2r_2^2y_1),\\ F &=& R^2-4r_2^2(x_1^2+y_1^2) \ \text{and}\\ R &=& (h_2^2+k_2^2)-(x_1^2+y_1^2)-r_2^2 \end{array}$$

is equidistant from the point  $P_1(x_1, y_1)$  and the circle

$$(x - h_2)^2 + (y - k_2)^2 = r_2^2.$$

MEDIAL CURVE, LINE-LINE.....mdlnln.nb

Show that the pair of lines whose equations are

$$\frac{A_1x + B_1y + C_1}{\sqrt{A_1^2 + B_1^2}} = \pm \frac{A_2x + B_2y + C_2}{\sqrt{A_2^2 + B_2^2}}$$

is equidistant from the two lines  $A_1x + B_1y + C_1 = 0$  and  $A_2x + B_2y + C_2 = 0$ .

- 0

MEDIAL CURVE, LINE-CIRCLE.....mdlncir.nb Show that the two quadratics whose equations are given by

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

where

$$\begin{array}{rcl} A & = & B_1^2, \\ B & = & -2A_1B_1, \\ C & = & A_1^2, \\ D & = & -2(h_2 + A_1(C_1 + sr_2)), \\ E & = & -2(k_2 + B_1(C_1 + sr_2)), \\ F & = & h_2^2 + k_2^2 - r_2^2 - C_1(C_1 + 2sr_2) \ \text{and} \\ s & = & \pm 1 \end{array}$$

are equidistant from the line

$$A_1x + B_1y + C_1 = 0$$

and the circle

$$(x - h_2)^2 + (y - k_2)^2 = r_2^2,$$

assuming  $A_1^2 + B_1^2 = 1$ .

MEDIAL CURVE, CIRCLE-CIRCLE.....mdcircir.nb Show that the two quadratics whose equations are given by

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

where

$$A = 4((h_1 - h_2)^2 - R),$$
  

$$B = 8(h_1 - h_2)(k_1 - k_2),$$
  

$$C = 4((k_1 - k_2)^2 - R),$$
  

$$D = 4(h_1(-D_1 + D_2 + R) + h_2(D_1 - D_2 + R)),$$
  

$$E = 4(k_1(-D_1 + D_2 + R) + k_2(D_1 - D_2 + R)) \text{ and }$$
  

$$F = (D_1 - D_2)^2 - 2(D_1 + D_2)R + R^2$$

and

$$R = (r_1 - sr_2)^2,$$
  

$$D_1 = h_1^2 + k_1^2,$$
  

$$D_2 = h_2^2 + k_2^2 \text{ and }$$
  

$$s = \pm 1$$

are equidistant from the two circles

$$(x-h_1)^2 + (y-k_1)^2 = r_1^2$$
 and  $(x-h_2)^2 + (y-k_2)^2 = r_2^2$ .

MEDIAL CURVE TYPE.....mdtype.nb

Show that the medial curve equidistant from a point and a circle is a hyperbola when the point is *outside* the circle and it is an ellipse when the point is *inside* the circle. (Hint: Examine the value of the discriminant  $B^2 - 4AC$  of the medial quadratic.) Part IV Geometric Functions

# Chapter 15

# Transformations

A transformation is a mathematical operation that changes a function of variables, say f(x, y), into a new function f'(x', y') where

$$x' = f_1(x, y)$$
 and  $y' = f_2(x, y)$ .

These equations are called the *equations of the transformation*. Transformations can often be constructed so that f' is much simpler than f. In this chapter we will study four transformations that have useful geometric interpretations: *translation*, *rotation*, *scaling* and *reflection*.

### 15.1 Translations

F

A translation is a transformation that maps coordinates (x, y) into

(x+u, y+v).

When a translation is applied to a locus of points, the resulting locus has the same shape and orientation as the original one, but its position with respect to the coordinate axes is offset by distances u in the x-direction and v in the y-direction. The equations of the transformation are

x' = x + u and y' = y + v.

**Example.** Determine the coordinates of the point that results from translating (3, 2) by u = -1 and v = -3.

**Solution.** The function Translate2D[ $\{x, y\}$ ,  $\{u, v\}$ ] translates a coordinate list (x, y) by the specified offset (u, v), returning a new coordinate list. The function Translate2D[*point*,  $\{u, v\}$ ] performs the same translation and returns a translated point.

```
In[1]: {Translate2D[{3, 2}, {-1, -3}],
Translate2D[Point2D[{3, 2}], {-1, -3}]}
Out[1] {{2, -1}, Point2D[{2, -1}]}
```

A translation can also be applied to an equation. For example, if

$$f(x,y) = Ax + By + C$$

is a linear equation in two variables, we can translate this by making the substitutions x = x' - u and y = y' - v. Mathematica provides powerful functions for performing these transformations.

In standard mathematical notation the translated equation is

$$Ax + By - Au - Bv + C.$$

Mathematica Hint. The Mathematica function Replace, represented by the /. operator, applies a set of replacement rules to an expression.

In a similar manner a quadratic equation can be translated. Again *Mathematica* provides a convenient means for performing the algebraic operations.

Collecting terms and writing in standard mathematical notation yields the translated quadratic equation

$$A'x^{2} + B'xy + C'y^{2} + D'x + E'y + F' = 0$$

where

$$A' = A$$
  

$$B' = B$$
  

$$C' = C$$
  

$$D' = D - 2Au - Bv$$
  

$$E' = E - 2Cv - Bu$$
  

$$F' = Au^{2} + Buv + Cv^{2} - Du - Ev + F.$$

F

F

Using these basic formulas for translations it is easy to translate other objects. The location of curves, such as circles, ellipses and conic arcs, are defined by points and can be translated by translating the points themselves. For example, Ellipse2D[ $\{h, k\}$ ,  $a, b, \theta$ ] is translated to Ellipse2D[ $\{h + u, k + v\}$ ,  $a, b, \theta$ ].

**Example.** Translate the ellipse

$$\frac{(x-1)^2}{16} + \frac{(y+3)^2}{9} = 1$$

by the offsets u = 2 and v = -2. Plot both the original ellipse and the translated ellipse.

**Solution.** The function Translate2D[*object*,  $\{u, v\}$ ] translates an object u in the x-direction and v in the y-direction. The object may be a coordinate list, a geometric object or a list of Descarta2D objects.

```
In[4]: e2 = Translate2D[e1 = Ellipse2D[{1, -3}, 4, 3, 0], {2, -2}]
Out[4] Ellipse2D[{3, -5}, 4, 3, 0]
```

```
In[5]: Sketch2D[{e1, e2}];
```



# 15.2 Rotations

A rotation by an angle  $\theta$  about the origin is a transformation that maps coordinates (x, y) into  $(x \cos \theta - y \sin \theta, y \cos \theta + x \sin \theta)$ . The mapping is easily confirmed using trigonometry as shown in Figure 15.1.

$$\cos \alpha = x/r$$
$$\sin \alpha = y/r$$



Figure 15.1: Rotation transformation.

$$\begin{aligned} x' &= OA' \\ &= r\cos(\alpha + \theta) \\ &= r(\cos\alpha\cos\theta - \sin\alpha\sin\theta) \\ &= r((x/r)\cos\theta - (y/r)\sin\theta) \\ &= x\cos\theta - y\sin\theta. \end{aligned}$$

Similarly, it can be shown that  $y' = y \cos \theta + x \sin \theta$ .

In order to rotate about a point (h, k), we first translate the coordinates to the origin, perform the rotation using the equations above, then apply the inverse translation to restore the object to its original position with the rotation applied. The general equations of a rotation so derived are

$$\begin{aligned} x' &= h + (x - h)\cos\theta - (y - k)\sin\theta \\ y' &= k + (x - h)\sin\theta + (y - k)\cos\theta. \end{aligned}$$

In order to rotate a linear equation Ax + By + C = 0 we need to solve these equations for x and y so that we can substitute these values into the equation. Solving for x and y in terms of x' and y' (and making use of the identity  $\sin^2 \theta + \cos^2 \theta = 1$ ) yields the equations

$$x = h + (x' - h)\cos\theta + (y' - k)\sin\theta$$
  

$$y = k - (x' - h)\sin\theta + (y' - k)\cos\theta.$$

Substituting into Ax + By + C yields the equation

$$A'x + B'x + C' = 0$$

where

$$A' = A\cos\theta - B\sin\theta$$

$$B' = B\cos\theta + A\sin\theta$$
  

$$C' = Ah + Bk + C - (Ah + Bk)\cos\theta - (Ak - Bh)\sin\theta.$$

Rotating the quadratic equation  $Q \equiv Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$  is accomplished in the same manner, by replacing x and y with the proper rotated coordinates. The resulting expressions for the coefficients of the rotated quadratic equation,

$$Q' \equiv A'x^{2} + B'xy + C'y^{2} + D'x + E'y + F' = 0,$$

are somewhat long, but can be written symbolically as

$$\begin{array}{rcl} A' &=& A\cos^2\theta - B\cos\theta\sin\theta + C\sin^2\theta\\ B' &=& B(\cos^2\theta - \sin^2\theta) + 2(A-C)\cos\theta\sin\theta\\ C' &=& A\sin^2\theta + B\cos\theta\sin\theta + C\cos^2\theta\\ D' &=& (-2Ch+Bk)\sin^2\theta - (2Ah+Bk)\cos^2\theta + \\&& 2(Bh-(A-C)k)\cos\theta\sin\theta + \\&& (2Ah+Bk+D)\cos\theta - (Bh+2Ck+E)\sin\theta\\ E' &=& (Bh-2Ak)\sin^2\theta - (Bh+2Ck)\cos^2\theta - \\&& 2((A-C)h+Bk)\cos\theta\sin\theta + \\&& (Bh+2Ck+E)\cos\theta + (2Ah+Bk+D)\sin\theta\\ F' &=& (Ah^2+Bhk+Ck^2)\cos^2\theta - \\&& (B(h^2-k^2) - 2(A-C)hk)\cos\theta\sin\theta + \\&& (Ch^2-Bhk+Ak^2)\sin^2\theta - \\&& (2Ah^2+2Bhk+2Ck^2+Dh+Ek)\cos\theta + \\&& (Bh^2-2(A-C)hk-Bk^2+Eh-Dk)\sin\theta + \\&& Ah^2+Bhk+Ck^2+Dh+Ek+F. \end{array}$$

By applying the formulas for rotating coordinates, linear equations and quadratic equations, we can now specify how to rotate all of the *Descarta2D* objects. Points and lines can be rotated by directly applying the formulas for coordinates and linear equations, respectively. Curves that are located by points can be rotated by rotating the defining points; additionally, curves that have orientation angles, such as arcs, parabolas, ellipses and hyperbolas, are rotated by adding the rotation angle,  $\theta$ , to the angle of the curve.

**Example.** Rotate the ellipse

$$\frac{(x-3)^2}{1} + \frac{y^2}{4} = 1$$

 $\pi/2$  radians about its center point and about the origin. Plot all three ellipses.

**Solution.** The Descarta2D function Rotate2D[object,  $\theta$ , { $x_0$ ,  $y_0$ }] rotates an object by angle  $\theta$  about point ( $x_0, y_0$ ). The function Rotate2D[object,  $\theta$ ] rotates an object by angle  $\theta$  about the origin. The object may be a coordinate list, a geometric object or a list of Descarta2D objects.

```
Out[6] {Ellipse2D[{3, 0}, 2, 1, 0], Ellipse2D[{0, 3}, 2, 1, 0]}
```

```
In[7]: Sketch2D[{e1, e2, e3}];
```



# 15.3 Scaling

A scaling transformation maps coordinates (x, y) to (x', y') using the transformation equations

$$x' = h + s(x - h)$$
 and  $y' = k + s(y - k)$ .

The scale factor, s > 0, indicates the ratio of corresponding lengths of the scaled object with respect to the original object. The point (h, k) is called the *center* of scaling. A point at the center of scaling does not change coordinates during a scaling transformation.

Solving the scaling transformation equations for (x, y) in terms of (x', y') yields

$$x = h + \frac{(x'-h)}{s}$$
 and  $y = k + \frac{(y'-k)}{s}$ .

Substituting the coordinates (x, y) into the linear polynomial Ax + By + C yields the scaled linear polynomial

$$Ax + By + Ah(s-1) + Bk(s-1) + sC.$$

Similarly, applying a scaling transformation to the quadratic polynomial

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F$$

F

F

yields

$$A'x^{2} + B'xy + C'y^{2} + D'x + E'y + F' = 0$$

where

The scaling transformation may be applied to *Descarta2D* geometric objects by applying the coordinate scaling transformation to the positioning arguments and simultaneously multiplying the length arguments by the scale factor.

**Example.** Scale the circle  $(x - 2)^2 + (y - 1)^2 = 1$  by a factor of 3/2 about its center point and the origin. Plot the three circles.

**Solution.** The function  $Scale2D[object, s, \{h, k\}]$  scales the object using scale factor s about the center of scaling (h, k). Scale2D[object, s] scales the object about the origin. The object may be a coordinate list, a geometric object or a list of Descarta2D objects.

```
\textit{Out[8]} \left\{ \texttt{Circle2D}\left[ \left\{ 2,\,1 \right\},\,\frac{3}{2} \right],\,\texttt{Circle2D}\left[ \left\{ 3,\,\frac{3}{2} \right\},\,\frac{3}{2} \right] \right\}
```

```
In[9]: Sketch2D[{c1, c2, c3}];
```



\_

### 15.4 Reflections

A reflection transformation maps the coordinates (x, y) to coordinates that are the "mirror" reflection of the coordinates with respect to a line that represents the position of the mirror. Consider the point  $P_1(x_1, y_1)$  and the reflection line  $L_2 \equiv A_2 x + B_2 y + C_2 = 0$ . The following Descarta2D commands can be used to determine the coordinates of the reflection of point  $P_1$  in line  $L_2$ .

In standard mathematical notation the coordinates  $P_2(x_2, y_2)$  of the reflected point are given by the transformation equations

$$\begin{aligned} x_2 &= x_1 - \frac{2A_2(A_2x_1 + B_2y_1 + C_2)}{A_2^2 + B_2^2} \\ y_2 &= y_1 - \frac{2B_2(A_2x_1 + B_2y_1 + C_2)}{A_2^2 + B_2^2}. \end{aligned}$$

Solving the transformation equations for  $P_1(x_1, y_1)$  in terms of  $P_2(x_2, y_2)$  yields

$$\begin{aligned} x_1 &= \frac{(B_2^2 - A_2^2)x_2 - 2A_2B_2y_2 - 2A_2C_2}{A_2^2 + B_2^2} \\ y_1 &= \frac{-2A_2B_2x_2 + (A_2^2 - B_2^2)y_2 - 2B_2C_2}{A_2^2 + B_2^2}. \end{aligned}$$

Mathematica Hint. While it is feasible to solve these equations manually using algebra, it is much less effort to let Mathematica do the work using the Solve function. The command would be of the form

 $\begin{aligned} & \text{In[11]: Solve[{x2 == XCoordinate2D[p2],} \\ & \text{y2 == YCoordinate2D[p2]}, {x1, y1}] \text{ // Simplify} \\ & \text{Out[11] } \left\{ \left\{ x1 \rightarrow \frac{-A2^2 x2 + B2^2 x2 - 2 A2 (C2 + B2 y2)}{A2^2 + B2^2}, y1 \rightarrow \frac{-2 B2 (C2 + A2 x2) + A2^2 y2 - B2^2 y2}{A2^2 + B2^2} \right\} \right\} \end{aligned}$ 

Substituting the coordinates  $(x_1, y_1)$  into a linear equation  $A_1x + B_1y + C_1$  yields the reflected linear equation

$$A_3x + B_3y + C_3$$

where

$$A_{3} = A_{1}(B_{2}^{2} - A_{2}^{2}) - 2B_{1}A_{2}B_{2}$$
  

$$B_{3} = B_{1}(A_{2}^{2} - B_{2}^{2}) - 2A_{1}A_{2}B_{2}$$
  

$$C_{3} = C_{1}(A_{2}^{2} + B_{2}^{2}) - 2C_{2}(A_{1}A_{2} + B_{1}B_{2})$$

Substituting the coordinates  $(x_1, y_1)$  into a quadratic polynomial

$$Q \equiv Ax^2 + Bxy + Cy^2 + Dx + Ey + F$$

yields a quadratic Q' reflected in the line  $L_2 \equiv A_2 x + B_2 y + C_2 = 0$  of the form

$$Q' \equiv A'x^{2} + B'xy + C'y^{2} + D'x + E'y + F$$

where

$$\begin{array}{rcl} A' &=& Af_1f_2 + 2Bpf_4 + 4Cp^2 \\ B' &=& 4Apf_4 + B(4p^2 - f_4^2) - 4Cpf_4 \\ C' &=& 4Ap^2 - 2Bpf_4 + Cf_1f_2 \\ D' &=& 4Aqf_4 + 2Br(2A_2^2 + f_4) + 8CqB_2^2 - Df_4f_3 - 2Epf_3 \\ E' &=& 8AA_2^2r + 2Bq(2B_2^2 - f_4) - 4Crf_4 - 2Dpf_3 + Ef_3f_4 \\ F' &=& 4(Aq^2 + BpC_2^2 + Cr^2) - 2f_3(Dq + Er) + Ff_3^2 \end{array}$$

and

$$p = A_2 B_2, \quad q = A_2 C_2, \quad r = B_2 C_2,$$
  
$$f_1 = (A_2 + B_2)^2, \quad f_2 = (A_2 - B_2)^2,$$
  
$$f_3 = (A_2^2 + B_2^2), \quad f_4 = (A_2^2 - B_2^2).$$

### **Reflection of an Angle**

E

F

What angle does a reflected line make with the +x-axis? Let L be a line that makes an angle  $\theta$  with the +x-axis,  $L_R$  be a reflection line that makes an angle  $\alpha$  with the +x-axis, and L' the reflection of L in  $L_R$  as shown in Figure 15.2. We wish to determine the angle  $\theta'$  that L' makes with the +x-axis. Using the fact that supplementary angles sum to  $\pi$  and that the interior angles of a triangle sum to  $\pi$ , the angle  $\theta' = 2\alpha - \theta$ . The relationship also holds true when  $\alpha = \theta$ , in which case L and L' do not intersect. By applying the methods for reflecting coordinates, equations and angles we are able to reflects all of the geometric objects in the Descarta2D system.

**Example.** Reflect the arc centered at (3, 2) with radius 1 and start and end angles of  $\pi$  and  $3\pi/2$  in the line x + 3y + 2 = 0.



Figure 15.2: A reflected angle.

**Solution.** The Descarta2D function Reflect2D[object, line] reflects the object with respect to the line. The object may be a coordinate list, a geometric object, or a list of objects.

Out[12] Arc2D[{2, -3},  $\left\{-\frac{4}{5}, -\frac{17}{5}\right\}, -1 + \sqrt{2}$ ]

```
In[13]: Sketch2D[{a1, l1, a2}];
```



# 15.5 Explorations

REFLECTION IN A POINT. ..... reflection of a point P(x, y) in the point C(H, K) if C is the midpoint of the segment PP'. Using this definition show that

• The transformation equations for a reflection in a point are

$$x' = 2H - x$$
  $x = 2H - x'$   
 $y' = 2K - y$   $y = 2K - y'$ .

• The reflection of the line ax + by + c = 0 in the point (H, K) is

$$ax + by - (2aH + 2bK + c) = 0.$$

• The reflection of the quadratic  $ax^2 + bxy + cy^2 + dx + ey + f = 0$  in the point (H, K) is  $ax^2 + bxy + cy^2 - (4aH + 2bK + d)x - (2bH + 4cK + e)y + 4aH^2 + 4bHK + 4cK^2 + 2dH + 2eK + f = 0.$ 

Also, verify that the reflection in a point transformation is equivalent to a rotation of  $\pi$  radians about the reflection point (H, K).

INVERSION. ..... inverse.nb A point P'(x', y') is said to be the *inverse* of a point P(x, y) in the circle

$$C \equiv (x-h)^2 + (y-k)^2 = r^2$$

if points O(h,k), P and P' are collinear and  $|OP| |OP'| = r^2$ . Using this definition show that

• The coordinates of P'(x', y') are

$$x' = h + \frac{r^2(x-h)}{(x-h)^2 + (y-k)^2}$$
 and  $y' = k + \frac{r^2(y-k)}{(x-h)^2 + (y-k)^2}$ .

• If the circle of inversion is  $x^2 + y^2 = 1$ , the coordinates of P' are

$$x' = \frac{x}{x^2 + y^2}$$
 and  $y' = \frac{y}{x^2 + y^2}$ .

• If the circle of inversion is  $x^2 + y^2 = 1$ , the inverse of the line  $L \equiv A_1 x + B_1 y + C_1 = 0$ , assuming L does not pass through the origin, is the circle

$$\left(x + \frac{A_1}{2C_1}\right)^2 + \left(y + \frac{B_1}{2C_1}\right)^2 = \frac{A_1^2 + B_1^2}{4C_1^2}$$

- If the circle of inversion is  $x^2 + y^2 = 1$ , the inverse of the line  $L \equiv A_1 x + B_1 y + C_1 = 0$ , assuming L passes through the origin  $(C_1 = 0)$ , is L itself.
- If the circle of inversion is  $x^2 + y^2 = 1$ , the inverse of the circle  $(x h_1)^2 + (y k)^2 = r_1^2$  is

$$\left(x - \frac{h_1}{D}\right)^2 + \left(y - \frac{k_1}{D}\right)^2 = \frac{r_1^2}{D}, \text{ where } D = h_1^2 + k_1^2 - r_1^2$$

• If the circle of inversion is  $x^2 + y^2 = 1$ , the inversion of  $C \equiv (x-h)^2 + (y-k)^2 = h_1^2 + k_1^2$ , which passes through the origin, is the line  $L \equiv 2h_1x + 2k_1y = 1$ . L is parallel to the tangent line to C through the origin. The equation of the tangent line is  $2h_1x + 2k_1y = 0$ .

Inversion is clearly a *non-rigid* transformation.

# Chapter 16

# Arc Length

Intuitively, *arc length* is a measure of distance along a curve. For a straight line the distance is called the *length* and is easily computed using the distance formula. For some curves the arc length has other special names such as the *perimeter* of a triangle or the *circumference* of a circle. This chapter discusses methods for computing the arc lengths of simple geometric curves, such as those provided in *Descarta2D*.

# 16.1 Lines and Line Segments

#### Length of a Line

F

By definition, a line is a curve of infinite length. We can, however, specify two parameter values on the line and determine the distance between the points associated with these parameter values. Since lines in *Descarta2D* are parameterized by distance, the distance, s, between the points represented by any two parameter values,  $t_1$  and  $t_2$ , is simply the absolute value of the difference of the parameter values,  $s = |t_2 - t_1|$ .

**Example.** Find the distance between the parameter values -2 and 1 on any line (assuming the parameterizations defined in the *Descarta2D* packages).

**Solution.** The function  $ArcLength2D[line, \{t_1, t_2\}]$  returns the arc length between two parameter values on a line.

#### Length of a Line Segment

The *length* of a line segment is the distance between its start and end points. In *Descarta2D* the start and end points have parameter values of 0 and 1, respectively. The distance, s, between any two parameter values,  $t_1$  and  $t_2$ , is given by  $|t_2 - t_1|l$ , where l is the length of the line segment.

**Example.** Find the length of the line segment connecting the points (1,3) and (2,4). Find the arc length on the line segment between the parameter values 1/4 and 1/2.

**Solution.** The function Length2D[*lnseg*] returns the length of a line segment (the distance between the start and end points). ArcLength2D[*lnseg*,  $\{t_1, t_2\}$ ] returns the distance between two parameter values on a line segment.

```
In[2]: l1 = Segment2D[{1, 3}, {2, 4}];
        {Length2D[l1], ArcLength2D[l1, {1/4, 1/2}]}
Out[2] {\sqrt{2}, \frac{1}{2\sqrt{2}}}
```

# 16.2 Perimeter of a Triangle

The sum of the lengths of the sides of a triangle is called the *perimeter*, s, and is given by  $s = s_1 + s_2 + s_3$ , where  $s_n$  is the length of side n of the triangle.

**Example.** Find the perimeter of a triangle whose vertices are (1, 2), (3, 4) and (5, 6).

**Solution.** The Descarta2D function Perimeter2D[*triangle*] returns the perimeter of a triangle.

In [3]: Perimeter2D[Triangle2D[{1, 2}, {4, 4}, {5, 6}]] Out[3]  $4\sqrt{2} + \sqrt{5} + \sqrt{13}$ 

F



Figure 16.1: Circle approximated by an inscribed polygon.

### 16.3 Polygons Approximating Curves

If we *inscribe* a polygon in any closed curve, it is evident that as the number of sides of the polygon is increased, the area of the polygon approaches the area bounded by the curve. Likewise, the perimeter of the polygon approaches the perimeter, or *arc length*, of the curve. If the number of sides of the polygon is increased *ad infinitum*, the polygon will coincide with the curve. In like manner, we can see that as the number of sides of a *circumscribing* polygon is increased, the more nearly its area and perimeter will approach the area and perimeter of the curve. Therefore, when investigating the area or arc length of a curve, we may substitute for the curve an inscribed or circumscribed polygon with an indefinitely increasing number of sides. These notions are formalized in the study of calculus, but they can be applied intuitively in the study of areas and perimeters of simple curves as will be shown in the following sections.

# 16.4 Circles and Arcs

#### Circumference of a Circle

Consider the circle shown in Figure 16.1. The length, d, of the perpendicular segment from the center of the circle to one of the sides of a regular, inscribed polygon is given by  $d = r \sin(\frac{1}{2}\theta)$  where r is the radius of the circle and  $\theta$  is angle between adjacent radii connecting the sides of the polygon. The length of the sides of the polygon, s, is given by  $s = 2r \cos(\frac{1}{2}\theta)$ . Clearly, the perimeter of the inscribed polygon, S, is given by S = ns, where n represents the number of sides of the polygon. Now consider the ratio of the perimeter of polygons for two circles,

 $C_1$  and  $C_2$ , which is given by

$$\frac{S_1}{S_2} = \frac{ns_1}{ns_2} = \frac{2nr_1\cos\left(\frac{1}{2}\theta\right)}{2nr_2\cos\left(\frac{1}{2}\theta\right)} = \frac{r_1}{r_2}$$

(1)

As n increases  $S_1$  and  $S_2$  approach the circumferences of  $C_1$  and  $C_2$ ; therefore, the ratio of the circumferences of two circles equals the ratio of their radii. Since the radii of the circles are proportional to their diameters, the ratio of the circumferences to the diameters is also a constant which has been given the symbol  $\pi$ . Therefore,

$$\frac{S}{D} \equiv \pi$$

relating the circumference of a circle to its diameter is a constant for all circles; or writing in a different form, the circumference S of a circle is given by

$$S = \pi D = 2\pi r.$$

**Example.** Find the circumference of a circle centered at (0,0) with a radius of 2. Also, find the arc length of 1/4 of the circle's circumference.

**Solution.** The function Circumference2D[*circle*] returns the circumference of a circle. The function ArcLength2D[*circle*,  $\{t_1, t_2\}$ ] returns the arc length of a circle between two parameter values.

```
In[4]: cl = Circle2D[{0, 0}, 2];
        {Circumference2D[cl], ArcLength2D[cl, {0, Pi / 2}]}
Out[4] {4 π, π}
```

#### Arc Length of an Arc

The arc length, s, (or span) of an arc is the ratio of the angular span of the arc to the angular span of a full circle  $(2\pi)$  times the circumference of a circle and is given by

$$s = \frac{\theta}{2\pi}(2\pi r) = (\theta_2 - \theta_1)r.$$

**Example.** Find the arc length of the sector defined by the arc centered at (0,0) with radius 2 and start and end angles of  $\pi/4$  and  $3\pi/4$ .

Solution. The function Span2D[arc] returns the arc length of an arc.

```
In[5]: Span2D[a1 = Arc2D[Point2D[{0, 0}], 2, {Pi/4, 3Pi/4}]] // Simplify
Out[5] π
```

**Example.** For the arc defined in the previous example, find the arc length between the parameter values 0.25 and 0.75.

**Solution.** The function ArcLength2D[*arc*,  $\{t_1, t_2\}$ ] returns the arc length of an arc between two parameter values.

```
In[6]: ArcLength2D[a1, {0.25, 0.75}] // N
Out[6] 1.5708
```

# 16.5 Ellipses and Hyperbolas

If  $x = f_x(t)$  and  $y = f_y(t)$  are the parametric equations of a curve, then the arc length, s, of the curve between parameter values  $t_1$  and  $t_2$  is given by the integral

$$s = \int_{t_1}^{t_2} \sqrt{(x')^2 + (y')^2} \, dt$$

where x' and y' are the derivatives of the parametric equations of the curve with respect to t. For many curves this integral is difficult to evaluate in symbolic form, but by using the numerical integration capabilities of *Mathematica* we can find an approximate arc length.

Even for the conic curves (except the parabola, which we will discuss subsequently) the integral for arc length leads to *elliptic integrals*, a class of integrals that cannot be expressed in closed form in terms of elementary functions. This does not mean that these integrals do not exist, but require the definition of *non-elementary* functions. Fortunately, the *elliptic integral* needed to evaluate the arc lengths of ellipses and hyperbolas is built-in to *Mathematica* as the EllipticE[*phi*, *m*] function, which is written  $E(\phi|m)$  in traditional mathematical notation. The arc length, *s*, in the parameter range  $0 \le t \le 2\pi$ , of an ellipse in terms of this elliptic integral is given by

$$s = b E\left(t \mid 1 - \frac{a^2}{b^2}\right)$$

where a and b are the lengths of the semi-major and semi-minor axes, respectively, of the ellipse. Since all elliptic arcs can be expressed as sums or differences of such arcs, the formula serves to provide a means for expressing the arc length between any pair of parameters.

Similarly, the arc length, s, of a hyperbola, using the parametric equations for a hyperbola defined in *Descarta2D*, can be expressed in terms of this elliptic integral and is given by

$$s = i \ b \ E\left(i \cos^{-1}\left(\sqrt{\frac{a^2 + b^2}{a}}\right) t \ \middle| \ 1 + \frac{a^2}{b^2}\right)$$

where a and b are the lengths of the semi-transverse and semi-conjugate axes, respectively, of the hyperbola and  $i = \sqrt{-1}$ . Even though complex numbers are present in this formula, the resulting arc length is a real number.

**Example.** Find the approximate arc length of the ellipse  $x^2/9 + y^2/4 = 1$  between parameter values 0 and  $\pi/2$ .

**Solution.** The Descarta2D function ArcLength2D[curve,  $\{t_1, t_2\}$ ] returns the arc length of a curve between two parameter values.

```
In[7]: e1 = Ellipse2D[{0, 0}, 3, 2, 0];
ArcLength2D[e1, {0, Pi/2}] // N
Out[7] 3.96636
```

# 16.6 Parabolas

Consider a parabola represented by the parametric equations

 $x = ft^2$  and y = 2ft.

The arc length, s, of such a parabola between two parameters,  $t_1 < t_2$ , can be derived in terms of elementary functions. The derivation is provided in the exploration pbarclen.nb where the arc length is shown to be  $s = f(S_2 - S_1)$  where

$$S_n = t_n \sqrt{1 + t_n^2} + \sinh^{-1}(t_n).$$

**Example.** Find the arc length of the parabola  $y^2 = 4x$  between parameter values -2 and 3. Find the arc length cut off by the focal chord of the parabola.
**Solution.** The Descarta<sub>2D</sub> function ArcLength<sub>2D</sub>[parabola,  $\{t_1, t_2\}$ ] returns the arc length of the parabola between the two parameter values. The focal chord of a parabola has end points at parameter values  $\pm 1$ .

#### 16.7 Chord Parameters

For some curves, such as circles and ellipses, it is fairly easy to determine the parameter value that corresponds to a particular point on the curve; however, for hyperbolas and parabolas, whose parametric representation is more complex, it may be difficult to determine the parameter values needed to compute the arc length of some specific portion of the curve. The function Parameters2D provides a more geometric definition of the chord that can be used with the arc length functions. Essentially, the Parameters2D function computes the parameter values of the points of intersection between a line and a second-degree curve (circle, ellipse, hyperbola or parabola). This function will also be useful in the area functions introduced in the next chapter.

**Example.** Find the arc length of the parabola with vertex at (0,0), focal length of 1 (opening upward) cut off by the line 2x + 4y - 5 = 0.

**Solution.** The Descarta2D function Parameters2D[*line*, *curve*] returns a list of the two parameters which are the points of intersection between the line and the curve. The curve may be a circle, an ellipse, a hyperbola or a parabola.

```
In [9]: pl = Parabola2D[{0, 0}, 1, Pi / 2];
Il = Line2D[2, 4, -5];
tl2 = Parameters2D[11, pl]
Out [9] \left\{ \frac{1}{2} \left( 1 - \sqrt{6} \right), \frac{1}{2} \left( 1 + \sqrt{6} \right) \right\}
In [10]: ArcLength2D[pl, tl2] // FullSimplify
Out [10] \frac{1}{4} \sqrt{202 + 10\sqrt{97}} - Log \left[ 1 - \sqrt{6} + \sqrt{11 - 2\sqrt{6}} \right] + Log \left[ 1 + \sqrt{6} + \sqrt{11 + 2\sqrt{6}} \right]
```

-

F

**Descarta2D Hint.** Only the primary branch of a hyperbola in standard position is parameterized (the primary branch is the branch opening to the right when the hyperbola's rotation angle is zero); positions on the other branch are generated by reflecting coordinates on the primary branch. As a result of this parameterization, the Parameters2D function will only return parameter values if the line intersects the primary branch of the hyperbola.

## 16.8 Summary of Arc Length Functions

Descarta2D provides a general function, ArcLength2D for computing the arc length of parametric curves and several special functions for computing arc lengths of specific curves. The Descarta2D function ArcLength2D[curve,  $\{t_1, t_2\}$ ] can be used to compute the arc length of any parametric curve in Descarta2D (arcs, lines, line segments, circles, parabolas, ellipses, hyperbolas and conic arcs). The function Length2D[lnseg] computes the length of a complete line segment. The function Circumference2D[curve] computes the arc length of a complete circle or ellipse. The function Span2D[curve] computes the arc length of a complete arc or conic arc. The function Perimeter2D[triangle] computes the perimeter of a triangle.

## 16.9 Explorations

ARC LENGTH OF A PARABOLA.....pbarclen.nb Show that the arc length, s, of a parabola whose parametric equations are

$$x = ft^2$$
 and  $y = 2ft$ 

is given by  $s = f(S_2 - S_1)$  where

$$S_n = t_n \sqrt{1 + t_n^2} + \sinh^{-1}(t_n).$$

APPROXIMATE ARC LENGTH OF A CURVE. .....narclen.nb

The arc length of a smooth, parametrically defined curve can be approximated by a polygon connecting a sequence of points on the curve. Write a *Mathematica* function of the form NArcLength2D[*crv*, { $t_1$ ,  $t_2$ }, n] that approximates the arc length of a curve between two parameter values using a specified number of coordinates at equal parameter intervals between the two given parameters. Produce a graph illustrating the convergence of the approximation to the *Descarta2D* function ArcLength2D[*crv*, { $t_1$ ,  $t_2$ }] //N.

Using exact integration in Mathematica show that the arc length of a parabolic conic arc with control points  $P_0(0,0)$ ,  $P_A(a,b)$ , and  $P_1(1,0)$  can be expressed exactly in symbolic form in terms elementary functions of a and b.

Arc Length of a Parabolic Conic Arc.....caarclen.nb

# Chapter 17

# Area

Intuitively, *area* is the measure of the number of unit squares that can be contained inside a boundary. For a square with sides of length s, the area, A, is given by  $A = s^2$ . For a rectangle with sides a and b, A = ab. As the boundary becomes more complex or contains curved elements, the computation of the area requires more complex considerations. In this chapter we will derive formulas for the areas of *Descarta2D* objects.

# 17.1 Areas of Geometric Figures

Before exploring formulas for computing areas using analytic geometry, we will look at some formulas from planar geometry. Consider the right triangle ABC shown in Figure 17.1 with height h and base b. Clearly, the area of  $\triangle ABC$  is one-half of the area of rectangle ABCD, so the area, A, of a right triangle is

$$A = \frac{bh}{2}.$$

Now consider the acute  $\triangle ABC$  in the center of Figure 17.1. The area of ABC is given by

Area $ABC=\ {\rm Area}\ BCDE-\ {\rm Area}\ ABE-\ {\rm Area}\ ACD$ 



Figure 17.1: Areas of right, acute and obtuse triangles.



Figure 17.2: Area of a trapezoid.

or

Area 
$$ABC = bh - \frac{b_1h}{2} - \frac{b_2h}{2}$$
.

Simplifying and using  $b = b_1 + b_2$  yields

$$A = \frac{bh}{2}.$$

The same formula results for the area of an obtuse triangle (using  $b = b_2 - b_1$ ), as shown on the right in Figure 17.1.

Now consider the trapezoid ABCD shown in Figure 17.2. The area of ABCD is given by

Area 
$$ABCD =$$
 Area  $ABEF -$  Area  $ADF -$  Area  $BCE$ 

or

Area 
$$ABCD = bh - \frac{hb_1}{2} - \frac{hb_2}{2}$$
.

Simplifying and using  $b = a + b_1 + b_2$  yields

$$A = \frac{(a+b)}{2}h.$$

These formulas from planar geometry will be useful in upcoming sections for deriving the formulas using analytic geometry.

#### **Triangular Area**

There are several formulas for the area, A, of a triangle that involve lengths associated with the triangle. The simplest is the familiar A = bh/2, where b is the length of one of the sides of the triangle (the *base*) and h is the *height* of the triangle (the distance from the base to the opposite vertex).



Figure 17.3: Area of a triangle by coordinates.

The formula of Heron gives the area of the triangle in terms of the lengths of its sides,  $s_n$ , alone:

$$A = \sqrt{s(s-s_1)(s-s_2)(s-s_3)}$$

where  $s = (s_1 + s_2 + s_3)/2$  is the *semi-perimeter*. This formula is derived in the exploration heron.nb.

Since a triangle is represented in *Descarta2D* by the coordinates of the vertices, we wish to derive a formula based on the coordinates. Consider the triangle ABC as shown in Figure 17.3, where the coordinates are  $A(x_1, y_1)$ ,  $B(x_2, y_2)$  and  $C(x_3, y_3)$ . Projecting A, B and C onto the x-axis produces three points  $A'(x_1, 0)$ ,  $B'(x_2, 0)$  and  $C'(x_3, 0)$ . The area of triangle ABC is given by

Area 
$$ABC =$$
 Area  $AA'C'C -$  Area  $AA'B'B -$  Area  $BB'C'C$ 

The height and base lengths of these trapezoids can be determined as the difference of the coordinates of the points, yielding

Area 
$$ABC = \frac{(y_1 + y_3)(x_3 - x_1)}{2} - \frac{(y_1 + y_2)(x_2 - x_1)}{2} - \frac{(y_2 + y_3)(x_3 - x_2)}{2}$$

Expanding and rearranging will show that the area of a triangle, A, is given by the determinant

$$A = \pm \frac{1}{2} \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

where  $(x_1, y_1)$ ,  $(x_2, y_2)$  and  $(x_3, y_3)$  are the coordinates of the vertices of the triangle. The sign is selected to yield a positive area.

Alternately, if we multiply the length of the line segment joining two of the points, by the length of the perpendicular line segment on that line from the third point, we have double the area of the triangle determined by the three points. **Example.** Find the area of a triangle whose vertices are (1, 2), (4, 4) and (5, 6).

Solution. The function Area2D[triangle] returns the area of a triangle.

```
In[1]: Area2D[Triangle2D[{1, 2}, {4, 4}, {5, 6}]]
Out[1] 2
```

# 17.2 Curved Areas

If we *inscribe* a polygon inside any closed curve, it is evident that as the number of sides of the polygon is increased, the area of the polygon approaches the area bounded by the curve. Likewise, the perimeter of the polygon approaches the perimeter, or *arc length*, of the curve. If the number of sides of the polygon is increased *ad infinitum*, the polygon will coincide with the curve. In like manner, we can see that as the number of sides of a *circumscribing* polygon is increased, the more nearly its area and perimeter will approach the area and perimeter of the curve. Therefore, when investigating the area or perimeter of a closed curve, we may substitute for the curve an inscribed or circumscribed polygon with an indefinitely increasing number of sides. These notions are formalized in the study of calculus, but they can be applied intuitively in the study of areas of simple curves as will be shown in the following sections.

#### 17.3 Circular Areas

To determine the area of a circle, we examine a polygon inscribed in the circle as shown in Figure 17.4. The area of any triangle in the figure is given by  $A_{\triangle} = \frac{1}{2}sd$ , and the area of the entire polygon is given by  $nA_{\triangle}$ , because there are *n* such triangles. As *n* increases without limit, we find that *ns* approaches  $S = 2\pi r$  and *d* approaches *r*. Therefore, the area of the polygon approaches  $\frac{1}{2}Sr$  or  $\pi r^2$ . Accordingly, the area, *A*, of a circle is given by  $A = \pi r^2$ , where *r* is the radius of the circle.

**Example.** Find the area of a circle centered at (0,0) with a radius of 2.

Solution. The Descarta2D function Area2D[circle] returns the area of a circle.

```
In[2]: Area2D[Circle2D[{0, 0}, 2]]
Out[2] 4 π
```

L

F

F



Figure 17.4: Circle approximated by an inscribed polygon.

The area of an *arc sector* of radius r as shown in Figure 17.5 may be determined as the ratio of the angular span of the arc to the span of a complete circle ( $2\pi$  radians) times the area of the circle. Since the area of a complete circle is  $\pi r^2$ , the area of an arc sector is given by

$$A = \frac{r^2}{2}(\theta_2 - \theta_1).$$

**Example.** Find the area of the sector defined by the arc centered at (0,0) with radius 2 and start and end angles of  $\pi/4$  and  $3\pi/4$ .



Figure 17.5: Areas of an arc sector and segment.

\_

**Solution.** The Descarta2D function SectorArea2D[circle,  $\{\theta_1, \theta_2\}$ ] returns the area of the sector defined by an arc of a circle.

In[3]: SectorArea2D[Circle2D[{0, 0}, 2], {Pi/4, 3 Pi/4}]
Out[3] π

The area of an *arc segment*, which is the area bounded by the arc and the chord connecting the end points of the arc as shown in Figure 17.5, is calculated as the *difference* of the areas of the sector and the triangle whose vertices are the end points and the center. Since the area of this triangle is  $A = \frac{1}{2}r^2 \sin \theta$ , the formula for the area of the arc segment is

$$A = \pi r^2 \left(\frac{\theta}{2\pi}\right) - \frac{r^2}{2}\sin\theta$$
$$= \frac{r^2}{2} \left(\theta - \sin\theta\right)$$

where  $\theta = \theta_2 - \theta_1$  is the span of the arc, and r is the radius of the arc.

**Example.** Find the area of the segment defined by the arc centered at (0,0) with radius 2 and start and end angles of  $\pi/4$  and  $3\pi/4$ .

**Solution.** The Descarta2D function Area2D[arc] returns the area of the segment defined by an arc and its chord.

```
In[4]: Area2D[Arc2D[Point2D[{0, 0}], 2, {Pi/4, 3Pi/4}]] // Simplify
Out[4] -2 + π
```

Notice that if the angle  $\theta$  is greater than  $\pi$  radians, the formula is still valid because  $\sin \theta$  will be negative and the area of the central triangle will be added to the sector area producing the correct result.

# 17.4 Elliptic Areas

The area of an ellipse depends only on the lengths of its semi-major and semi-minor axes, and is independent of its position and orientation. It is shown in calculus that integrating the equation y = f(x) of the curve between limits on the x-axis produces the area between the curve and the x-axis. The equation of an ellipse in standard position is given by

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

F

Solving for y yields  $y = b\sqrt{1 - x^2/a^2}$  for the upper portion of the ellipse. The following steps outline the integration process that is used to compute the area:

$$A = \int_{-a}^{a} y \, dx$$
$$= \int_{-a}^{a} b \sqrt{1 - x^2/a^2} \, dx$$
$$= \frac{\pi a b}{2}.$$

Therefore, the area of the complete ellipse (both the upper and lower portions) is given by

$$A = \pi a b$$

where a and b are the lengths of the semi-major and semi-minor axes, respectively, of the ellipse.

**Example.** Calculate the area of the ellipse  $\frac{x^2}{4} + \frac{y^2}{9} = 1$ .

Solution. The Descarta2D function Area2D[ellipse] returns the area of an ellipse.

```
In[5]: Area2D[Ellipse2D[{0, 0}, 3, 2, Pi / 2]]
```

Out[5] 6π

F

Consider an ellipse in standard position with the equation

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

as shown in Figure 17.6. The area, A, of a *segment* of the ellipse bounded by the chord defined from (a, 0) and a general point on the ellipse,  $(a \cos \theta, b \sin \theta)$ , can be determined by subtracting the area under the chord from the area under the ellipse between limits on the *x*-axis. The equation of the line containing the chord is given by

$$y_2 = \frac{b\sin\theta(a-x)}{a(1-\cos\theta)}$$

as can be determined from the two-point form of a line. The area of the segment is determined using integration as follows:

$$A = \int_{a\cos\theta}^{a} (y_1 - y_2) dx$$
  
= 
$$\int_{a\cos\theta}^{a} \left( b\sqrt{1 - x^2/a^2} - \frac{b\sin\theta(a - x)}{a(1 - \cos\theta)} \right) dx$$
  
= 
$$\frac{ab}{4} \left( \pi - 2\sin^{-1}(\cos\theta) - 2\sin\theta \right).$$



Figure 17.6: Areas of an elliptic segment and sector.

The formula is valid for angles  $\theta$  in the range  $0 \le \theta \le \pi$ . Since all segments can be computed as sums or differences of such segments and simple triangles, the area of all ellipse segments can be determined using this basic formula.

**Example.** Find the area of the ellipse segment from  $\pi/6$  to  $\pi/3$  radians for an ellipse with semi-major axis length of 3 and semi-minor axis length of 1.

**Solution.** The function SegmentArea2D[*ellipse*,  $\{t_1, t_2\}$ ] returns the area of an ellipse segment defined by two parameter values.

In [6]: SegmentArea2D[Ellipse2D[{0, 0}, 3, 1, 0], {Pi/6, Pi/3}] Out [6]  $-\frac{3}{4} + \frac{\pi}{4}$ 

The area of an ellipse *sector*, as illustrated in Figure 17.6, can be found by adding the area of the triangle formed by the sector sides and the chord of the sector. The area of the triangle is given by

$$A_{\triangle} = \frac{bh}{2} = \frac{ab\sin\theta}{2}$$

and the resulting formula for the area of the sector is given by

$$A = \frac{ab}{4} \left( \pi - 2\sin^{-1}(\cos\theta) \right).$$

**Example.** Find the area of the ellipse sector from  $\pi/6$  to  $\pi/3$  radians for an ellipse with semi-major axis length of 3 and semi-minor axis length of 1.

F

 $\vdash$ 



Figure 17.7: Areas of a hyperbolic segment and sector.

**Solution.** The Descarta2D function SectorArea2D[*ellipse*,  $\{t_1, t_2\}$ ] returns the area of an ellipse sector defined by two parameter values.

```
In[7]: SectorArea2D[Ellipse2D[{0, 0}, 3, 1, 0], {Pi/6, Pi/3}]
Out[7] π/4
```

# 17.5 Hyperbolic Areas

Using the integration techniques demonstrated previously for ellipses the areas associated with hyperbolas can also be computed. Consider the hyperbolic *segment* as shown in Figure 17.7. In *Descarta2D* the parametric equations for the hyperbola are

$$x = a \cosh(st)$$
 and  $y = b \sinh(st)$ 

where a and b are the lengths of the semi-transverse and semi-conjugate axes, respectively,  $s = \cosh^{-1}(e)$ , and e is the eccentricity of the hyperbola. The exploration **areahyp.nb** uses calculus to derive the formula for the area of the segment, which is given by

$$A_{segment} = \frac{ab}{2}(\sinh(s(t_2 - t_1)) - s(t_2 - t_1)).$$

Interestingly, the area does not depend on the values of  $t_1$  and  $t_2$  directly, but only upon the difference between the two parameters.

Since we know the coordinates of the vertex points of the triangle  $OP_1P_2$  we can compute its area directly as

$$A_{\triangle} = \frac{ab}{2}\sinh(s(t_2 - t_1)).$$

The area of a hyperbolic *sector* is the difference between the area of the triangle  $OP_1P_2$ and the hyperbolic segment as illustrated in Figure 17.7. The area of the hyperbolic sector is given by

 $A_{sector} = A_{\triangle} - A_{segment}$  $= \frac{abs}{2}(t_2 - t_1).$ 

**Example.** Find the area of the hyperbolic segment between parameters  $t_1 = -2$  and  $t_2 = 1$  for a hyperbola centered at the origin with a = 1 and b = 1/2 in standard position. Also, find the area of the associated hyperbolic sector.

**Solution.** The Descarta<sub>2D</sub> function SegmentArea<sub>2D</sub>[hyperbola, { $t_1$ ,  $t_2$ }] returns the area of a segment of a hyperbola defined by two parameters. The function SectorArea<sub>2D</sub>[hyperbola, { $t_1$ ,  $t_2$ }] returns the area of the associated hyperbolic sector.

```
In[8]: h1 = Hyperbola2D[{0, 0}, 1, 1/2, 0];
        {SegmentArea2D[h1, {-2, 1}],
        SectorArea2D[h1, {-2, 1}]} // N
Out[8] {0.139091, 0.360909}
```

# 17.6 Parabolic Areas

Consider a parabola in standard position with vertex at (0,0), axis parallel to the x-axis, focal length of f, and opening to the left as shown in Figure 17.8. Such a parabola can be represented with the set of parametric equations

$$x = ft^2$$
 and  $y = 2ft$ .

The area of the chordal area defined by the parameters  $t_1$  and  $t_2$  can be determined by subtracting the area between the parabola and the *y*-axis from the area between the chord and the *y*-axis. The end points of the chord are  $(ft_1^2, 2ft_1)$  and  $(ft_2^2, 2ft_2)$ , and the line through these two points is given by

$$x = \frac{(t_1 + t_2)y - 2ft_1t_2}{2}.$$

The appropriate integral is then given by

$$A = \int_{y_1}^{y_2} (x_2 - x_1) \, dy$$
  
=  $\int_{y_1}^{y_2} \left( \frac{(t_1 + t_2)y - 2ft_1t_2}{2} - \frac{y^2}{4f} \right) \, dy.$ 



Figure 17.8: Area of a parabolic segment.

Performing the integration and making the substitutions  $y_1 = 2ft_1$  and  $y_2 = 2ft_2$  yields the formula for the area, A, of a parabolic segment to be

$$A = \frac{f^2(t_2 - t_1)^3}{3}$$

where  $t_1$  and  $t_2$  are the parameters of the end points of the chord defining the segment, and  $t_1 < t_2$  for positive areas. A parabola has no sector area definition because a parabola does not have a center point.

**Example.** Find the area between the parabola  $y^2 = x/2$  rotated  $\pi/2$  radians about its vertex and its focal chord.

**Solution.** The function SegmentArea2D[*parabola*,  $\{t_1, t_2\}$ ] returns the area of a parabolic segment defined by parameters  $t_1$  and  $t_2$ . In Descarta2D the end points of the focal chord occur at parameter values  $t_1 = -1$  and  $t_2 = 1$ .

```
In[9]: p1 = Parabola2D[{0, 0}, 1/2, Pi/2];
SegmentArea2D[p1, {-1, 1}]
Out[9] 2/3
```



F

**Descarta2D Hint.** Notice that the parabola's position and orientation have no bearing on the area of a parabolic segment. The area depends solely on the focal length and the chord position.

#### 17.7 Conic Arc Area

The area between a conic arc and its chord can also be computed by summing infinitesimal rectangles through the use of calculus. Consider, for example, a conic arc whose start point is (0,0), end point is (d,0), apex point is (a,b) and projective discriminant is  $\rho$ . Intuitively, since the chord of this conic arc is coincident with the x-axis we can imagine subdividing the area of the conic arc into a large number of horizontal rectangles of very small height. By summing the areas of these small rectangles we can provide an approximation to the area of the conic arc. The methods of integral calculus accomplish this summing process, and in the limit as the height of the rectangles approaches zero, the area approaches the true area of the conic arc segment. The details of this process are captured in the exploration caarea1.nb. The area of the conic arc segment is found to be

$$A = \frac{bd\rho}{2r^3} \left( \rho r + (-1+\rho)^2 \log_e \left( \frac{1-\rho}{\rho+r} \right) \right)$$

where  $r = \sqrt{-1 + 2\rho}$  (assuming b > 0 and d > 0). Notice that the abscissa, a, of the apex point has no bearing on the area of the segment bounded by the conic arc and its chord.

If the conic arc is a parabola, then  $\rho = \frac{1}{2}$  and the formula for the area given above is invalid. The formula for a parabola is much simpler and is given by

$$A = \frac{bd}{3}$$

as shown in the exploration caarea2.nb.

This process can be generalized to find the segment area of any conic arc. Notice that the position of the conic arc in the plane has no bearing on its chordal area. Therefore, we can translate the start point to (0,0) and rotate the conic so that the end point is on the x-axis.

**Example.** Find the area between the conic arc with start point (1, 2), end point (5, 0), apex point (3, 4) and  $\rho = 0.75$  and its chord.

**Solution.** The Descarta2D function Area2D[cnarc] computes the area between a conic arc and its chord.

In[10]: Area2D[ConicArc2D[{1, 2}, {3, 4}, {5, 0}, 0.75]] // N
Out[10] 5.34774

| Object    | Area2D | SectorArea2D | SegmentArea2D |
|-----------|--------|--------------|---------------|
| arc       | yes    | no           | no            |
| circle    | yes    | yes          | yes           |
| conic arc | yes    | no           | no            |
| ellipse   | yes    | yes          | yes           |
| hyperbola | no     | yes          | yes           |
| parabola  | no     | no           | yes           |
| triangle  | yes    | no           | no            |

Table 17.1: Descarta2D area functions.

# 17.8 Summary of Area Functions

Table 17.1 summarizes the area functions provided by Descarta2D.

Area2D[*object*] returns the area enclosed by a closed object (circle, ellipse or triangle). SectorArea2D[*object*,  $\{t_1, t_2\}$ ] returns the area of a sector defined by two parameters.

SegmentArea2D[object, { $t_1$ ,  $t_2$ }] returns the area between a chord and the curve.

# 17.9 Explorations

HERON'S FORMULA.....heron.nb Show that the area, K, of a triangle  $\triangle ABC$  is given by

$$K = \sqrt{s(s-a)(s-b)(s-c)},$$

where the semi-perimeter s = (a + b + c)/2, and a, b and c are the lengths of the sides.

AREA OF TRIANGLE CONFIGURATIONS.....triarea.nb



For the triangle illustrated in the figure, show that the area,  $A_1$ , associated with the AAS (angle-angle-side) configuration whose parameters are  $a_1$ ,  $a_2$  and  $s_1$  is given by

$$A_1 = \frac{s_1^2 \sin(a_2) \sin(a_1 + a_2)}{2 \sin(a_1)}.$$

Show that the area,  $A_2$ , associated with the ASA (angle-side-angle) configuration whose parameters are  $a_1$ ,  $s_3$  and  $a_2$  is given by

$$A_2 = \frac{s_3^2 \sin(a_1) \sin(a_2)}{2 \sin(a_1 + a_2)}.$$

Show that the area,  $A_3$ , associated with the SAS (side-angle-side) configuration whose parameters are  $s_1$ ,  $a_2$  and  $s_3$  is given by

$$A_3 = \frac{s_1 s_3 \sin(a_2)}{2}.$$

AREA OF TRIANGLE BOUNDED BY LINES.....triarlns.nb Show that the area of the triangle bounded by the lines

$$y = m_1 x + c_1$$
,  $y = m_2 x + c_2$  and  $x = 0$ 

is given by

$$A = \frac{1}{2} \frac{(c_1 - c_2)^2}{\sqrt{(m_1 - m_2)^2}}.$$

AREAS RELATED TO HYPERBOLAS.....hyparea.nb

Referring to Figure 17.7, use calculus to verify that the areas between two parameters  $t_1$  and  $t_2$  of a segment and a sector of a hyperbola are given by

$$A_{segment} = \frac{ab}{2}(\sinh(s(t_2 - t_1)) - s(t_2 - t_1))$$
$$A_{sector} = \frac{abs}{2}(t_2 - t_1)$$

where a and b are the lengths of the semi-transverse and semi-conjugate axes, respectively,  $s = \cosh^{-1}(e)$ , and e is the eccentricity of the hyperbola (assuming the parameterization Descarta<sub>2D</sub> uses for a hyperbola).

Area of a Conic Arc (General).....caarea1.nb

For the conic arc whose control points are (0,0), (a,b) and (d,0), show that the area between the conic arc and its chord is given by

$$A = \frac{bd\rho}{2r^3} \left( \rho r + (-1+\rho)^2 \log_e \left( \frac{1-\rho}{\rho+r} \right) \right)$$

where  $r = \sqrt{-1 + 2\rho}$  (assuming b > 0 and d > 0).

Area of a Conic Arc (Parabola).....caarea2.nb

Show that the area between a conic arc whose projective discriminant is  $\rho = \frac{1}{2}$  and its chord is given by

$$A = \frac{bd}{3}$$

when the control points are (0,0), (a,b) and (d,0).

ONE-THIRD OF A CIRCLE'S AREA.....circarea.nb

Show that the angle,  $\theta$ , subtended by a segment of a circle whose area is one-third the area of the full circle is the root of the equation

$$\frac{\pi}{3} = \frac{\theta - \sin\theta}{2}.$$

Also, show that  $\theta$  is within 1/2 percent of  $5\pi/6$  radians.

EQUAL AREAS POINT ......eqarea.nb

Given  $\triangle ABC$  with vertices  $A(x_A, y_A)$ ,  $B(x_B, y_B)$  and  $C(x_C, y_C)$  show that there are four positions of a point  $P_n(x, y)$  such that  $\triangle APB$ ,  $\triangle APC$  and  $\triangle BPC$  have equal areas. The coordinates of  $P_n$  are given by

$$P_0(\frac{1}{3}(x_A + x_B + x_C), \frac{1}{3}(y_A + y_B + y_C))$$

$$P_1(-x_A + x_B + x_C, -y_A + y_B + y_C)$$

$$P_2(+x_A - x_B + x_C, +y_A - y_B + y_C)$$

$$P_3(+x_A + x_B - x_C, +y_A + y_B - y_C).$$

 $P_0$  is the centroid of  $\triangle ABC$  and  $\triangle P_1P_2P_3$ .  $\triangle ABC$  connects the midpoints of the sides of  $\triangle P_1P_2P_3$ .

AREA OF A TETRAHEDRON'S BASE ......tetra.nb

A tetrahedron is a three-dimensional geometric object bounded by four triangular faces. Given a tetrahedron with vertices O(0,0,0), A(a,0,0), B(0,b,0) and C(0,0,c) show that the areas of the triangular faces are related by the equation

$$(A_{ABC})^{2} = (A_{AOB})^{2} + (A_{AOC})^{2} + (A_{BOC})^{2}$$

where  $A_{xyz}$  is the area of the triangle whose vertices are x, y and z. Note the similarity to the Pythagorean Theorem for right triangles.

Part V Tangent Curves

# Chapter 18

# **Tangent Lines**

Let two points, P and Q, be taken on any locally smooth convex curve, and let the point Q move along the curve nearer and nearer to the point P; then the limiting position of the line PQ, as Q moves up to and ultimately coincides with P, is called the *tangent line* to the curve at point P. The line through P perpendicular to the tangent line is called the *normal* to the curve at the point P.

# 18.1 Lines Tangent to a Circle

#### Tangent Through a Point On a Circle

Let  $(x - h)^2 + (y - k)^2 = r^2$  be a circle and  $P_1(x_1, y_1)$  a point on it as shown in Figure 18.1. We desire to find the equation of the tangent line at  $P_1$ . Since the slope of the line joining the center (h, k) and  $P_1$  is  $(y_1 - k)/(x_1 - h)$ , the slope of the line tangent to the circle will be the negative reciprocal  $-(x_1 - h)/(y_1 - k)$  and the equation of the line tangent to the circle at point  $P_1$  becomes (point-slope form)

$$y - y_1 = -\frac{(x_1 - h)}{(y_1 - k)}(x - x_1).$$
(18.1)

Since the point  $P_1$  is on the circle we also have the equation

$$(x_1 - h)^2 + (y_1 - k)^2 = r^2. (18.2)$$

Adding Equation (18.1) to Equation (18.2) results in

$$(x_1 - h)x + (y_1 - k)y + (h^2 + k^2 - r^2 - hx_1 - ky_1) = 0$$

or, in a factored form that is easier to remember,

$$(x-h)(x_1-h) + (y-k)(y_1-k) = r^2.$$
(18.3)



Figure 18.1: Line tangent to a circle.

If the circle is centered at the origin,  $x^2 + y^2 = r^2$ , the equation of the tangent line at  $P_1$  is  $x_1x + y_1y = r^2$ .

If the circle is given in general form,

$$x^2 + y^2 + ax + by + c = 0$$

then the tangent line at  $P_1$  is

F

F

$$xx_1 + yy_1 + \frac{a}{2}(x+x_1) + \frac{b}{2}(y+y_1) + c = 0.$$

**Example.** Confirm that the point  $(\frac{5}{2}, 1 + \frac{\sqrt{3}}{2})$  is on the circle

$$(x-2)^2 + (y-1)^2 = 1$$

and find the tangent line at that point.

**Solution.** The function IsOn2D[*point*, *circle*] returns True if the point is on the circle; otherwise, it returns False. TangentLines2D[*point*, *circle*] returns a list of lines through the point and tangent to the circle.



Figure 18.2: Lines through a point, tangent to a circle at the origin.

In[3]: Sketch2D[{p1, c1, lns}, PlotRange -> {{0, 4}, {-1, 3}}];



## Tangents Through a Point Outside a Circle

If the point  $P_1(x_1, y_1)$  is outside of the circle  $(x - h)^2 + (y - k)^2 = r^2$  there will be two tangent lines from  $P_1$  to the circle. Consider the circle  $x^2 + y^2 = r^2$  and the point D(d, 0) in a convenient position as shown in Figure 18.2. Clearly, the two tangent lines can be determined directly from the normal form of a line as

$$x\cos\theta + y\sin\theta - r = 0$$

where

$$\cos \theta = \frac{r}{d}$$
 and  $\sin \theta = \pm \frac{\sqrt{d^2 - r^2}}{d}$ .

If the point D is rotated by an angle  $\alpha$  about the origin, as shown in Figure 18.3, it will have new coordinates  $P_0(x_0, y_0)$  and the tangent lines will also be rotated by  $\alpha$  resulting in the two lines

$$x\cos(\alpha + \theta) + y\sin(\alpha + \theta) - r = 0$$



Figure 18.3: Lines through a rotated point, tangent to a circle at the origin.

where

$$\cos \alpha = \frac{x_0}{d}$$
 and  $\sin \alpha = \frac{y_0}{d}$ .

Using the standard trigonometric formulas for the sine and cosine of the sum of two angles yields

$$\begin{aligned} \cos(\alpha + \theta) &= \cos \alpha \cos \theta - \sin \alpha \sin \theta \\ &= \frac{x_0}{d} \frac{r}{d} - \frac{y_0}{d} \frac{\pm \sqrt{d^2 - r^2}}{d} \\ &= \frac{1}{d^2} \left( x_0 r \mp y_0 \sqrt{d^2 - r^2} \right) \end{aligned}$$

and

$$\sin(\alpha + \theta) = \cos \alpha \sin \theta + \cos \theta \sin \alpha$$
$$= \frac{x_0}{d} \frac{\pm \sqrt{d^2 - r^2}}{d} + \frac{r}{d} \frac{y_0}{d}$$
$$= \frac{1}{d^2} \left( y_0 r \pm x_0 \sqrt{d^2 - r^2} \right)$$

As a final adjustment we translate the geometry so that the center of the circle may be a general location (h, k) as shown in Figure 18.4. Translating the tangent lines from (0, 0) to (h, k) using  $x_0 = x_1 + h$  and  $y_0 = y_1 + k$  yields

$$x\cos(\alpha+\theta) + y\sin(\alpha+\theta) - (r + h\cos(\alpha+\theta) + k\sin(\alpha+\theta)) = 0$$

where  $\cos(\alpha + \theta)$  and  $\sin(\alpha + \theta)$  are functions of  $(x_1, y_1)$ , (h, k), and r and d is the distance between (h, k) and  $(x_1, y_1)$ . Notice that after the substitutions are performed no trigonometric functions are present in the formulas.



Figure 18.4: Lines through a general point, tangent to any circle.

#### **Tangent Contact Points**

Given a circle  $(x - h)^2 + (y - k)^2 = r^2$  and a point  $P_1(x_1, y_1)$  outside the circle, as shown in Figure 18.4, we desire the coordinates of the points of contact between the circle and the two tangent lines. From the previous section it is clear that when the geometry is in the standard position the coordinates of the contact points are given by

$$Q_{1,2}(r\cos\theta, r\sin\theta)$$

where

$$\cos \theta = \frac{r}{d}$$
 and  $\sin \theta = \pm \frac{\sqrt{d^2 - r^2}}{d}$ .

If the geometry is rotated and translated to a general position the coordinates of the contact points are given by

 $Q_{1,2}\left(h+r\cos(\alpha+\theta),k+r\sin(\alpha+\theta)\right)$ 

where  $\cos(\alpha + \theta)$  and  $\sin(\alpha + \theta)$  have the same formulas as in the previous section.

**Example.** Find the lines passing through the point (3, -1) and tangent to the circle  $(x + 1)^2 + (y - 1)^2 = 4$ . Find the coordinates of the points of tangency and plot.

**Solution.** The function TangentLines2D[*point*, *circle*] returns a list of lines through the point and tangent to the circle. TangentPoints2D[*point*, *circle*] returns a list of the points of tangency.

```
In[4]: pl = Point2D[{3, -1}]; cl = Circle2D[{-1, 1}, 2];
objs = {TangentLines2D[pl, cl], TangentPoints2D[pl, cl]}
Out[4] {{Line2D[0, -20, -20], Line2D[16, 12, -36]},
{Point2D[{-1, -1}], Point2D[{3/5, 11/5}]}}
```

```
In[5]: Sketch2D[{p1, c1, objs}];
```



#### **Tangent Line Segment Length**

1

To find the length of the tangent line segment drawn from a given point,  $P_1(x_1, y_1)$ , to a circle  $(x-h)^2 + (y-k)^2 = r^2$  without computing the point of tangency, the following method can be used. Since  $\triangle OP_1P$  in Figure 18.2 is a right triangle the distance D between P and  $P_1$  is given by

$$D^{2} = d^{2} - r^{2}$$
  
=  $(x_{1} - h)^{2} + (y_{1} - k)^{2} - r^{2}$ .

Therefore the length of the tangent line segment (squared) is found by substituting the coordinates of the point into the equation of the circle.

**Example.** Find the length of the tangent line segment from the point (4,3) to the circle  $(x+1)^2 + (y+2)^2 = 4$ .

**Solution.** Descarta2D does not have a built-in function to compute the length of a tangent line segment. However, a few built-in functions can be combined to apply the technique described in this section. The function Quadratic2D[circle] returns the quadratic equation of a circle. Polynomial2D[quad,  $\{x, y\}$ ] substitutes the coordinates (x, y) into the quadratic equation.



Figure 18.5: Lines tangent to a circle.

```
In[6]: c1 = Circle2D[{-1, -2}, 2];
    Sqrt[Polynomial2D[Quadratic2D[c1], {4, 3}]]
Out[6] \sqrt{46}
```

Of course this gives the same result as constructing the tangent points and finding the distance directly.

#### Tangents Parallel to a Line

The equations of the two lines parallel to the line  $L \equiv Ax + By + C = 0$  and tangent to the circle  $(x - h)^2 + (y - k)^2 = r^2$  as shown in Figure 18.5 are given by

$$L' \equiv Ax + By - \left(Ah + Bk \pm r\sqrt{A^2 + B^2}\right) = 0.$$

Notice that the constant coefficient C of the line is not involved in the equations of the tangent lines since only the slope is involved in establishing the parallel condition. The formula is derived by constructing a line  $L_c$  that passes through the center (h, k) of the circle with slope m = -A/B. The two tangent lines are then determined by offsetting  $L_c$  a distance  $\pm r$ . Using equations the derivation is

$$L \equiv Ax + By + C = 0$$

$$L_c \equiv Ax + By - (Ah + Bk) = 0$$
  

$$ax + by - (ah + bk) = 0$$
  

$$L' \equiv ax + by - (ah + bk \pm r) = 0$$
  

$$Ax + By - (Ah + Bk \pm r\sqrt{A^2 + B^2}) = 0.$$

where

$$a = \frac{A}{\sqrt{A^2 + B^2}}$$
 and  $b = \frac{b}{\sqrt{A^2 + B^2}}$ 

are the normalized coefficients of the line.

#### **Tangents Perpendicular to a Line**

To find the equations of the two lines, L'', perpendicular to the line

$$Ax + By + C = 0$$

and tangent to the circle

$$(x-h)^2 + (y-k)^2 = r^2$$

as shown in Figure 18.5, simply use the line Bx - Ay + C = 0 (which is perpendicular to the given line) and apply the formula from the previous section. Once again the value of the constant coefficient C has no bearing on the equations of the resulting lines.

**Example.** Find the lines tangent to the circle  $(x-3)^2 + (y-2)^2 = 1$  and parallel and perpendicular to the line 2x + 3y - 1 = 0 and plot.

**Solution.** The Descarta2D function TangentLines2D[*line*, *circle*, Parallel2D] returns a list of lines parallel to the line and tangent to the circle. The function TangentLines2D[*line*, *circle*, Perpendicular2D] returns a list of lines perpendicular to the line and tangent to the circle.

```
In[8]: l1 = Line2D[2, 3, -1]; c1 = Circle2D[{3, 2}, 1];
Ins = {TangentLines2D[11, c1, Parallel2D],
TangentLines2D[11, c1, Perpendicular2D]}
Out[8] {{Line2D[2, 3, -12 - \sqrt{13}], Line2D[2, 3, -12 + \sqrt{13}]},
{Line2D[-3, 2, 5 - \sqrt{13}], Line2D[-3, 2, 5 + \sqrt{13}]}}
```

In[9]: Sketch2D[{l1, c1, lns}, PlotRange -> {{-2, 5}, {-2, 5}}];



F

F

Descarta2D Hint. TangentLines2D[line, circle] returns the same result as

TangentLines2D[line, circle, Parallel2D]

because specifying the keyword Parallel2D is optional.

**Example.** Using the geometric objects from the previous example, find the points of contact of the four tangent lines.

**Solution.** The function Point2D[*line*, *circle*] will return the point of contact if the line is tangent to the circle.

 $\begin{array}{l} \textit{Out[10]} & \left\{\texttt{Point2D}\left[\left\{3+\frac{2}{\sqrt{13}},\ 2+\frac{3}{\sqrt{13}}\right\}\right],\ \texttt{Point2D}\left[\left\{3-\frac{2}{\sqrt{13}},\ 2-\frac{3}{\sqrt{13}}\right\}\right],\\ & \texttt{Point2D}\left[\left\{3-\frac{3}{\sqrt{13}},\ 2+\frac{2}{\sqrt{13}}\right\}\right],\ \texttt{Point2D}\left[\left\{3+\frac{3}{\sqrt{13}},\ 2-\frac{2}{\sqrt{13}}\right\}\right]\right\} \end{array}$ 



-



Figure 18.6: Lines tangent to two circles.

#### **Tangents to Two Circles**

Suppose  $C_1$  and  $C_2$  are two circles and we wish to determine the equations of the lines tangent to both circles. We proceed by finding the equations of tangent lines when the circles are in a special position and then we transform the result to a general position.

Let  $C_1$  be a circle, with radius  $r_1$ , centered at the origin with equation  $x^2 + y^2 = r_1^2$ , and let  $C_2$ , with radius  $r_2$ , be positioned so that its center is on the +x-axis a distance d from the origin with equation  $(x - d)^2 + y^2 = r_2^2$ . Since  $C_1$  is centered at the origin any line tangent to  $C_1$  can be written in the form  $L \equiv Ax + By + 1 = 0$  because no line tangent to  $C_1$  can pass through the origin.

Let  $d_1$  and  $d_2$  be the distances from the center of  $C_1$  and  $C_2$  to L, respectively. If L is tangent to the circles then  $d_1$  and  $d_2$  must equal the radii of the circles, yielding

$$r_n = \frac{Ah_n + Bk_n + 1}{\sqrt{A^2 + B^2}}$$
$$r_n^2 = \frac{(Ah_n + Bk_n + 1)^2}{A^2 + B^2}$$
$$r_n^2(A^2 + B^2) = (Ah_n + Bk_n + 1)$$

where  $h_1 = 0$ ,  $k_1 = 0$ ,  $h_2 = d$  and  $k_2 = 0$ . Simplifying, we have the two equations

$$r_1^2(A^2 + B^2) = 1 r_2^2(A^2 + B^2) = (1 + Ad)^2.$$

Solving these two equations for A and B produces four pairs of solutions given by

$$A = r_1 + s_A r_2 B = s_B \sqrt{d^2 - (r_1 + s_A r_2)^2}$$

where the sign constants  $s_A = \{-1, -1, 1, 1\}$  and  $s_B = \{1, -1, 1, -1\}$  take on the values  $\pm 1$  in pairs as shown in the lists. The first pair of solutions gives the *external*, or *direct*, tangents and the second pair gives the *internal*, or *transverse*, tangents.

We now use the special solution given above to find the tangent lines when the circles are in a general position. Let  $C_1 \equiv (x - h_1)^2 + (y - k_1)^2 = r_1^2$  and  $C_2 \equiv (x - h_2)^2 + (y - k_2)^2 = r_2^2$ be the equations of the two circles. To attain a general positioning we first rotate the lines given in the special solution by an angle  $\theta$  where  $\sin \theta = (h_1 - h_2)/d$  and  $\cos \theta = (k_1 - k_2)/d$ . After the rotation we translate the lines from (0,0) to  $(h_1, k_1)$ . Applying these transformations yields the four lines

$$(AH - BK)x + (BH + AK)y + d^{2}r_{1} - h_{1}(AH - BK) - k_{1}(BH + AK) = 0$$

where

E

$$H = h_1 - h_2$$
 and  $K = k_1 - k_2$ 

and A and B take the values given as before.

**Example.** Find the four lines tangent to the circles  $(x - 3)^2 + y^2 = 4$  and  $(x + 3)^2 + y^2 = 4$ . Sketch the external tangents and the internal tangents in separate plots.

**Solution.** The Descarta2D function TangentLines2D[circle, circle] returns a list of lines tangent to two circles. The first two lines in the list are the external tangents (if returned); the third and fourth lines in the list (if returned) are the internal tangents.

```
In[13]: Sketch2D[{c1, c2, l1, l2}, PlotRange -> {{-6, 6}, {-3, 3}}];
Sketch2D[{c1, c2, l3, l4}, PlotRange -> {{-6, 6}, {-3, 3}}];
```





# 18.2 Lines Tangent to Conics

#### **Tangent Through Point on Conic**

Suppose we have the general equation of a conic curve given by

$$ax^{2} + bxy + cy^{2} + dx + ey + f = 0$$

The equation of the chord joining any two points,  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$ , on the curve may be written

$$a(x - x_1)(x - x_2) + b(x - x_1)(y - y_2) + c(y - y_1)(y - y_2)$$
  
=  $ax^2 + bxy + cy^2 + dx + ey + f$ 

as the equation is clearly first-degree in x and y (the terms above first-degree cancel out), and it is satisfied by the two points  $P_1$  and  $P_2$ . Putting  $x_1 = x_2$  and  $y_1 = y_2$ , we get the equation of the tangent line

$$a(x - x_1)^2 + b(x - x_1)(y - y_1) + c(y - y_1)^2 = ax^2 + bxy + cy^2 + dx + ey + f;$$

or, expanding,

$$2ax_1x + b(x_1y + y_1x) + 2cy_1y + dx + ey + f = ax_1^2 + bx_1y_1 + cy_1^2.$$

Adding  $dx_1 + ey_1 + f_1$  to both sides will cause the right-hand side to vanish, because  $P_1$  satisfies the equation of the curve. Thus the equation of the tangent becomes

$$ax_1x + \frac{b}{2}(x_1y + y_1x) + cy_1y + \frac{d}{2}(x + x_1) + \frac{e}{2}(y + y_1) + f = 0.$$
(18.4)

This equation is most easily remembered if we compare it with the equation of the curve and notice that it is derived by replacing  $x^2$  and  $y^2$  with  $x_1x$  and  $y_1y$ , xy with  $\frac{1}{2}(x_1y + y_1x)$ and x and y with  $\frac{1}{2}(x+x_1)$  and  $\frac{1}{2}(y+y_1)$ . Whether or not  $P_1(x_1, y_1)$  is on the curve, the line represented by Equation (18.4) is called the *polar* of  $P_1$  with respect to the curve, and  $P_1$  is the *pole* of the line with respect to the curve. F

F

**Example.** Find the line tangent to the parabola  $y^2 = 4x$  at the point (4, 4).

**Solution.** The Descarta2D function Line2D[point, conic] returns the polar (line) of a pole (point) with respect to a conic. If the point is on the conic, then the line will be tangent to the conic.

#### Pole Point and Point of Tangency

Given a line  $L \equiv px + qy + r = 0$  and a conic

$$Q \equiv Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

we wish to determine the coordinates of the pole point,  $P_1(x_1, y_1)$ , of L with respect to Q. The equation of the polar (line) of the pole (point)  $P_1$  is derived in general form from Equation (18.4) and is given by

$$(2Ax_1 + By_1 + D)x + (Bx_1 + 2Cy_1 + E)y + (Dx_1 + Ey_1 + 2F) = 0$$

If this line and line L are the same line, then the coefficients of the polar line must be equal to some multiple of the coefficients of L yielding the following system of three linear equations in three unknowns

$$kp = 2Ax_1 + By_1 + D$$
  

$$kq = Bx_1 + 2Cy_1 + E$$
  

$$kr = Dx_1 + Ey_1 + 2F.$$

-

Solving theses equations for  $(x_1, y_1)$  and the constant k (k is ignored) gives

$$x_1 = \frac{Q_1}{Q_{12}}$$
 and  $y_1 = \frac{Q_2}{Q_{12}}$ 

where

E

-----

$$Q_{1} = p(4CF - E^{2}) + q(DE - 2BF) + r(BE - 2CD)$$
  

$$Q_{2} = p(DE - 2BF) + q(4AF - D^{2}) + r(BD - 2AE)$$
  

$$Q_{12} = p(BE - 2CD) + q(BD - 2AE) + r(4AC - B^{2}).$$

If the line L is tangent to Q, then  $(x_1, y_1)$  is the point of tangency; otherwise,  $(x_1, y_1)$  is the pole of the polar L with respect to Q. If  $Q_{12}$  is zero, the coordinates of the pole are invalid. This condition occurs when the polar L passes through the center of the conic.

**Example.** Show that the polar (line) found in the previous example corresponds to the pole (point) of tangency.

**Solution.** The Descarta2D function Point2D[*line*, *conic*] returns the pole (point) of a polar (line) with respect to a conic.

In[16]: Point2D[11, crv]
Out[16] Point2D[{4, 4}]

#### Line Tangent to a Conic Condition

To find the relationship between the coefficients of a line px + qy + r = 0 and a general conic  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$  such that the line is tangent to the conic we note that the two intersection points between the line and the conic must be coincident. This condition is equivalent to

$$(4CF - E^2)p^2 + (4AF - D^2)q^2 + (4AC - B^2)r^2 + 2(BD - 2AE)qr + 2(BE - 2CD)pr + 2(DE - 2BF)pq = 0.$$

The exploration lntancon.nb derives this equation.

**Example.** Find the value of c such that the line 2x + 5y + c = 0 is tangent to the conic represented by  $2x + xy - 4y^2 - 2x - 3y + 1 = 0$  and plot.

**Solution.** The Descarta2D function TangentEquation2D[line, quad] returns an equation establishing the condition that a line be tangent to the conic represented by the quadratic. Solve this equation for the unknown coefficient c.

```
In [17]: Clear[c];

11 = Line2D[2, 5, c];

q1 = Quadratic2D[2, 1, -4, -2, -3, 1];

eq1 = TangentEquation2D[11, q1]

Out [17] 80 + 24 c - 33 c<sup>2</sup> == 0

In [18]: ans = Solve[eq1, c]

Out [18] {{c \rightarrow \frac{4}{33} (3 - \sqrt{174})}, {c <math>\rightarrow \frac{4}{33} (3 + \sqrt{174})}}

In [19]: Sketch2D[{Map[(11 /. #)&, ans], Loci2D[q1]},

PlotRange -> {{-3, 3}, {-2, 2}}];

1.5

0.5

0
```

#### Polar of a Conic

As previously shown, if the point  $P_1(x_1, y_1)$  is on the conic curve

-0.5 -1 -1.5

-2 -1

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

2

3

0 1

the equation of the tangent line at  $P_1$  is

$$2Axx_1 + B(xy_1 + x_1y) + 2Cyy_1 + D(x + x_1) + E(y + y_1) + 2F = 0.$$

This equation expresses a relation between the coordinates (x, y) of any point on the tangent line, and those of the point of contact  $(x_1, y_1)$ . But the equation, being symmetrical with respect to the coordinates (x, y) and  $(x_1, y_1)$ , can be interpreted to represent the line passing through the points of contact from  $(x_1, y_1)$  when  $(x_1, y_1)$  is not on the curve. Thus the polar, which has the same equation as the tangent line, passes through the points of tangency (when they are real) when the point is not on the curve.

Without proof we list the following theorems concerning poles and polars that refer to Figure 18.7.



Figure 18.7: Poles and polars.

- If the polar  $L_1$  of pole  $P_1$  passes through pole  $P_2$ , then the polar  $L_2$  of  $P_2$  passes through  $P_1$ .
- If the polars of  $P_1$  and  $P_2$  intersect at point P, then P is the pole of the line  $P_1P_2$ .
- The polar of an exterior point  $P_1$  is the line joining the points of contact of the tangents drawn from  $P_1$ .
- The polar of an interior point P is the locus of the point of intersection of the tangents at the extremities of every chord through P.
- The polar of a focus is the corresponding directrix.
- There is no (finite) polar of the center of a conic.

**Example.** Show the inverse functional relationship between the polar and the pole (3, -1) with respect to the quadratic equation  $2x^2 + 3xy - y^2 + 4x - 2y + 1 = 0$ .

**Solution.** The Descarta2D function Line2D[point, quad] returns the polar line of the point with respect to the quadratic. The function Point2D[line, quad] returns the pole (point) of the line with respect to the quadratic.

F

L----
## Tangents Parallel to a Line

Once again, consider the conic curve whose equation is

$$Q \equiv Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0,$$

and the equation of the tangent line at the point  $P_1(x_1, y_1)$  on the conic whose equation is

$$2Axx_1 + B(xy_1 + x_1y) + 2Cyy_1 + D(x + x_1) + E(y + y_1) + 2F = 0$$

or, in general form,

$$(2Ax_1 + By_1 + D)x + (Bx_1 + 2Cy_1 + E)y + (Dx_1 + Ey_1 + 2F) = 0.$$

To find the lines tangent to Q and parallel to a line  $L_1 \equiv A_1x + B_1y + C_1 = 0$  the following technique can be used. Let  $L_2 \equiv A_1x + B_1y + C_2 = 0$  be the desired tangent line (the linear coefficients  $A_1$  and  $B_1$  of  $L_2$  are set equal to the corresponding coefficients of  $L_1$  because the lines are parallel). If  $L_2$  is to be tangent to Q, then the pole point P of  $L_2$  with respect to Q must satisfy the equation for Q. The coordinates of P are functions in the variable  $C_2$ , therefore, solving this equation for  $C_2$  gives the coefficients of the desired tangent line(s)  $L_2$ . The Descarta2D function TangentLines2D[line, quad] implements this technique and can be used to derive the specialized formulas for lines tangent to conics in standard position as presented later in this chapter.

## Lines Tangent to Two Conics

The equation relating the coefficients of a quadratic equation

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

to the coefficients of a line px + qy + r = 0 tangent to the quadratic given previously is

$$(4CF - E^2)p^2 + (4AF - D^2)q^2 + (4AC - B^2)r^2 + 2(BD - 2AE)qr + 2(BE - 2CD)pr + 2(DE - 2BF)pq = 0.$$

If we select a suitable translation, we can insure that the tangent line does not pass through the origin (i.e.  $r \neq 0$ ) and the equation of the tangent line can be written in the form

$$\frac{p}{r}x + \frac{q}{r} + \frac{r}{r} = p'x + q'y + 1 = 0.$$

Now, given two quadratic equations

$$Q_1 \equiv A_1 x^2 + B_1 xy + C_1 y^2 + D_1 x + E_1 y + F_1 = 0 \text{ and}$$
  

$$Q_2 \equiv A_2 x^2 + B_2 xy + C_2 y^2 + D_2 x + E_2 y + F_2 = 0$$

and using Equation (18.2) we can find the coefficients p' and q' of the lines tangent to the quadratic by solving two quadratic equations in two unknowns, resulting in equations for four tangent lines. The formulas can be derived in symbolic form, but the results are too unwieldy to be of practical use. *Descarta2D*, however, can be used to construct such tangents when the problem involves numerical coefficients.

-

**Example.** Find the four lines tangent to the ellipses

$$\frac{x^2}{16} + \frac{y^2}{4} = 1$$
 and  $\frac{x^2}{4} + \frac{y^2}{9} = 1$ .

Plot the ellipses and the tangent lines.

**Solution.** The Descarta<sub>2D</sub> function TangentLines2D[curve, curve] returns a list of lines tangent to the two curves. The following result was computed using *Mathematica* Version 3.0.1. Version 4.0 produces lines in the same positions with slightly different coefficients.

```
In[22]: e1 = Ellipse2D[{0, 0}, 4, 2, 0];
e2 = Ellipse2D[{0, 0}, 3, 2, Pi/2];
Ins = TangentLines2D[e1, e2] // N
Out[22] {Line2D[-0.542326, -0.840168, -2.74398],
Line2D[-0.542326, 0.840168, -2.74398],
Line2D[0.542326, -0.840168, -2.74398], Line2D[0.542326, 0.840168, -2.74398]}
```

```
In[23]: Sketch2D[{e1, e2, lns}];
```



## Line Segments Tangent to Two Conics

Given a line tangent to a conic, the tangent point is the pole point of the line with respect to the conic. Using this relationship the line segments connecting the points of tangency can be determined as illustrated in the next example.

**Example.** Using the geometric objects from the previous example, find the line segments connecting the contact points of the tangent lines.

F

н

E

**Solution.** Use the function TangentSegments2D[*curve*, *curve*] to construct a list of line segments connecting the contact points of the lines tangent to the two curves. The following result was computed using *Mathematica* Version 3.0.1. Version 4.0 produces the same line segments, but in a different order.

```
In[24]: lnSegs = TangentSegments2D[e1, e2] // N
```

```
Out[24] {Segment2D[{-3.16228, -1.22474}, {-0.790569, -2.75568}],
Segment2D[{-3.16228, 1.22474}, {-0.790569, 2.75568}],
Segment2D[{3.16228, -1.22474}, {0.790569, -2.75568}],
Segment2D[{3.16228, 1.22474}, {0.790569, 2.75568}]}
```

```
In[25]: Sketch2D[{e1, e2, lnSegs}];
```



E

## 18.3 Lines Tangent to Standard Conics

Lines that are tangent to conics in standard position have particularly simple forms. This section summarizes the equations for these tangent lines for the parabola, ellipse and hyperbola.

#### Tangents to a Parabola

A line that is parallel to the axis of a parabola intersects the parabola in only one (finite) point; all other lines will cut the parabola in two points real and distinct, real and coincident, or complex. Any line which meets a parabola in two coincident points is a tangent line. Table 18.1 provides formulas for the line tangent to a parabola in standard form at a point  $P_1(x_1, y_1)$ . Table 18.2 provides the formulas for tangents to a parabola in standard form with a given slope m.

**Example.** Find the lines through the point (-1, -1) that are tangent to the parabola  $(y+1)^2 = 2(x-1)$  and plot.

| PARABOLA EQUATION   | Tangent at $P_1(x_1, y_1)$    |
|---------------------|-------------------------------|
| $y^2 = 4fx$         | $yy_1 = 2f(x+x_1)$            |
| $x^2 = 4fy$         | $xx_1 = 2f(y+y_1)$            |
| $(y-k)^2 = 4f(x-h)$ | $(y-k)(y_1-k) = 2f(x+x_1-2h)$ |
| $(x-h)^2 = 4f(y-k)$ | $(x-h)(x_1-h) = 2f(y+y_1-2k)$ |

Table 18.1: Tangents to a parabola at a point.

**Solution.** The function TangentLines2D[*point*, *curve*] returns a list of the lines through the point and tangent to the curve. The function Point2D[*line*, *curve*] will return the point of tangency for each tangent line.

```
In[26]: Clear[x, y];
pl = Point2D[{-1, -1}];
crv1 = First[Loci2D[Quadratic2D[(y+1)^2 == 2 (x - 1), {x, y}]]];
lns = TangentLines2D[p1, crv1]
```

```
Out[26] {Line2D[2, 4, 6], Line2D[-2, 4, 2]}
```

```
In[27]: Sketch2D[{pl, crv1, lns, Map[Point2D[#, crv1]&, lns]}];
```



### Tangents to an Ellipse

A line that intersects an ellipse in two coincident points is a tangent line. As in the case of the circle, but unlike that of the parabola, there will be two tangents with a given slope. Table 18.3 provides formulas for the line tangent to an ellipse in standard form at a point  $P_1(x_1, y_1)$ . In the formulas *a* is the length of the semi-major axis of the ellipse and *b* is the length of the semi-minor axis. Table 18.4 provides the formulas for tangents to an ellipse in standard form with a given slope *m*.

F

F

| PARABOLA EQUATION   | Tangent with Slope $m$    |  |
|---------------------|---------------------------|--|
| $y^2 = 4fx$         | y = mx + f/m              |  |
| $x^2 = 4fy$         | $y = mx - fm^2$           |  |
| $(y-k)^2 = 4f(x-h)$ | y - k = m(x - h) + f/m    |  |
| $(x-h)^2 = 4f(y-k)$ | $y - k = m(x - h) + fm^2$ |  |

Table 18.2: Tangents to a parabola with slope m.

**Example.** Find the lines tangent to the ellipse

$$\frac{(x-1)^2}{9} + y^2 = 1$$

rotated  $45^{\circ}$  counter-clockwise about its center point and passing through the point (4, -1) and plot.

**Solution.** The function TangentLines2D[*point*, *curve*] returns a list of lines through the point tangent to the curve. Point2D[*line*, *curve*] will return the tangent point of the line with respect to the curve.

```
In[28]: pl = Point2D[{4, -1}];
el = Ellipse2D[{1, 0}, 3, 1, 45 Degree] // N;
lns = TangentLines2D[pl, el] // N
```

```
Out[28] {Line2D[1.19166, -4.48728, -9.25392], Line2D[-2.94842, -0.782995, 11.0107]}
```

```
In[29]: Sketch2D[{p1, e1, lns, Map[Point2D[#, e1]&, lns]}];
```



| Ellipse Equation  | Tangent at $P_1(x_1,y_1)$                                 |
|---|---|
| $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$                 | $\frac{xx_1}{a^2} + \frac{yy_1}{b^2} = 1$                 |
| $\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$         | $\frac{(x-h)(x_1-h)}{a^2} + \frac{(y-k)(y-k_1)}{b^2} = 1$ |
| $\frac{x^2}{b^2} + \frac{y^2}{a^2} = 1$                 | $\frac{xx_1}{b^2} + \frac{yy_1}{a^2} = 1$                 |
| $\boxed{\frac{(x-h)^2}{b^2} + \frac{(y-k)^2}{a^2}} = 1$ | $\frac{(x-h)(x_1-h)}{b^2} + \frac{(y-k)(y-k_1)}{a^2} = 1$ |

Table 18.3: Tangents to an ellipse at a point.

Table 18.4: Tangents to an ellipse with slope m.

| Ellipse Equation  | Tangent with Slope $m$                      |
|---|---|
| $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$                 | $y = mx \pm \sqrt{a^2m^2 + b^2}$            |
| $\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$         | $y - k = m(x - h) \pm \sqrt{a^2 m^2 + b^2}$ |
| $\frac{x^2}{b^2} + \frac{y^2}{a^2} = 1$                 | $y = mx \pm \sqrt{b^2 m^2 + a^2}$           |
| $\boxed{\frac{(x-h)^2}{b^2} + \frac{(y-k)^2}{a^2}} = 1$ | $y - k = m(x - h) \pm \sqrt{b^2 m^2 + a^2}$ |

## Tangents to a Hyperbola

E

A line that intersects a hyperbola in two coincident points is a tangent line. For the hyperbola there will be two tangent lines (real and distinct, coincident with an asymptote, or complex) with a given slope. Table 18.5 provides formulas for the line tangent to a hyperbola in standard form at a point  $P_1(x_1, y_1)$ . In the formulas a is the length of the semi-transverse axis of the hyperbola and b is the length of the semi-conjugate axis. Table 18.6 provides the formulas for tangents to a hyperbola in standard form with a given slope m. Note that for real tangents with slope m the quantity under the radical must be positive. If, for a given slope, the tangents are real for a particular hyperbola, then the tangents are complex for the conjugate hyperbola.

**Example.** Find the lines tangent to the hyperbola

$$\frac{(x-1)^2}{9} - \frac{(y+1)^2}{4} = 1$$

rotated  $30^{\circ}$  counter-clockwise about its center point passing through the point (0,0) and plot.

**Solution.** The function TangentLines2D[*point*, *curve*] returns a list of lines through the point tangent to the curve. Point2D[*line*, *curve*] will return the tangent point of the line with respect to the curve.

```
In[30]: p1 = Point2D[{0, 0}];
h1 = Hyperbola2D[{1, -1}, 3, 2, 30 Degree] // N;
lns = TangentLines2D[p1, h1] // N
Out[30] {Line2D[5.38369, -1.8453, 0], Line2D[1.04481, 8.2738, 0]}
In[31]: Sketch2D[{p1, h1, lns, Map[Point2D[#, h1]&, lns]},
CurveLength2D -> 15];
7.5
```



| Hyperbola Equation                               | Tangent at $P_1(x_1,y_1)$                                  |
|--|--|
| $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$          | $\frac{xx_1}{a^2} - \frac{yy_1}{b^2} = 1$                  |
| $\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1$  | $\frac{(x-h)(x_1-h)}{a^2} - \frac{(y-k)(y-k_1)}{b^2} = 1$  |
| $-\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$         | $-\frac{xx_1}{a^2} + \frac{yy_1}{b^2} = 1$                 |
| $-\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$ | $-\frac{(x-h)(x_1-h)}{a^2} + \frac{(y-k)(y-k_1)}{b^2} = 1$ |

Table 18.5: Tangents to a hyperbola at a point.

Table 18.6: Tangents to a hyperbola with slope m.

| Hyperbola Equation                               | Tangent with Slope $m$                      |  |
|--|---|--|
| $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$          | $y = mx \pm \sqrt{a^2m^2 - b^2}$            |  |
| $\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1$  | $y - k = m(x - h) \pm \sqrt{a^2 m^2 - b^2}$ |  |
| $-\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$         | $y = mx \pm \sqrt{b^2 - a^2 m^2}$           |  |
| $-\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$ | $y - k = m(x - h) \pm \sqrt{b^2 - a^2 m^2}$ |  |

## Parallel and Perpendicular Tangents

Using the equations from the tables in the previous sections in the columns labeled TANGENT WITH SLOPE m, we can easily construct lines parallel or perpendicular to a given line and tangent to a given second-degree curve.

**Example.** Find the lines parallel and perpendicular to the line x + 2y - 2 = 0and tangent to the ellipse

$$\frac{(x+1)^2}{4} + \frac{(y-2)^2}{16} = 1$$

and plot.

F

**Solution.** The Descarta<sub>2D</sub> function TangentLines<sub>2D</sub>[*line*, *curve*, Parallel<sub>2D</sub>] constructs a list of lines parallel to the given line and tangent to the curve; the function TangentLines<sub>2D</sub>[*line*, *curve*, Perpendicular<sub>2D</sub>] returns a list of lines perpendicular to the given line and tangent to the curve.

```
In[32]: l1 = Line2D[1, 2, -2];
e1 = Ellipse2D[{-1, 2}, 4, 2, Pi / 2];

lns = {TangentLines2D[11, e1, Parallel2D],

TangentLines2D[11, e1, Perpendicular2D]}

Out[32] {{Line2D[1, 2, -3 - 2\sqrt{17}], Line2D[1, 2, -3 + 2\sqrt{17}]},

{Line2D[-2, 1, 4 (-1 - \sqrt{2}], Line2D[-2, 1, 4 (-1 + \sqrt{2}]]}}
```

In[33]: Sketch2D[{l1, e1, lns}, PlotRange -> {{-8, 6}, {-5, 7}}];



# 18.4 Explorations

LINE TANGENT TO A CIRCLE.....lntancir.nb Show that the line  $y = m(x - a) + a\sqrt{1 + m^2}$  is tangent to the circle  $x^2 + y^2 = 2ax$  for all values of m.

LINE NORMAL TO A QUADRATIC.....lnquad.nb

Show that the normal line passing through the point  $(x_1, y_1)$  on the quadratic whose equation is  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$  is given by

$$k_1x - k_2y - k_1x_1 + k_2y_1 = 0$$

where

$$k_1 = Bx_1 + 2Cy_1 + E$$
 and  $k_2 = 2Ax_1 + By_1 + D$ .

EYEBALL THEOREM.....eyeball.nb



The tangents to each of two circles from the center of the other are drawn as shown in the figure. Prove that the chords illustrated are equal in length.

PERPENDICULAR TANGENTS TO A PARABOLA.....pbtnlns.nb Show that if  $L_1$  and  $L_2$  are two lines tangent to a parabola that intersect on the directrix of the parabola, then  $L_1$  and  $L_2$  are perpendicular to each other.

TANGENT TO A PARABOLA WITH A GIVEN SLOPE......pbslp.nb Show that the line tangent to the parabola  $y^2 = 4px$  with slope m is y = mx + p/m. TANGENT TO AN ELLIPSE WITH GIVEN SLOPE.....ellslp.nb Show that the lines tangent to the ellipse  $x^2/a^2 + y^2/b^2 = 1$  with slope m are given by  $y = mx \pm \sqrt{a^2m^2 + b^2}$ .

TANGENT TO A HYPERBOLA WITH GIVEN SLOPE......hypslp.nb Show that the lines tangent to the hyperbola  $x^2/a^2 - y^2/b^2 = 1$  with slope *m* are given by  $y = mx \pm \sqrt{a^2m^2 - b^2}$ .

TANGENCY POINT ON CIRCLE. .....tancirpt.nb Show that if a line Ax + By + C = 0 is tangent to a circle  $(x - h)^2 + (y - k)^2 = r^2$  then the coordinates of the point of tangency are

$$\left(h - \frac{Ar^2}{Ah + Bk + C}, k - \frac{Br^2}{Ah + Bk + C}\right).$$

MONGE'S THEOREM......monge.nb Given three circles and the external tangent lines of the circles taken in pairs, show that

the three intersection points of the three tangent pairs lie on a straight line.

LINE TANGENT TO A CONIC. ..... Intancon.nb Show that the relationship between the coefficients of a line px + qy + r = 0 tangent to the conic  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$  is given by

$$(4CF - E^2)p^2 + (4AF - D^2)q^2 + (4AC - B^2)r^2 + 2(BD - 2AE)qr + 2(BE - 2CD)pr + 2(DE - 2BF)pq = 0.$$

NORMALS AND MINIMUM DISTANCE.....normal.nb

Given a point  $P_0(x_0, y_0)$  and a quadratic Q, find point(s) P(x, y) on Q such that the line  $PP_0$  is perpendicular to Q at P. Such a line is called a *normal* to the quadratic. Use the points P to find the minimum distance from  $P_0$  to Q. Assume that  $P_0$  and Q are defined numerically.

# Chapter 19

# **Tangent Circles**

In our study of circles we noted that the equation of a circle

$$(x-h)^2 + (y-k)^2 = r^2$$

has three parameters, h, k and r, that may be varied independently. A circle, therefore, is said to have three *degrees of freedom* (DOF). These degrees of freedom allow us to construct a circle so that it meets three conditions. Some common conditions and the corresponding equations that establish the condition are shown in Table 19.1.

By specifying a combination of conditions so that the degrees of freedom add up to three, we can then solve three simultaneous equations in three unknowns (h, k and r) and determine the (possibly empty) set of circles that satisfy the conditions. For economy of expression in the following sections, we will use the convention that a point which is *on* a circle (i.e. satisfies the equation of the circle) will be said to be *tangent* to the circle.

# **19.1** Tangent Object, Center Point

To construct a circle tangent to an object (a point, line or circle) with a given center point, we select equations as follows from Table 19.1. To establish the condition that a circle be tangent to a point, line or circle, we select the appropriate equation from cases [5], [6] or [7]; to establish the condition that a circle have a given center point we select the two equations from case [1]. Solving these three equations in three unknowns produces the values for the parameters h, k and r of the circles which satisfy the stated conditions.

**Example.** Construct the circle(s) tangent to the circle  $x^2 + y^2 = 4$  with center point (-1, 0) and plot.

**Solution.** The function TangentCircles2D[{*circle*}, *point*] returns a list of circles tangent to a circle with a given center point.

| Case | Condition                           | Equation(s)                         | DOF |
|------|-------------------------------------|-------------------------------------|-----|
| [1]  | Fixed radius                        | $r = r_1$                           | 1   |
|      | $r_1$                               |                                     |     |
| [2]  | Fixed center point                  | $h = x_1$ and $k = y_1$             | 2   |
|      | $(x_1,y_1)$                         |                                     |     |
| [3]  | Center on line                      | $A_1h + B_1k + C_1 = 0$             | 1   |
|      | $A_1x + B_1y + C_1 = 0$             |                                     |     |
| [4]  | Center on circle                    | $(h - h_1)^2 + (k - k_1)^2 = r_1^2$ | 1   |
|      | $(x - h_1)^2 + (y - k_1)^2 = r_1^2$ |                                     |     |
| [5]  | Through a point                     | $(x_1 - h)^2 + (y_1 - k)^2 = r^2$   | 1   |
|      | $(x_1,y_1)$                         |                                     |     |
| [6]  | Tangent to a line                   | $(A_1^2 + B_1^2)r^2 =$              | 1   |
|      | $A_1x + B_1y + C_1 = 0$             | $(A_1h + B_1k + C_1)^2$             |     |
| [7]  | Tangent to a circle                 | $(D - (r_1 - r)^2) \times$          | 1   |
|      | $(x - h_1)^2 + (y - k_1)^2 = r_1^2$ | $(D - (r_1 + r)^2) = 0,$            |     |
|      |                                     | where                               |     |
|      |                                     | $D = (h_1 - h)^2 + (k_1 - k)^2$     |     |

Table 19.1: Circle tangency equations.

 $\textit{Out[1]} \ \{\texttt{Circle2D[\{-1, 0\}, 1], Circle2D[\{-1, 0\}, 3]}\}$ 

In[2]: Sketch2D[{p1, c1, cirs}];



**Descarta2D Hint.** TangentCircles2D[ $\{pt | ln | cir\}$ , point] is the general function that returns a list of circles tangent to an object (a point, line or circle) with a given center point. The vertical bar syntax separating the point, line and circle arguments indicates that a point, line or circle may be specified.

# 19.2 Tangent Object, Center on Object, Radius

To construct a circle tangent to an object (a point, line or circle) with center point on an object (a line or circle), with a given radius, we select equations as follows from Table 19.1. To establish the condition that a circle be tangent to a point, line or circle, we select the appropriate equation from cases [5], [6] or [7]; to establish the condition that the center of the circle be on a given line or circle we select the appropriate equation from cases [3] or [4]; to establish the condition that the circle have a given radius we select the equation from case [1]. Solving these three equations in three unknowns produces the values for the parameters h, k and r of the circles which satisfy the stated conditions.

**Example.** Construct the circle(s) tangent to the circle  $x^2 + y^2 = 4$ , center on the line x - 2y + 1 = 0 and with radius 1 and plot.

**Solution.** The function  $TangentCircles2D[{circle}, line, r]$  returns a list of circles tangent to a given circle, with center on a line, with a given radius.

```
 \begin{array}{l} In \left[ 3 \right]: \mbox{ cirs = TangentCircles2D[{cl = Circle2D[{0, 0}, 2]}, \\ 11 = Line2D[1, -2, 1], 1] \\ Out \left[ 3 \right] \ \left\{ \mbox{Circle2D[{-1, 0}, 1], \\ Circle2D[\left\{ \frac{3}{5}, \frac{4}{5} \right\}, 1], \mbox{Circle2D[\left\{ \frac{1}{5} \left( -1 - 4\sqrt{11} \right), \frac{2}{5} \left( 1 - \sqrt{11} \right) \right\}, 1], \\ \mbox{Circle2D[\left\{ \frac{1}{5} \left( -1 + 4\sqrt{11} \right), \frac{2}{5} \left( 1 + \sqrt{11} \right) \right\}, 1] \right\} \end{array} \right.
```

```
In[4]: Sketch2D[{l1, c1, cirs}];
```



**Descarta2D Hint.** TangentCircles2D[ $\{pt | ln | cir\}$ , ln | cir, r] is the generalized function that returns a list of circles tangent to an object (a point, line or circle) with center on a line or circle, with a given radius, r.

# 19.3 Two Tangent Objects, Center on Object

To construct a circle tangent to two objects (points, lines or circles) with center point on an object (a line or circle), we select equations as follows from Table 19.1. To establish the condition that a circle be tangent to a point, line or circle, we select the appropriate equation from cases [5], [6] or [7]—this produces two equations (one for each tangent object); to establish the condition that the center be on a line or circle we select the appropriate equation from cases [3] or [4]. Solving these three equations in three unknowns produces the values for the parameters h, k and r of the circles which satisfy the stated conditions.

**Example.** Construct the circle(s) tangent to the two circles

 $(x+2)^2 + y^2 = 1$  and  $(x-2)^2 + y^2 = 1$ ,

with center point on the line x - 2y + 1 = 0 and plot.

**Solution.** The function TangentCircles2D[{*cir*, *cir*}, *line*] returns a list of circles tangent to two circles, with center point on a given line.

```
In[5]: \text{ cirs = TangentCircles2D[{c1 = Circle2D[{-2, 0}, 1], c2 = Circle2D[{2, 0}, 1]}, 11 = Line2D[1, -2, 1]]
Out[5] \left\{ \text{Circle2D[{-1, 0}, 2], Circle2D[{0, <math>\frac{1}{2}}, \frac{1}{2} (-2 + \sqrt{17}) \right], \text{Circle2D[{0, <math>\frac{1}{2}}, \frac{1}{2} (2 + \sqrt{17}) ], \text{Circle2D[{1, <math>\frac{1}{2}}, \frac{12}{11}, \frac{26}{11} ]} \right\}
```

In[6]: Sketch2D[{11, c1, c2, cirs}];



**Descarta2D Hint.** TangentCircles2D[ $\{pt | ln | cir, pt | ln | cir\}$ , ln | cir] is the general function that returns a list of circles tangent to two objects (points, lines or circles) with center point on a line or circle.

## 19.4 Two Tangent Objects, Radius

To construct a circle tangent to two objects (points, lines or circles) with a given radius, we select equations as follows from Table 19.1. To establish the condition that a circle be tangent to a point, line or circle, we select the appropriate equation from cases [5], [6] or [7]—this produces two equations (one for each tangent object); to establish the condition that the circle have a given radius we select the equation from case [1]. Solving these three equations in three unknowns produces the values for the parameters h, k and r of the circles which satisfy the stated conditions.

**Example.** Construct the circle(s) tangent to the two circles

$$(x+2)^2 + y^2 = 9$$
 and  $(x-2)^2 + y^2 = 9$ ,

with a radius of 1 and plot.

**Solution.** The function TangentCircles2D[{*circle*, *circle*}, *r*] returns a list of circles tangent to two circles, with a given radius.

```
In[7]: cirs = TangentCircles2D[{c1 = Circle2D[{-2, 0}, 3], c2 = Circle2D[{2, 0}, 3], 1]
Out[7] \{Circle2D[\{-\frac{3}{2}, -\frac{\sqrt{15}}{2}\}, 1], Circle2D[\{-\frac{3}{2}, \frac{\sqrt{15}}{2}\}, 1], Circle2D[{0, 0}, 1], Circle2D[{0, -2\sqrt{3}}, 1], Circle2D[{0, 2\sqrt{3}}, 1], Circle2D[{\frac{3}{2}, -\frac{\sqrt{15}}{2}}, 1], Circle2D[{\frac{3}{2}, -\frac{\sqrt{15}}{2}}], Circle2D[{\frac{3}{2}, -\frac{\sqrt{15}}{2}}], Circle2D[{\frac{3}{2}, -\frac{\sqrt{15}}{2}}], Circle2D[{\frac{3}{2}, -
```

```
In[8]: Sketch2D[{c1, c2, cirs}];
```



**Descarta2D Hint.** TangentCircles2D[ $\{pt | ln | cir, pt | ln | cir\}$ , r] is the general function that returns a list of circles tangent to two objects (points, lines or circles) with a given radius, r.

# 19.5 Three Tangent Objects

To construct a circle tangent to three objects (points, lines or circles), we select equations as follows from Table 19.1. To establish the condition that a circle be tangent to a point, line or circle, we select the appropriate equation from cases [5], [6] or [7]—this produces three equations (one for each tangent object). Solving these three equations in three unknowns produces the values for the parameters h, k and r of the circles which satisfy the stated conditions.

**Example.** Construct and plot the circle(s) tangent to the three lines x - y + 1 = 0, x + y - 1 = 0 and y + 1 = 0.

**Solution.** Use the function  $TangentCircles2D[{ln, ln, ln}]$  that returns a list of circles tangent to the three lines.

E

```
\begin{array}{l} \textit{Out[9]} & \left\{ \texttt{Circle2D}\left[ \left\{ 0 \,,\, -3-2\,\sqrt{2} \right\} ,\,\, 2+2\,\sqrt{2} \right] ,\,\, \texttt{Circle2D}\left[ \left\{ 0 \,,\, -3+2\,\sqrt{2} \right\} ,\,\, -2+2\,\sqrt{2} \right] ,\,\, \texttt{Circle2D}\left[ \left\{ -2\,\sqrt{2} \,,\, 1 \right\} ,\,\, 2 \right] ,\,\, \texttt{Circle2D}\left[ \left\{ 2\,\sqrt{2} \,,\, 1 \right\} ,\,\, 2 \right] \right\} \end{array}
```

```
In[10]: Sketch2D[{11, 12, 13, cirs}, PlotRange -> {{-5, 5}, {-5, 5}}];
```



**Descarta2D Hint.** The function TangentCircles2D[{*obj*<sub>1</sub>, *obj*<sub>2</sub>, *obj*<sub>3</sub>}] is the general function that returns a list of circles tangent to three objects (points, lines or circles).

# 19.6 Explorations

ARCHIMEDES' CIRCLES.....archimed.nb



Draw the vertical tangent line at the intersection point of the two smaller tangent circles,  $c_1$  and  $c_2$ , in an *arbelos* (shoemaker's knife, see figure). Prove that the two circles C' and C'' tangent to this line, the large semicircle,  $c_3$  and  $c_1$  and  $c_2$  are *congruent* (have equal radii). These circles are known as Archimedes' Circles.

CIRCLE TANGENT TO CIRCLE, GIVEN CENTER ......tancir1.nb Show that the radii of the two circles centered at  $(h_1, k_1)$  and tangent to the circle

$$(x - h_2)^2 + (y - k_2)^2 = r_2^2$$

are given by

$$r = |d \pm r_2|$$

where  $d = \sqrt{(h_1 - h_2)^2 + (k_1 - k_2)^2}$ . This formula is a special case of the *Descarta2D* function TangentCircles2D[{pt | ln | cir}, point].

CIRCLE TANGENT TO CIRCLE, CENTER ON CIRCLE, RADIUS.....tancir2.nb

Show that the centers (h, k) of the two circles passing through the point  $(x_1, y_1)$  with center on the circle  $x^2 + y^2 = 1$  and radius r = 1 are given by

$$(h,k) = \left(\frac{x_1}{2} \pm \frac{y_1\sqrt{4-d_1^2}}{2d_1}, \frac{y_1}{2} \mp \frac{x_1\sqrt{4-d_1^2}}{2d_1}\right)$$

where  $d_1 = \sqrt{x_1^2 + y_1^2}$ . This is a special case of the Descarta2D function

TangentCircles2D[ $\{pt \mid ln \mid cir\}$ ,  $ln \mid cir, r$ ]

that constructs a list of circles.

$$A_1x + B_1y = 0$$
 and  $-B_1x + A_1y = 0$ 

with radius r = 1 are given by

$$(h,k) = (A_1 - B_1, A_1 + B_1),$$
  
=  $(A_1 + B_1, -A_1 + B_1),$   
=  $(-A_1 - B_1, A_1 - B_1)$  and  
=  $(-A_1 + B_1, -A_1 - B_1).$ 

Assume that the two lines are normalized,  $A_1^2 + B_1^2 = 1$ . This construction is a special case of the Descarta2D function TangentCircles2D[{ $obj_1$ ,  $obj_2$ }, r] when the two objects are lines.

CIRCLE THROUGH TWO POINTS, CENTER ON CIRCLE......tancir4.nb

Show that the radii of the two circles passing through the points (0, a) and (0, -a) with centers on the circle  $x^2 + y^2 = r_2^2$  are both given by

$$r = \sqrt{a^2 + r_2^2}.$$

This is a special case of TangentCircles2D[ $\{obj_1, obj_2\}$ ,  $ln \mid cir$ ] where the two objects are points.

CIRCLE TANGENT TO THREE LINES.....tancir5.nb Show that the radii of the four circles tangent to the lines

$$x = 0, y = 0$$
 and  $Ax + By + C = 0$ .

are given by

$$r = \left| \frac{C}{1 \pm A \pm B} \right|$$

taking all four combinations of signs and assuming the lines are normalized. This is a special case of the function  $TangentCircles2D[{obj_1, obj_2, obj_3}]$  where all three of the objects are lines.

CIRCLES TANGENT TO AN ISOSCELES TRIANGLE.....tncirtri.nb

A circle is inscribed in an isosceles triangle with sides a, a and 2b in length. A second, smaller circle is inscribed tangent to the first circle and to the equal sides of the triangle. Show that the radius of the second circle is

$$r = b\sqrt{\frac{(a-b)^3}{(a+b)^3}}.$$

Assume that a > b.

# Chapter 20

# **Tangent Conics**

The most general quadratic equation in two unknowns

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

has six coefficients, but since we can divide the coefficients by any non-zero constant, say F, without altering the equality obtaining

$$A'x^{2} + B'xy + C'y^{2} + D'x + E'y + 1 = 0$$

a quadratic equation only has five degrees of freedom. Thus we may specify five conditions (or constraints) on a quadratic equation. In this chapter we will investigate the construction of conic curves (circles, parabolas, ellipses and hyperbolas) that satisfy a set of five conditions, when the conditions are of two specific types: either passing through a given point, or tangent to a given line. The resulting equations are sufficiently complex that obtaining the solutions in symbolic, closed form is of no practical value, so we will illustrate the solution techniques and use the numerical capabilities of *Mathematica* to compute specific solutions.

## 20.1 Constraint Equations

As mentioned in previous chapters, if the curve represented by the quadratic equation

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

passes through the point  $P_1(x_1, y_1)$ , then the point will satisfy the equation yielding the relationship

$$Ax_1^2 + Bx_1y_1 + Cy_1^2 + Dx_1 + Ey_1 + F = 0.$$
 (20.1)

It has also been shown in previous chapters that the line px + qy + r = 0 will be tangent to the curve represented by the quadratic equation if the coefficients satisfy the equation

$$(4CF - E^2)p^2 + (4AF - D^2)q^2 + (4AC - B^2)r^2 + 2(BD - 2AE)qr + 2(BE - 2CD)pr + 2(DE - 2BF)pq = 0.$$
(20.2)

-

# 20.2 Systems of Quadratics

In this section we will outline the general technique for finding quadratics that pass through the four points of intersection of two quadratic curves. These techniques will be the basis for subsequent sections wherein we will find quadratics satisfying a variety of conditions.

Two quadratics intersect in four points (four real, two real and two complex, or four complex) since each equation is of the second degree. If

$$Q_1 \equiv A_1 x^2 + B_1 x y + C_1 y^2 + D_1 x + E_1 y + F_1 = 0 \text{ and}$$
  

$$Q_2 \equiv A_2 x^2 + B_2 x y + C_2 y^2 + D_2 x + E_2 y + F_2 = 0$$

represent the equations of the two quadratics, then  $Q \equiv Q_1 + kQ_2 = 0$ , for any constant k, is the equation of a quadratic through the points of intersection of  $Q_1$  and  $Q_2$ . The equation Qis called a *system* or *pencil* of quadratics, and placing one additional condition on the equation for Q allows us to solve for k and find a specific quadratic in the pencil. The equation of the pencil is sometimes written as  $Q \equiv (1 - k)Q_1 + kQ_2$  in order to allow the original quadratics,  $Q_1$  and  $Q_2$ , to be in the pencil (for k = 0 and k = 1, respectively).

**Example.** Find the quadratic that passes through the intersection of the ellipse  $x^2 + 4y^2 - 10x - 39 = 0$  and the hyperbola  $-x^2 + y^2 - 1 = 0$  and also passes through the point (-4, 0).

Solution. The equation of the quadratic pencil containing the solutions is

$$(x^{2} + 4y^{2} - 10x - 39) + k(-x^{2} + y^{2} - 1) = 0$$

and this must be satisfied by (-4, 0). Hence, solving for k yields k = 1 and the final equation of the conic sought is  $x^2 + 2y - 8 = 0$  (a parabola).





## A Degenerate Case

F

F

If Q = 0 is the equation of a quadratic and L = 0 is the equation of a straight line, then  $Q + kL^2 = 0$  is the equation of a quadratic tangent to Q at the intersection points of Q = 0 and L = 0. We may think of  $L^2 = 0$  as a (degenerate) quadratic (two coincident lines) intersecting Q = 0 in two pairs of coincident points each.

**Example.** Find the quadratic tangent to  $Q \equiv x^2 - y^2 - y + 1 = 0$  at the points of intersection of Q and  $L \equiv 3x - 2y - 1 = 0$  and passing through the point (-1, 0).

Solution. The equation is of the form

$$(x^{2} - y^{2} - y + 1) + k(3x - 2y - 1)^{2} = 0.$$

Substituting (-1, 0) into this equation we get k = 1/8, which yields as the equation of the conic sought

 $x^{2} - 12xy + 12y^{2} - 6x + 12y - 7 = 0$  (a hyperbola).

```
In[2]: Sketch2D[{Quadratic2D[1, 0, -1, 0, -1, 1],
Line2D[3, -2, -1],
Quadratic2D[1, -12, 12, -6, 12, -7],
Point2D[{-1, 0}]};
```



Π

\_\_\_\_\_



Figure 20.1: Quadratic through five points.

# 20.3 Validity Conditions

In the remainder of this chapter we will outline techniques for finding conics that satisfy five conditions. The conditions will be of two types: either passing through a given point, or tangent to a given line. The following assumptions are made with respect to the five points and/or lines:

- no pair of points is coincident,
- no pair of lines is coincident,
- no triple of points is collinear,
- no triple of lines is concurrent,
- no triple of lines is mutually parallel,
- no more than one point is on each line, and
- no more than one line passes through each point.

Degenerate conics may exist that satisfy configurations of points and lines that violate these restrictions, but we will focus our attention on cases that produce proper conics (circles, parabolas, ellipses and hyperbolas).

## 20.4 Five Points

Given five points,  $P_1$ ,  $P_2$ ,  $P_3$ ,  $P_4$  and  $P_5$ , satisfying the validity conditions stated in Section 20.3, we wish to find the quadratic that passes through all five points. Consider the lines  $L_{12}$ ,  $L_{34}$ ,  $L_{13}$  and  $L_{24}$  passing through the points in pairs as shown in Figure 20.1. Let

 $Q_1 \equiv L_{12}L_{34}$  be a (degenerate) quadratic (a pair of lines) passing through  $P_1$ ,  $P_2$ ,  $P_3$  and  $P_4$ . Similarly, let  $Q_2 \equiv L_{13}L_{24} = 0$  be a second quadratic passing through the same four points. The equation  $Q \equiv Q_1 + kQ_2 = 0$  will then represent the pencil of quadratics parameterized by the variable k passing through the four points.

Applying Equation (20.1), by substituting the coordinates of the point  $P_5$  into the equation for Q, we can solve this linear equation for the value of k, thereby yielding the specific quadratic in the pencil of quadratics that passes through all five points. Mathematica can be used to solve for k and form the symbolic equation for Q, although the result is quite cumbersome in expanded form. A determinant can be used to represent the resulting quadratic in a more convenient and simplified form and is given by

$$Q = \begin{vmatrix} x^2 & xy & y^2 & x & y & 1 \\ x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & x_2y_2 & y_2^2 & x_2 & y_2 & 1 \\ x_3^2 & x_3y_3 & y_3^2 & x_3 & y_3 & 1 \\ x_4^2 & x_4y_4 & y_4^2 & x_4 & y_4 & 1 \\ x_5^2 & x_5y_5 & y_5^2 & x_5 & y_5 & 1 \end{vmatrix}$$

**Example.** Find the quadratic passing through the five points (3,0), (3,1), (0,1), (-3,0) and (0,-1).

**Solution.** The function Quadratic2D[*pt*, *pt*, *pt*, *pt*, *pt*] returns the quadratic passing through the five points.

**Example.** Find the conic represented by the quadratic found in the previous example. Plot the points and the conic curve.

**Solution.** The conic can be determined directly from the result of the previous example using the function Loci2D[quad]. Descarta2D also provides the function TangentConics2D[{pt, pt, pt, pt, pt}] that constructs a list containing the conic directly from the five points.



**Descarta2D Hint.** TangentQuadratics2D[{pt, pt, pt, pt, pt}] constructs a list containing the single quadratic passing through five points. Except for the fact the TangentQuadratics2D checks for the validity conditions stated in Section 20.3, this function is equivalent to Quadratic2D[pt, pt, pt, pt, pt].

# 20.5 Four Points, One Tangent Line

In this section we will consider the construction of quadratics and conics passing through four points and tangent to a line. Two cases are distinguished: the first constructs the quadratic or conic when none of the given points lie on the tangent line; the second constructs the quadratic or conic when one of the given points does lie on the tangent line.

## Points Not on a Tangent Line

Assume that points  $P_1$ ,  $P_2$ ,  $P_3$  and  $P_4$  and line  $L_5$  as shown in Figure 20.2 satisfy the validity conditions stated in Section 20.3 and that none of the four points lie on  $L_5$ . To find the equation of the quadratic passing through the four points and tangent to the line, we form a pencil of quadratics passing through the four points parameterized by the variable k that is given by

$$Q \equiv L_{12}L_{34} + kL_{13}L_{24} = 0.$$

We now apply the condition that line  $L_5$  is tangent to Q by using Equation (20.2) resulting in a quadratic equation in the variable k. Solving this equation yields two values for k that can be substituted into the equation for Q, giving two quadratics satisfying the stated conditions. E



Figure 20.2: Four points, one line, no points on the line.

**Example.** Find the quadratics passing through the points (2, 1), (-2, 1), (-2, -1) and (2, -1) and tangent to the line 3x+4y-12 = 0. Plot the conic curves associated with the quadratics.

**Solution.** The Descarta2D function TangentQuadratics2D[ $\{pt, pt, pt, pt, ln\}$ ] constructs a list of quadratics passing through the four points and tangent to the line. TangentConics2D[ $\{pt, pt, pt, pt, ln\}$ ] constructs a list of conic curves passing through the four points and tangent to the line. Both functions allow the points and line to be listed in any order.

```
In[6]: objs = \{Point2D[\{2, 1\}], Point2D[\{-2, 1\}], Point2D[\{-2, -1\}], Point2D[\{-2, -1\}], Point2D[\{2, -1\}], Line2D[3, 4, -12]\}; TangentQuadratics2D[objs]
Out[6] \{Quadratic2D[\frac{1}{2}(-23 - \sqrt{385}), 0, -2(-23 - \sqrt{385}) - 16(1 + \frac{1}{8}(23 + \sqrt{385}))), 0, 0, 16(1 + \frac{1}{8}(23 + \sqrt{385}))], Quadratic2D[\frac{1}{2}(-23 + \sqrt{385}), 0, -2(-23 + \sqrt{385}) - 16(1 + \frac{1}{8}(23 - \sqrt{385}))], 0, 0, 0, 16(1 + \frac{1}{8}(23 - \sqrt{385}))]\}
In[7]: crvs = TangentConics2D[objs] // N
```

```
Out[7] {Ellipse2D[{0, 0}, 2.51549, 2.17963, 1.5708],
Ellipse2D[{0, 0}, 3.67034, 1.19261, 0]}
```

In[8]: Sketch2D[{objs, crvs}];



Figure 20.3: Four points, one line, one point on the line.



F

## **One Point on Tangent Line**

We now examine the case when one of the four points is on the tangent line. Consider the points  $P_1$ ,  $P_2$ ,  $P_3$  and  $P_4$  and the line  $L_1$  as shown in Figure 20.3 satisfying the validity conditions stated in Section 20.3, where the point  $P_1$  is on  $L_1$ . Since the desired quadratic is tangent to  $L_1$  at  $P_1$ , we can consider  $P_1$  to be a pair of coincident intersection points of the pencil of quadratics passing through the four points  $P_1$ ,  $P_2$  and  $P_3$  ( $P_1$  is counted as two coincident intersection points). We now form the pencil of quadratics parameterized by the variable k and given by  $Q \equiv L_{12}L_{13} + kL_1L_{23} = 0$ . The coordinates of the remaining point,  $P_4$ , must satisfy the equation of the quadratic, and by applying Equation (20.1) we generate a linear equation in the variable k that can be solved yielding the single quadratic equation satisfying the stated conditions.

**Example.** Find the conic curve passing through the points (2,0), (0,1), (-2,0) and (0,-1) and tangent to the line y = 1.

**Solution.** The function TangentConics2D[ $\{pt, pt, pt, pt, pt, ln\}$ ] returns a list of conic curves passing through four points and tangent to a line. The points and line may be listed in any order.

```
In[9]: crv = TangentConics2D[
    objs = {Point2D[{2, 0}], Point2D[{0, 1}],
        Point2D[{-2, 0}], Point2D[{0, -1}],
        Line2D[0, 1, -1]}]
```

```
Out[9] {Ellipse2D[{0,0},2,1,0]}
```

In[10]: Sketch2D[{objs, crv}, PlotRange -> {{-3, 3}, {-1.5, 1.5}}];



**Descarta2D Hint.** In the remaining sections of this chapter we will use the function TangentConics2D to find the tangent curves satisfying a variety of conditions. The function TangentQuadratics2D is also available in all these cases and will return a list of quadratics instead of a list of conic curves.

## 20.6 Three Points, Two Tangent Lines

We now consider the construction of a conic passing through three points and tangent to two lines. Three cases need to be considered: the first constructs the conic when none of the given points lie on either of the tangent lines; the second constructs the conic when one of the given points lies on one of the tangent lines; and, finally, two points lie on two of the tangent lines.

## Points Not on Tangent Lines

Consider three points  $P_1$ ,  $P_2$  and  $P_3$  and two lines  $L_4$  and  $L_5$  satisfying the validity conditions stated in Section 20.3. We also assume that none of the points are on either line as shown in Figure 20.4. The line  $L_{45}$  passing through the points of tangency between lines  $L_4$  and  $L_5$ and the desired conic curve can be written in the form

$$L_{45} \equiv ax + by - 1 = 0$$



Figure 20.4: Three points, two lines, no points on the lines.

assuming we can guarantee that  $L_{45}$  does not pass through the origin. The point  $P_1$  is clearly not on  $L_{45}$  because that would imply that the conic passes through three distinct, collinear points, which clearly violates the validity conditions. Therefore, if we translate all five of the original objects so that point  $P_1$  is at the origin, we can guarantee that  $L_{45}$  does not pass through the origin. Of course we need to perform the inverse translation on the resulting conic curves to produce the solution for the geometry in its original position.

We now proceed with  $L_{45} \equiv ax + by - 1 = 0$ , a line that does not pass through the origin. Consider the pencil of quadratics parameterized by the variable k and represented by the equation

$$Q \equiv L_4 L_5 - k L_{45}^2 = 0.$$

Solving this equation for k yields

$$k = \frac{L_1 L_2}{L_{45}^2}.$$

The right side of this equation is an expression in x and y involving the unknowns a and b. The expression must produce the same value for k for any point (x, y) on the desired quadratic. In particular, points  $P_1$ ,  $P_2$  and  $P_3$  must all produce the same value for k. Using the expression  $f[P_n]$  to indicate the expression f evaluated at the point  $P_n$ , we can write the system of equations

$$\frac{L_4L_5}{L_{45}^2}[P_1] = \frac{L_4L_5}{L_{45}^2}[P_2] = \frac{L_4L_5}{L_{45}^2}[P_3]$$

since all of these expressions must equal k. Rewriting these equations as a system of two equations and cross-multiplying yields two quadratic equations in two unknowns, a and b,

$$(L_4[P_1])(L_5[P_1])(L_{45}^2[P_2]) = (L_4[P_2])(L_5[P_2])(L_{45}^2[P_1]) (L_4[P_2])(L_5[P_2])(L_{45}^2[P_1]) = (L_4[P_1])(L_5[P_1])(L_{45}^2[P_2]).$$

F

F

Solving these equations for a and b yields four pairs of solutions which can be substituted into

$$k = \frac{L_1 L_2}{L_{45}^2} [P_1]$$

producing four quadratics Q satisfying the stated conditions. The resulting quadratics may be translated back to the original position of the defining objects by translating the origin back to  $P_1$ .

**Example.** Find the conics passing through (1,0), (0,-1),  $(1/\sqrt{2},-1/\sqrt{2})$  and tangent to the lines x = 1 and y = -1.

**Solution.** The function  $TangentConics2D[{pt, pt, pt, ln, ln}]$  returns a list of conic curves passing through three points and tangent to two lines. The points and lines may be listed in any order.

```
In[11]: objs = {Point2D[{1, 0}], Point2D[{0, -1}],
          Point2D[{1 / Sqrt[2], -1 / Sqrt[2]}],
          Line2D[0, 1, -1], Line2D[1, 0, 1]};
       crvs = TangentConics2D[objs] // N
Out[11] {Circle2D[{0, 0}, 1.],
        Ellipse2D[{0.135729, -1.32236}, 2.57181, 0.262711, 1.12421],
        Ellipse2D[{0.600884, -0.600884}, 2.259, 0.150221, 0.785398],
        Ellipse2D[{1.32236, -0.135729}, 2.57181, 0.262711, 0.446587]}
In[12]: Map[Sketch2D[{objs, #},
             PlotRange -> \{\{-2, 4\}, \{-4, 2\}\}\},
         crvs];
                     2
                                                2
                     1
                                               1
                                               0
                     0
                                              -1
                   -1
```

-2

-3

-4 -101234 -4 -101234

-2

-3



Figure 20.5: Three points, two lines, one point on a line.



## **One Point on Tangent Line**

F

We now consider the case where one of the three points is on one of the two tangent lines. Assume that point  $P_1$  is on line  $L_1$  and points  $P_2$  and  $P_3$  are on neither line  $L_1$  or  $L_4$  as shown in Figure 20.5. Also, we assume the points and lines satisfy the validity conditions stated in Section 20.3. We form the pencil of quadratics

$$Q \equiv L_1 L_{23} + k L_{12} L_{13}$$

where  $L_{12}$ ,  $L_{13}$  and  $L_{23}$  are the lines passing through points  $P_1$  and  $P_2$ ,  $P_1$  and  $P_3$  and  $P_2$ and  $P_3$ , respectively. We now apply the tangency condition by using Equation (20.2) with Qand line  $L_4$  to form a quadratic equation in the variable k. Solving the equation for k gives the two quadratics passing through the points and tangent to the lines.

**Example.** Find the conic curves passing through the points (0, 1), (-3, 0) and (0, -1) and tangent to the lines y = 1 and x - 2y - 3 = 0.



Figure 20.6: Three points, two lines, two points on the lines.

**Solution.** The function  $TangentConics2D[{pt, pt, pt, ln, ln}]$  returns a list of conic curves passing through three points and tangent to two lines. The points and lines may be listed in any order.

```
In[13]: objs = {Point2D[{0, 1}], Point2D[{-3, 0}], Point2D[{0, -1}],
Line2D[0, 1, -1], Line2D[1, -2, -3]};
crvs = TangentConics2D[objs] // N
Out[13] {Ellipse2D[{-2.54874, -1.26827}, 3.8739, 1.32515, 0.530218],
```

```
Ellipse2D[{-0.267309, -0.00955005}, 2.7504, 1.00402, 0.0412054]}
```

```
In[14]: Sketch2D[{objs, crvs}];
```



## **Two Points on Tangent Lines**

Let  $P_1$  be a point on line  $L_1$ ,  $P_3$  be a point on line  $L_3$ , and  $P_2$  a point not on either line as shown in Figure 20.6 and assume that these points and lines satisfy the validity conditions stated in Section 20.3. We form the pencil of quadratics

$$Q \equiv L_1 L_3 + k L_{13}^2 = 0$$

where  $L_{13}$  is the line passing through points  $P_1$  and  $P_3$ . We now apply Equation (20.1) establishing the condition that  $P_2$  must be on Q and, therefore, the coordinates  $(x_2, y_2)$  must satisfy Q. We can solve this linear equation for k and determine the coefficients of the quadratic Q satisfying the stated conditions.

**Example.** Find the conic curve passing through the points (2, 1), (-2, 1) and (0, 2) and tangent to the lines x - 3y + 1 = 0 and 2x + y + 3 = 0.

**Solution.** The function  $TangentConics2D[{pt, pt, pt, ln, ln}]$  returns a list of conic curves passing through three points and tangent to two lines. The points and lines may be listed in any order.

```
In[15]: objs = {Point2D[{2, 1}], Point2D[{-2, 1}], Point2D[{0, 2}],
    Line2D[1, -3, 1], Line2D[2, 1, 3]};
    crvs = TangentConics2D[objs] // N
```

```
Out[15] {Ellipse2D[{0.463576, 1.37086}, 2.50231, 0.689197, 0.122489]}
```

```
In[16]: Sketch2D[{objs, crvs}];
```



Notice that a ConicArc2D object is a special case of this construction where the start and end points are the points  $P_1$  and  $P_3$  and the apex point defines the lines  $L_1$  and  $L_3$ .

# 20.7 Conics by Reciprocal Polars

In this section we will introduce the concept of *reciprocal polars*, and a technique that will allow us to solve tangent conic problems involving more than two tangent lines. Proofs of all the concepts involved in these techniques are beyond the scope of this book, but applying the techniques to solve tangent conic problems is easily grasped.
Let C be a circle in the plane and P a point. The *reciprocal* of P with respect to C is simply the polar line of P with respect to C. Similarly, let L be a line. The reciprocal of Lwith respect to C is the pole point of L with respect to C. It is noteworthy that the center point of the circle has no reciprocal, and any line passing through the center of the circle has no reciprocal.

If we have any figure consisting of any number of points and straight lines, and we take the polars of those points and the poles of the lines, with respect to a circle C, we obtain another figure which is called the *polar reciprocal* of the former with respect to the *auxiliary* circle C. When a point in one figure and a line in the reciprocal figure are pole and polar with respect to the auxiliary circle, C, the point and line are said to *correspond* to one another.

An important theorem from the analytic geometry of conics states that taking the reciprocal of all the points of a conic Q with respect to some auxiliary circle C will produce an *envelope* of lines tangent to another conic Q'. Furthermore, any line tangent to Q will correspond to a point P' on Q', and any line L' tangent to Q' will correspond to a point P on Q(always using C as the auxiliary circle).

We use this theorem as follows to find conics tangent to three, four or five lines and passing through a corresponding number of points so the total number of conditions equals five. First we apply an arbitrary translation to the objects to insure that none of the points lie at the origin and that none of the lines pass through the origin. We now take the reciprocal of the points or lines with respect to a unit circle centered at the origin, thereby producing a new figure of corresponding lines and points. Note that if there are three or more lines in the original figure, there will be two or fewer lines in its reciprocal.

We now apply the techniques developed in the previous sections of this chapter to find the quadratics(s) satisfying the conditions imposed by the elements in the reciprocal figure. Finally, we find the reciprocal of the resulting quadratic with respect to the auxiliary circle yielding the sought-after quadratics in the original figure. If the equation of the quadratic in the reciprocal figure is

$$Q' \equiv ax^2 + bxy + cy^2 + dx + ey + f = 0$$

then its equation in the original figure is given by

$$Q' = (4cf - e^2)x^2 + (2de - 4bf)xy + (4af - d^2)y^2 + (4cd - 2be)x + (4ae - 2db)y + (4ac - b^2) = 0.$$

The validity of this relationship is demonstrated in the exploration recquad.nb. The relationship between Q and Q' is only valid when the auxiliary circle is a unit circle at the origin  $(x^2 + y^2 = 1)$ .

## Two Points, Three Tangent Lines

**Example.** Find the conic curves passing through the points (3, -1) and (1, 0) and tangent to the lines 4x - y - 3 = 0, x + 2y - 3 = 0 and y = -2.

**Solution.** The function TangentConics2D[ $\{pt, pt, ln, ln, ln\}$ ] returns a list of conic curves passing through two points and tangent to three lines. The points and lines may be listed in any order. If neither point is on any of the lines, there are at most four real conic curves; if one of the points is on one of the lines, then there are at most two real conic curves; if two of the points are on the tangent lines, then there is at most one real conic curve.

```
In[17]: objs = {Point2D[{3, -1}], Point2D[{1, 0}], Line2D[4, -1, -3],
          Line2D[1, 2, -3], Line2D[0, 1, 2]};
        crvs = TangentConics2D[objs] // N
Out[17] {Ellipse2D[{1.79784, -0.811805}, 1.30361, 1.13329, 0.587329],
         Ellipse2D[{2.03133, -0.577222}, 1.71297, 0.620762, 2.21117],
         Ellipse2D[{3.64793, -0.854517}, 3.04088, 0.374464, 2.77469],
         Ellipse2D[{3.77722, -0.99446}, 3.18508, 0.250467, 2.82987]}
In[18]: Sketch2D[{objs, crvs},
            CurveLength2D -> 20,
            PlotRange -> { {-1, 8 }, {-4, 2 } } ];
                             2
                             1
                             0
                            -1
                            -2
                           -3
                            -4
                                 0
                                        2
                                              4
                                                     6
                                                           8
```

## **One Point, Four Tangent Lines**

E

**Example.** Find the conic curves passing through the point (-1, 1) and tangent to the lines 4x - y - 3 = 0, x + 2y - 3 = 0, x = -3 and y = -2.

**Solution.** The function  $TangentConics2D[{pt, ln, ln, ln, ln}]$  returns a list of conic curves passing through a point and tangent to four lines. The points and lines may be listed in any order. If the point is not on any of the lines, there will be at most two real conic curves; if the point is on one of the lines, then there will be at most one real conic curve.

```
In[19]: objs = {Point2D[{-1, 1}], Line2D[4, -1, -3],
Line2D[1, 2, -3], Line2D[1, 0, 3], Line2D[0, 1, 2]};
crvs = TangentConics2D[objs] // N
```

Out[19] {Ellipse2D[{-1.35291, 0.441105}, 2.88243, 0.602889, 2.14575], Ellipse2D[{-1.05825, -0.344658}, 2.29656, 1.11191, 0.656401]}

In[20]: Sketch2D[{crvs, objs}, PlotRange -> {{-4, 2}, {-3, 3}}];



**Five Tangent Lines** 

**Example.** Find the conic curve tangent to the five lines x - 2y + 3 = 0, x = 3, 2x - 3y - 2 = 0, y = 2 and x = -2. Plot the lines and the conic curve.

**Solution.** The function TangentConics2D[{ln, ln, ln, ln, ln}] returns a list of at most one conic curve tangent to five lines.

```
In[21]: objs = {Line2D[1, -2, 3], Line2D[1, 0, -3],
        Line2D[2, -3, -2], Line2D[0, 1, -2], Line2D[1, 0, 2]};
        crvs = TangentConics2D[objs] // N
```

Out[21] {Ellipse2D[{0.5, 0.833333}, 2.64903, 0.770563, 0.352906]}

In[22]: Sketch2D[{objs, crvs}, PlotRange -> {{-4, 4}, {-2, 3}}];



309

# 20.8 Explorations

RECIPROCALS OF POINTS AND LINES..... recptln.nb Show that the polar reciprocal of  $A_1x + B_1y + C_1 = 0$  in the auxiliary conic  $C \equiv x^2 + y^2 = 1$ 

is the point  $(-A_1/C_1, -B_1/C_1)$ , assuming that the line does not pass through the origin. Also, show that the line x + y - 1 = 0 is the polar reciprocal of the point (x, y) with respect to C.

RECIPROCAL OF A CIRCLE.....reccir.nb

Given a circle  $C_1 \equiv (x - h)^2 + (y - k)^2 = r^2$  show that its polar reciprocal in the auxiliary conic  $x^2 + y^2 = 1$  is given by the quadratic

$$Q \equiv (r^{2} - h^{2})x^{2} - 2hkxy + (r^{2} - k^{2})y^{2} + 2hx + 2ky - 1 = 0.$$

Furthermore, show that Q is an *ellipse*, if the origin (0,0) is inside C; a *parabola*, if the origin is on C; and a *hyperbola*, if the origin is outside C.

RECIPROCAL OF A QUADRATIC. ..... recquad.nb

Given a general quadratic  $Q \equiv ax^2 + bxy + cy^2 + dx + ey + f = 0$  show that the reciprocal of Q is the quadratic

$$(4cf - e^2)x^2 + (2de - 4bf)xy + (4af - d^2)y^2 + (4cd - 2be)x + (4ae - 2bd)y + (4ac - b^2) = 0$$

when the auxiliary conic is  $C \equiv x^2 + y^2 = 1$ .

PARABOLAS THROUGH FOUR POINTS......pb4pts.nb

Describe a method for finding the two parabolas passing through four points. Show that the technique produces the correct results for the points (2, 1), (-1, 1), (-2, -1) and (4, -3) by plotting the parabolas and the four points.

EQUILATERAL HYPERBOLAS.....hyp4pts.nb

Describe a method for finding the equilateral hyperbola(s) passing through four points. Show that the technique produces the correct results for the points (2, 1), (-1, 1), (-2, -1) and (4, -3) by plotting the hyperbola(s) and the four points.

# Chapter 21

# **Biarcs**

In this chapter we will demonstrate some techniques for adding new functions to Descarta2D. To make the demonstration realistic, we will introduce the mathematics for a new type of tangent circle construction called a *biarc*. Biarcs are used in some graphical computer systems to connect a set of data points with smoothly joined arcs. The mathematics of biarcs is by itself interesting and will serve as a good example of extending the capabilities of Descarta2D.

# 21.1 Biarc Carrier Circles

A biarc is a composite curve consisting of two circular arcs, placed end to end with continuity of slope at the join point. The two circles underlying the arc are called the biarc *carrier circles*. The carrier circles may be internally or externally tangent to each other, and the point of tangency that joins the two arcs is called the *knot point* of the biarc. Referring to Figure 21.1, suppose we wish to construct a smooth curve between points  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$  such that the tangents to the curve at  $P_1$  and  $P_2$  are the unit vectors  $\hat{T}_1(u_1, v_1)$ and  $\hat{T}_2(u_2, v_2)$ .  $P_1$ ,  $P_2$ ,  $\hat{T}_1$  and  $\hat{T}_2$  are called the biarc *configuration parameters*.

The geometric condition that two circles are tangent can always be expressed as

sum or difference of radii = distance between the centers 
$$(21.1)$$

according to the kind of contact, external (sum of radii) or internal (difference of radii).

We take positive values of  $r_1$  and  $r_2$  to indicate that the center points of the carrier circles,  $C_1$  and  $C_2$ , are offset in the direction of  $\hat{T}'_1$  and  $\hat{T}'_2$ , respectively.  $\hat{T}'_1(-v_1, u_1)$  and  $\hat{T}'_2(-v_2, u_2)$ are unit vectors constructed by rotating tangent vectors  $\hat{T}_1$  and  $\hat{T}_2$  90° counter-clockwise. Suppose we now wish to form an expression for the left-hand side of Equation (21.1). It can be shown that the expressions  $(r_1+r_2)^2$  and  $(r_1-r_2)^2$  represent all cases for the sum (squared) and difference (squared) of the biarc radii for all combinations of positive or negative  $r_1$  and  $r_2$  for both internally and externally tangent carrier circles.

We introduce a radius sign constant,  $s_r$ , which may take on the values  $\pm 1$ , in order to



Figure 21.1: Biarc configuration parameters.

accommodate internally or externally tangent carrier circles in the same equation,

(sum or difference of radii)<sup>2</sup> = 
$$(r_1 + s_r r_2)^2$$
. (21.2)

Note that it makes no difference whether we associate  $s_r$  with  $r_1$  or  $r_2$ , since after squaring and applying  $s_r^2 = 1$ , the relationship is symmetric,

$$(r_1 + s_r r_2)^2 = r_1^2 + 2s_r r_1 r_2 + s_r^2 r_2^2 = r_1^2 + 2s_r r_1 r_2 + r_2^2$$

The carrier circle center points,  $C_1$  and  $C_2$ , may be written as

$$C_1 = (x_1 - v_1 r_1, y_1 + u_1 r_1)$$

$$C_2 = (x_2 - v_2 r_2, y_2 + u_2 r_2).$$
(21.3)

Combining Equations (21.2) and (21.3) as suggested by Equation (21.1), the sum or difference of the radii (squared) equals the distance (squared) between  $C_1$  and  $C_2$ , yields

$$(r_1 + s_r r_2)^2 = ((x_2 - v_2 r_2) - (x_1 - v_1 r_1))^2 + ((y_2 + u_2 r_2) - (y_1 + u_1 r_1))^2.$$
(21.4)

When simplifying Equation (21.4), note that the relationships  $u_1^2 + v_1^2 = 1$  and  $u_2^2 + v_2^2 = 1$  can be used, since  $\hat{T}_1$  and  $\hat{T}_2$  are defined as unit vectors. Rearranging Equation (21.4) and using the following substitutions

$$f_0 = u_1 u_2 + v_1 v_2$$
  

$$f_1 = -v_1 (x_2 - x_1) + u_1 (y_2 - y_1)$$
  

$$f_2 = -v_2 (x_2 - x_1) + u_2 (y_2 - y_1)$$
  

$$d^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$$

produces the equation

$$r_1 r_2 (s_r + f_0) + f_1 r_1 - f_2 r_2 = \frac{d^2}{2}$$
(21.5)

which establishes the general relationship between the radii,  $r_1$  and  $r_2$ , of the carrier circles. Constants  $f_0$ ,  $f_1$ ,  $f_2$  and d are referred to as the *biarc defining constants*. Geometrically,  $f_0$  is the cosine of the angle between the tangent vectors,  $f_1$  is the (signed) distance from  $P_2$  to the line defined by  $\hat{T}_1$  and  $P_1$ ,  $f_2$  is the (signed) distance from  $P_1$  to the line defined by  $P_2$  and  $\hat{T}_2$  and  $d^2$  is the distance (squared) between points  $P_1$  and  $P_2$ . Note that these defining constants depend only on the relative position of the defining geometry and are independent of the choice of coordinate axes.

#### Radii Ratio

If a relationship between the carrier circle radii,  $r_1$  and  $r_2$ , is specified, then Equation (21.5) can be solved for the radii. We choose to specify the biarc *radii ratio*,  $\kappa = r_1/r_2$ , as the defining relationship. Substituting  $r_1 = \kappa r_2$  into Equation (21.5) produces the equation

$$\kappa(s_r + f_0)r_2^2 + (\kappa f_1 - f_2)r_2 = \frac{d^2}{2}.$$
 (21.6)

Solving Equation (21.6) by using the quadratic formula yields

$$r_{2} = \frac{(f_{2} - \kappa f_{1}) \pm \sqrt{(f_{2} - \kappa f_{1})^{2} + 2\kappa d^{2}(s_{r} + f_{0})}}{2\kappa(s_{r} + f_{0})}$$

$$r_{1} = \frac{(f_{2} - \kappa f_{1}) \pm \sqrt{(f_{2} - \kappa f_{1})^{2} + 2\kappa d^{2}(s_{r} + f_{0})}}{2(s_{r} + f_{0})}.$$
(21.7)

In the special case where the tangent vectors are in the same direction, the denominator  $(s_r + f_0)$  will equal zero, and the quadratic equation degenerates and the solution of the resulting linear equation is

$$r_1 = \frac{\kappa d^2}{2(\kappa f_1 - f_2)}$$
 and  $r_2 = \frac{d^2}{2(\kappa f_1 - f_2)}$ . (21.8)

Thus, by specifying a biarc radii ratio,  $\kappa$ , and applying Equation (21.7) or (21.8) we may calculate values for  $r_1$  and  $r_2$ . Equation (21.3) allows us to calculate the corresponding coordinates of the carrier circle centers,  $C_1$  and  $C_2$ . In certain configurations only one arc is needed to satisfy the position and tangent constraints. In this case, the equations will produce centers and radii for two identical circles.

### Number of Solutions

Given a specific pair of points,  $P_1$  and  $P_2$  and tangent vectors,  $\hat{T}_1$  and  $\hat{T}_2$ , we now consider how many different carrier circle pairs exist for a given radii ratio,  $\kappa$ . The solutions may be enumerated as follows:

- Equations (21.7) or (21.8) may produce negative values for  $r_1$  or  $r_2$ . The sign indicates the directions the center points  $C_1$  and  $C_2$  should be offset from  $P_1$  and  $P_2$ , respectively. As a result, both  $\kappa$  and  $-\kappa$  may produce valid biarcs with a specified radii ratio.
- The radius sign constant,  $s_r$ , may take on two different values,  $\pm 1$ .
- The quadratic formula yields two different solutions due to the sign preceding the radical sign as shown in Equation (21.7).

Therefore, as many as  $2 \times 2 \times 2 = 8$  unique carrier circle pairs may exist for a given biarc configuration and radii ratio.

## 21.2 Knot Point

Suppose we wish to compute the coordinates of the *knot point*,  $P_0(x_0, y_0)$ , which is the point of tangency between the two carrier circles. The coordinates of the knot point can be found by intersecting the two circles and taking advantage of the fact that they intersect in a single point yielding

$$x_{0} = \frac{(h_{1} + h_{2})d^{2} - (h_{1} - h_{2})R}{2d^{2}}$$

$$y_{0} = \frac{(k_{1} + k_{2})d^{2} - (k_{1} - k_{2})R}{2d^{2}}$$
(21.9)

where

$$d^2 = (h_1 - h_2)^2 + (k_1 - k_2)^2$$
 and  
 $R = r_1^2 - r_2^2.$ 

While it is a simple matter to compute the knot point once we have the two carrier circles, we might instead want to specify the position of the knot point. We will show in this section that the knot point cannot be selected arbitrarily, but must lie on one of two specific circles, called the *knot circles*. The following theorem and corollary from elementary geometry (which are stated without proof) will be central to exploring the nature of the knot circles.

**Theorem.** Angles at the circumference of a circle subtended by the same arc are equal.

**Corollary.** Given two fixed points  $P_1$  and  $P_2$  and a variable point Q, the locus of Q is a circle if  $\angle P_1 Q P_2$  is a constant.

Consider two internally tangent carrier circles centered at  $C_1(r_1, 0)$  and  $C_2(r_2, 0)$  and with radii  $r_1$  and  $r_2$ , respectively, as shown in Figure 21.2. By construction the two circles are internally tangent at the origin. Pick two arbitrary points  $P_1$  and  $P_2$  on the circles with coordinates

$$P_1(r_1 + r_1 \cos \theta_1, r_1 \sin \theta_1)$$
 and  $P_2(r_2 + r_2 \cos \theta_2, r_2 \sin \theta_2)$ .



Figure 21.2: Knot point angles.

We will show that the angle  $\alpha \equiv \angle P_1 O P_2$  is a constant angle for all such points  $P_1$  and  $P_2$  when the angle  $\omega$  is constant, and, having established this fact, we will apply the corollary stated above to establish that the knot point must be on one of two circles.

The slopes of lines  $OP_1$  and  $OP_2$ ,  $m_1$  and  $m_2$ , are given by

$$m_1 = \frac{r_1 \sin \theta_1}{r_1 + r_1 \cos \theta_1} = \frac{\sin \theta_1}{1 + \cos \theta_1} = \tan\left(\frac{1}{2}\theta_1\right)$$
$$m_2 = \frac{r_2 \sin \theta_2}{r_2 + r_2 \cos \theta_2} = \frac{\sin \theta_2}{1 + \cos \theta_2} = \tan\left(\frac{1}{2}\theta_2\right)$$

and the angle,  $\alpha$ , between  $OP_1$  and  $OP_2$  is

$$\tan \alpha = \frac{m_2 - m_1}{1 + m_1 m_2}$$

$$= \frac{\tan\left(\frac{1}{2}\theta_2\right) - \tan\left(\frac{1}{2}\theta_1\right)}{1 + \tan\left(\frac{1}{2}\theta_1\right) \tan\left(\frac{1}{2}\theta_2\right)}$$

$$= \tan\left(\frac{1}{2}(\theta_2 - \theta_1)\right)$$

$$= \pm \sqrt{\frac{1 - \cos(\theta_2 - \theta_1)}{1 + \cos(\theta_2 - \theta_1)}}.$$



Figure 21.3: Knot circles.

The angle  $\omega$  between the two tangent lines  $L_1$  and  $L_2$  is given by

$$(\theta_2 + \frac{\pi}{2}) - (\theta_1 + \frac{\pi}{2}) = \omega \text{ (or } \pi - \omega)$$
$$(\theta_2 + \frac{\pi}{2}) - (\theta_1 + \frac{\pi}{2}) = \omega \text{ (or } \pi - \omega).$$
$$(\theta_2 - \theta_1) = \omega \text{ (or } \pi - \omega).$$

Notice that since  $\omega$  is a constant, by definition, for any given biarc configuration,  $\theta_2 - \theta_1$  is also a constant. Since  $\alpha$  is a function of  $\theta_2 - \theta_1$  it must also be a constant. Now applying the corollary, the knot point must be on one of two circles corresponding to the two constant values of  $\alpha$ . Similar proofs hold for other geometric configurations and also for externally tangent circles.

# 21.3 Knot Circles

In the previous section it was established that all the valid knot points for a given biarc configuration must lie on either of two circles. These two circles can be constructed geometrically by considering the limiting cases of the carrier circle radii,  $r_1$  and  $r_2$ , as shown in Figure 21.3. First, notice that the tangency points  $P_1$  and  $P_2$  must be on the knot circles because they correspond to the knot point in the trivial cases when  $r_1 = 0$  and  $r_2 = 0$ . Now imagine a circle anchored at  $P_1$ , tangent to line  $L_1$ , and increasing in radius from zero. At some value of  $r_1$ , the circle will become tangent to line  $L_2$  at the point labeled  $P'_1$  (or  $P''_1$ , if the circle is on the opposite side of  $L_1$ ). At this value of  $r_1$ ,  $r_2$  will be infinite and the second biarc circle will become a line. The three points  $P_1$ ,  $P_2$  and  $P'_1$  determine the first knot circle, and  $P_1$ ,  $P_2$ , and  $P''_1$  determine the second knot circle.

Since the construction is symmetrical, we could have increased the radius of the second carrier circle,  $r_2$ , from zero and found points  $P'_2$  and  $P''_2$  which are on the same two knot circles

as those defined by  $P'_1$  and  $P''_1$ .

The center points of the knot circles can be constructed by intersecting the perpendicular bisector of  $P_1P_2$  with the angle bisectors of lines  $L_1$  and  $L_2$ . Using simple trigonometric relationships it can be shown that the radii of the two knot circles, r' and r'', are given by

$$r' = \frac{d}{2\cos\left(\frac{1}{2}\omega\right)}$$
 and  $r'' = \frac{d}{2\sin\left(\frac{1}{2}\omega\right)}$ 

where d is the distance between  $P_1$  and  $P_2$  and  $\omega$  is the angle between  $L_1$  and  $L_2$ .

## 21.4 Biarc Programming Examples

Descarta2D does not provide built-in functions for computing biarcs directly, so we will use the facilities available in Descarta2D along with the programming capabilities provided by Mathematica to demonstrate how new functions can be added to Descarta2D. In order to keep the examples simple, we will ignore special cases and possible error conditions. Better implementations would check for special cases and report errors in the input arguments when such errors are detected.

## Knot Circles

The first example illustrates a Mathematica function that will return a list of two knot circles given a biarc configuration (tangent points and tangent lines). For simplicity we will use a triangle to define the biarc configuration with the implicit understanding that the first and third vertices of the triangle,  $V_1$  and  $V_3$ , are the tangent points,  $P_1$  and  $P_2$ , and sides  $V_1V_2$  and  $V_2V_3$  of the triangle are the tangent lines,  $L_1$  and  $L_2$ . Using a triangle as an input parameter has the added advantage that many invalid cases are avoided because they would involve invalid triangles.

Lines 1, 2 and 3 define a *Mathematica* function called KnotCircles2D that takes one parameter that is required to pattern match a *Descarta2D* triangle object. Line 4 opens a Module statement that defines two local variables pt1 and pt2. Line 5 constructs a point at the center of a circle inscribed in the triangle. Line 6 constructs a point offset from the apex point, pA, to the tangency point, p1, a *negative* distance defined by p2 and pA. Lines 8 and 9 construct two circles from the two tangency points and a third point, pt1 or pt2. **Example.** Construct the two knot circles associated with the triangle whose vertices are (0,0), (2,1) and (3,0). Plot the geometric objects.

**Solution.** The function KnotCircles2D[*triangle*], defined above, returns a list of two knot circles associated with a triangle.

In[3]: Sketch2D[{p1, p2, p3, t1, kc, Point2D[t1, Inscribed2D]}];



**Descarta2D Hint.** In the KnotCircles2D function whose implementation is shown above, the third point on the second knot circle is constructed following the technique described earlier in the chapter. However, the third point of the first knot circle is constructed as the center of the circle inscribed in the triangle. The exploration knotin.nb at the end of the chapter shows that this point is actually on the first knot circle. Using this point has the added advantage that it avoids an error condition that would otherwise occur when  $P_1$  and  $P_2$  are equidistant from the third point of the triangle (an isosceles biarc configuration).

## **Arc Construction**

In this section we will implement functions for constructing bulge factor arcs given defining points and tangent vectors. These will be used later to construct biarcs. First, we define a utility function, Cross2D, that computes a vector cross-product in two dimensions.

 $In[4]: Cross2D[{u1_, v1_}, {u2_, v2_}] := u1 * v2 - u2 * v1;$ 

F

F

Now we define an arc construction function that takes the arc's start and end points,  $P_0$  and  $P_1$ , as input, plus a point associated with the start point indicating the direction of the tangent to the arc at the start point. The justification for this function is provided in the exploration arcentry.nb.

```
In[5]: Arc2D[{Point2D[p0: {x0_, y0_}], Point2D[p: {x_, y_}]},
Point2D[p1: {x1_, y1_}]] :=
Module[{v0 = p - p0, chd = p1 - p0, s, c},
s = Cross2D[v0, chd];
c = Dot[v0, chd];
Arc2D[p0, p1, s / (c + Sqrt[c^2 + s^2])]];
```

The next arc construction function is similar to the previous one, except the point indicating the tangent direction is associated with the end point of the arc. The justification for this function is provided in the exploration arcexit.nb.

```
In[6]: Arc2D[Point2D[p0: {x0_, y0_}],
        {Point2D[p1: {x1_, y1_}], Point2D[p: {x_, y_}]}] :=
        Module[{v1 = p - p1, chd = p1 - p0, s, c},
        s = Cross2D[chd, v1];
        c = Dot[chd, v1];
        Arc2D[p0, p1, s / (c + Sqrt[c^2 + s^2])]];
```

## **Knot Points**

Now that we have functions for computing knot circles and constructing arcs involving tangent vectors, it is fairly easy to construct biarcs. Consider the following *Mathematica* function:

The function Biarc2D takes two arguments, a *Descarta2D* triangle object defining the biarc configuration (lines 1–3), and a *Descarta2D* point object defining the knot point (line 4). The function Arc2D[{point, point}, point], defined in the previous section, is used twice to actually construct the biarc which is returned as a list of two arcs (lines 5–6).

**Example.** Construct the biarc associated with the triangle whose vertices are (0,0), (2,1) and (3,0) and a knot point on the first knot circle at parameter value  $\pi/2$ . Plot the geometric objects.

3

**Solution.** The function Biarc2D[*triangle*, *point*] described above returns a list of two arcs (a biarc) given a triangle that defines the biarc configuration and the knot point.

```
In[8]: t1 = Triangle2D[p1 = Point2D[{0, 0}],
             p2 = Point2D[{2, 1}],
             p3 = Point2D[{3, 0}]];
      kc = KnotCircles2D[t1] // N;
      bi1 = Biarc2D[t1, pk = Point2D[kc[[1]][Pi/2]]] // N
Out[8] {Arc2D[{1.5, 0.484234}, {0, 0}, 0.075838],
       Arc2D[{3., 0}, {1.5, 0.484234}, 0.241083]}
In[9]: Sketch2D[{pl, p2, p3, t1, pk, bi1}];
              1
           0.8
           0.6
           0.4
           0.2
              0
                         0.5
                                           1.5
                                                              2.5
                                    1
                                                       2
                 0
```

**Descarta2D Hint.** The Biarc2D function does not check to insure that the knot point provided is a valid knot point. It will erroneously return two arcs that are not tangent to each other if it is called with a point not on one of the knot circles. Other errors will occur if the knot point coincides with one of the triangle vertices.

It would be convenient if the biarc construction function computed the knot point internally as shown in the following *Mathematica* function:

```
In[10]: (*1*) Biarc2D[t1:Triangle2D[p0:{x0_, y0_},
       (*2*)
                       pA:{xA_, yA_},
       (*3*)
                       p1:{x1_, y1_}],
               knotCircleNumber_Integer,
       (*4*)
               knotPointParameter_?IsScalar2D] :=
       (*5*)
        (*6*) Module[{kc, ptK},
       (*7*)
               kc = KnotCircles2D[t1][[knotCircleNumber]];
               ptK = Point2D[kc[knotPointParameter]] // N;
       (*8*)
               Biarc2D[t1, ptK]];
       (*9*)
```

This Biarc2D function takes three arguments, the first being a triangle defining the biarc configuration, the second an integer equal to 1 or 2 specifying which biarc circle the knot point should be on, and the third an angle (in radians) specifying the parameter location on the knot circle for the desired knot point. Lines 1–5 define the function arguments, line 7 computes the knot circle, line 8 computes the knot point, and line 9 computes the biarc.

**Example.** Construct the biarc associated with the triangle whose vertices are (0,0), (2,1) and (3,0) and a knot point on the first knot circle at parameter value  $\pi/2$ .

Solution. The function

Biarc2D[triangle, knotCircleNumber, knotCircleParameter]

as implemented above returns the required biarc.

The Biarc2D functions implemented above are restrictive in the sense that the tangent vectors always point toward the apex point of the triangle and no provision is available to allow either (or both) of the tangent vectors to point away from the apex. In order to overcome this restriction we will implement a function that takes two line segments to define the biarc configuration. The line segments will define a position (the start point of the line segment) and a direction (from the start point towards the end point).

```
In[12]: (*1*) Biarc2D[Segment2D[p0:{x0_, y0_}, d0:{u0_, v0_}],
       (*2*)
                Segment2D[p1:{x1_, y1_}, d1:{u1_, v1_}],
                ptK:Point2D[{xk_, yk_}]] :=
       (*3*)
       (*4*) {Arc2D[{Point2D[p0], Point2D[d0]}, ptK],
       (*5*) Arc2D[ptK, {Point2D[p1], Point2D[d1]}];
In[13]: (*1*) Biarc2D[L0:Segment2D[p0:{x0_, y0_}, d0:{u0_, v0_}],
       (*2*)
               L1:Segment2D[p1:{x1_, y1_}, d1:{u1_, v1_}],
       (*3*)
               knotCircleNumber Integer,
       (*4*)
              knotCircleParameter ?IsScalar2D] :=
       (*5*) Module[{ptA, t1, kc, ptK},
       (*6*) ptA = Point2D[Line2D[L0], Line2D[L1]];
              t1 = Triangle2D[p0, Coordinates2D[ptA], p1];
       (*7*)
              kc = KnotCircles2D[t1][[knotCircleNumber]];
       (*8*)
              ptK = Point2D[kc[knotCircleParameter]];
       (*9*)
       (*10*) Biarc2D[L0, L1, ptK]];
```

These Biarc2D functions are parallel implementations of the previous two, except two line segments are used to define the biarc configuration instead of a triangle.

**Example.** Given a biarc configuration defined by the line segments from (0,0) to (-3,2) and from (3,0) to (4,-2) construct a set of biarcs whose knot points are on knot circle 1 at parameter values  $\pi/3$ ,  $\pi/2$  and  $2\pi/3$ .

Solution. Use the function Biarc2D whose implementation is provided above.

```
In[14]: ls0 = Segment2D[{0, 0}, {-3, 2}] //N;
ls1 = Segment2D[{3, 0}, {4, -2}] //N;
bi1 = Map[(Biarc2D[ls0, ls1, 1, #] //N)&, {Pi/6, Pi/2, 5Pi/6}];
```

In[15]: Map[(Sketch2D[{ls0, ls1, #}];)&, bi1];



## Building on these basic *Mathematica* programs for computing biarcs, more elaborate construction functions could be provided. For example, we might write a function that attempts to automatically select the knot point based on some minimization criteria; or we might attempt to construct biarcs that have no reversal of curvature at the knot point (i.e. the carrier circles are internally tangent to each other). Another interesting exercise is to devise a strategy for connecting a predefined set of points with a smooth, piecewise curve consisting of biarcs.

# 21.5 Explorations

INCENTER ON KNOT CIRCLE......knotin.nb Show that the *incenter* of a triangle (the center point of the circle inscribed in the triangle) is on one of the knot circles for the biarc configuration defined by the triangle. Part VI

Reference

# Chapter 22

# **Technical Notes**

This chapter provides an overview of how Descarta2D is implemented using Mathematica. Descarta2D is an object-oriented application, which means that it provides a collection of objects (e.g. points, lines, etc.) and a set of methods that compute on these objects. The programs that comprise Descarta2D are organized into a small number of Mathematica packages that specify the behavior of the objects.

## 22.1 Computation Levels

Mathematica provides support for both symbolic and numerical computations. Descarta2D takes advantage of these capabilities to provide the following four levels of computation:

- **Symbolic.** At the *symbolic* level, sizes, angles and coefficients are expressed as variables and general formulas may be derived.
- **Analytic.** At the *analytic* level variables are replaced with exact numerical quantities that are not approximated by floating point numbers. Mathematical functions such as square roots and trigonometric functions are carried without evaluation.
- **Numerical.** At the *numerical* level numbers and functions are replaced with floating point representations that are approximations carried to any number of decimal places in *Mathematica*. Often, the accuracy is determined by the floating point hardware available in the computer and is sufficient for such computations.
- **Approximation.** At the *approximation* level iterative algorithms are used to converge to an approximation of a value. Generally, the tolerance of the approximation can be controlled to approach the floating point precision of the computer hardware or better.

Depending on the complexity of the problem, *Descarta2D* often provides a choice of the level of computation undertaken for a particular geometric investigation.

Table 22.1: Reserved names in Descarta<sub>2D</sub> (objects).

| Arc2D      | Hyperbola2D | Quadratic2D |
|------------|-------------|-------------|
| Circle2D   | Line2D      | Segment2D   |
| ConicArc2D | Parabola2D  | Triangle2D  |
| Ellipse2D  | Point2D     |             |

## 22.2 Names

In Mathematica symbolic names containing upper case letters are considered different than names using corresponding lower case letters (i.e. Mathematica is case-sensitive with respect to the interpretation of symbolic names). Mathematica uses the convention that system-defined names always begin with upper case letters or the dollar sign symbol and recommends that user-defined symbols begin with lower case letters to avoid naming conflicts. Descarta2D follows the same naming conventions as Mathematica. Descarta2D symbolic names begin with upper case letters; user symbolic names may contain upper case or lower case letters, but, generally, the first letter is advised to be lower case to prevent conflicts with built-in Mathematica functions and Descarta2D functions. In order to prevent conflicts with the names of Descarta2D functions, this chapter provides suggestions for naming Descarta2D objects to encourage consistency and clear understanding when using Descarta2D. Following these conventions will avoid most naming conflicts.

All Descarta2D function names are fully spelled out English words and each name has the ending 2D appended. If more than one word is used (for example, TangentConics2D, or FocalLength2D), then the first letter of each word is upper case.

Descarta2D adheres to several syntactic conventions for consistency and ease of use. Functions that return (construct) a Descarta2D object, such as Point2D, will always have the form

## $objectName[arg_1, arg_2, ...].$

For example, Point2D[*point*, *point*] returns the midpoint of two given points. Functions that return a list of objects are generally plural, such as Points2D, TangentLines2D and Foci2D.

If Descarta2D detects invalid input when constructing an object, it generally displays an error message and returns the **\$Failed** symbol. Descarta2D functions that return a list of objects will generally return an empty list, instead of the **\$Failed** symbol (indicating that no objects can be constructed).

## 22.3 Descarta2D Objects

In Descarta2D an object is a textual representation of a mathematical concept. Each object is represented using a Mathematica expression whose head is the name of the object and whose parameters are the arguments of the expression. Table 22.1 is a list of the object names

$$\begin{array}{c} object\\ geometry\\ \hline coordinates & \{x, y\}\\ \hline point & Point2D[\{x, y\}]\\ \hline curve\\ \hline line & Line2D[A, B, C]\\ \hline circle & Circle2D[\{h, k\}, r]\\ \hline parabola & Parabola2D[\{h, k\}, f, \theta]\\ \hline ellipse & Ellipse2D[\{h, k\}, a, b, \theta]\\ \hline quadratic & Quadratic2D[A, B, C, D, E, F]\\ \hline curve segment\\ \hline line segment & Segment2D[\{x_0, y_0\}, \{x_1, y_1\}, B]\\ \hline conic arc & ConicArc2D[\{x_0, y_0\}, \{x_A, y_A\}, \{x_1, y_1\}, \rho]\\ \hline composite\\ \hline triangle & Triangle2D[\{x_1, y_1\}, \{x_2, y_2\}, \{x_3, y_3\}]\\ \hline polynomial\\ \hline linear & Line2D[a, b, c]\\ ax^2 + bxy + cy^2 + dx + ey + f\end{array}$$

Figure 22.1: Descarta2D object hierarchy.



Figure 22.2: Standard representation of an Arc2D.

built into Descarta2D. The objects are organized into a hierarchy as shown in Figure 22.1. The hierarchy also includes meta-objects, objects that have no implementation, but serve to organize the Descarta2D objects. Meta-objects are shown in *italic* font. The following sections provide detailed descriptions of each object provided by Descarta2D. Each section provides the name of the object, the syntax of the Mathematica expression for the object, names typically used to refer to the object, a description of the object (using a mathematical equation when appropriate) and restrictions on the arguments of the object. All objects have the restriction that their arguments cannot involve complex numbers. The Line2D and Quadratic2D objects are listed twice in Figure 22.1 because they can be interpreted to represent geometry or polynomials.

#### Arc2D

Arc2D[ $\{x_0, y_0\}$ ,  $\{x_1, y_1\}$ , B] is the standard representation of an arc in Descarta2D as illustrated in Figure 22.2. The first and second arguments are the coordinates of the start and end points of the arc, respectively. The third argument is a positive scalar, B, representing the bulge factor of the arc. The bulge factor is the ratio of the arc's height, H, to half the chord length, D/2; so B = 2H/D. The arc is traversed counter-clockwise from  $P_0$  to  $P_1$ .

In the argument sequence of a function an arc is shown as *arc*, as in Radius2D[*arc*]. *arc*[0] gives the coordinates of the start point, *arc*[1] gives the coordinates of the end point and Bulge2D[*arc*] gives the bulge factor. Suggested symbolic names for an Arc2D include the series: (a1, a2, ...), (A1, A2, ...) and (arc1, arc2, ...).

The parametric equations of an Arc2D using parameter t are

$$x(t) = h + (x_0 - h)\cos(\beta t) - (y_0 - k)\sin(\beta t)$$



Figure 22.3: Standard representation of a Circle2D.

$$y(t) = k + (x_0 - h)\sin(\beta t) + (y_0 - k)\cos(\beta t)$$

where (h, k) is the center point of the arc, and  $\beta$  is the angular span of the arc. Both the center point and the angular span are functions of the defining points and the bulge factor as described in the "Arcs" chapter. Values of t in the range  $0 \le t \le 1$  generate coordinates on the complete span of the arc.  $\operatorname{Arc2D}[\{x_0, y_0\}, \{x_1, y_1\}, B]$  [t] returns the coordinates of a point on an arc at parameter t. The expression  $\operatorname{Arc2D}[\{x_0, y_0\}, \{x_1, y_1\}, B]$  [ $t_1, y_1$ ], B] [ $t_1, t_2$ ] when used in a plotting command, such as Sketch2D, will cause the portion of the arc between parameters  $t_1$  and  $t_2$  to be plotted.

#### Circle2D

Circle2D[ $\{h, k\}$ , r] is the standard representation of a circle in *Descarta2D* as illustrated in Figure 22.3. The center of the circle is given as a coordinate list,  $\{h, k\}$ , and the radius is the positive scalar, r. The equation of the circle is

$$(x-h)^2 + (y-k)^2 = r^2$$
.

In the argument sequence of a function, a Circle2D is shown as *circle* or *cir*, as in Radius2D[*circle*] or Radius2D[*cir*]. The function Coordinates2D[*circle*] gives the center point coordinates of a circle and Radius2D[*circle*] gives the radius, r. Suggested symbolic names for a Circle2D include the series: (c1, c2, ...), (C1, C2, ...) and (cir1, cir2, ...).

The parametric equations of a Circle2D using parameter  $\boldsymbol{\theta}$  are

$$\begin{aligned} x(\theta) &= h + r \cos \theta \\ y(\theta) &= k + r \sin \theta. \end{aligned}$$

Values of  $\theta$  in the range  $0 \le \theta < 2\pi$  generate coordinates on the complete circumference of the circle. Circle2D[{h, k}, r][ $\theta$ ] returns the coordinates of the point on a circle at parameter



Figure 22.4: Standard representation of a ConicArc2D.

 $\theta$ . The expression Circle2D[{h, k}, r][ $\theta$ ][{ $\theta_1, \theta_2$ }] when used in a plotting command, such as Sketch2D, will cause the arc of the circle between parameters  $\theta_1$  and  $\theta_2$  to be plotted.

#### Conic Arc

ConicArc2D[ $\{x_0, y_0\}$ ,  $\{x_A, y_A\}$ ,  $\{x_1, y_1\}$ ,  $\rho$ ] is the standard representation of a conic arc in *Descarta2D* as illustrated in Figure 22.4. The first and third arguments are the coordinates of the start and end points of the conic arc, respectively. The second argument is the coordinates of the apex point of the conic arc (the apex point is the intersection point of the tangent lines at the start and end points). The fourth argument,  $\rho$ , is a scalar representing the projective discriminant of the conic arc. Values of  $\rho$  in the range  $0 < \rho < 1/2$  are elliptical arcs; values in the range  $1/2 < \rho < 1$  are hyperbolic arcs; and the value 1/2 is a parabolic arc.

In an argument sequence, a ConicArc2D is shown as *cnarc*, as in Rho2D[*cnarc*]. The function Coordinates2D[*cnarc*, Apex2D] returns the coordinates of the apex point of a conic arc and Rho2D[*cnarc*] gives the value of  $\rho$ . ConicArc2D[{ $x_0, y_0$ }, { $x_A, y_A$ }, { $x_1, y_1$ },  $\rho$ ][t] with t = 0 gives the coordinates of the start point and with t = 1 gives the coordinates of the end point. Suggested symbolic names for a ConicArc2D include the series: (ca1, ca2, ...), (CA1, CA2, ...) and (cnarc1, cnarc2, ...).

The parametric equations of a ConicArc2D using parameter t are

$$\begin{aligned} x(t) &= \frac{b_0(1-\rho)x_0 + b_1\rho x_A + b_2(1-\rho)x_1}{b_0(1-\rho) + b_1\rho + b_2(1-\rho)} \\ y(t) &= \frac{b_0(1-\rho)y_0 + b_1\rho y_A + b_2(1-\rho)y_1}{b_0(1-\rho) + b_1\rho + b_2(1-\rho)} \end{aligned}$$

where  $b_0 = (1-t)^2$ ,  $b_1 = 2t(1-t)$  and  $b_2 = t^2$ . Values of t in the range  $0 \le t \le 1$  generate coordinates on the complete span of the conic arc. The expression

ConicArc2D[{
$$x_0, y_0$$
}, { $x_A, y_A$ }, { $x_1, y_1$ },  $\rho$ ][{ $t_1, t_2$ }]

when used in a plotting command, such as Sketch2D, will cause the portion of the conic arc between parameters  $t_1$  and  $t_2$  to be plotted.



Figure 22.5: Standard representation of an Ellipse2D.

## Coordinates

Coordinates  $\{x, y\}$  are used to represent an (x, y) position in Descarta2D. In an argument sequence coordinates are shown as *coords* such as Point2D[*coords*], or in explicit forms such as  $\{h, k\}$  or  $\{x, y\}$  as in Point2D[ $\{x, y\}$ ]. Suggested symbolic names for coordinates include the series: (c1, c2, ...), (C1, C2, ...) and (coords1, coords2, ...).

## Ellipse2D

Ellipse2D[ $\{h, k\}$ ,  $a, b, \theta$ ] is the standard representation of an ellipse in *Descarta2D* as illustrated in Figure 22.5. The first argument,  $\{h, k\}$ , is a list of coordinates representing the center of the ellipse. The second argument is a positive scalar, a, representing the length of the semi-major axis. The third argument is a positive scalar, b, representing the length of the semi-minor axis. In a valid ellipse, the length of the semi-major axis must be greater than the length of the semi-minor axis, a > b. The fourth argument,  $\theta$ , is the angle of rotation of the ellipse measured from the +x-axis counter-clockwise to the major axis of the ellipse and is normalized to the range  $0 \le \theta < \pi$ . The underlying equation of the (non-rotated) ellipse is

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1.$$

In an argument sequence, an ellipse is shown as *ellipse*, as in Angle2D[*ellipse*]. The function Coordinates2D[*ellipse*] returns the center point coordinates of an ellipse; the function

#### SemiMajorAxis2D[ellipse]

gives the length of the semi-major axis, a, and the function



Figure 22.6: Standard representation of a Hyperbola2D.

#### SemiMinorAxis2D[ellipse]

gives the length of the semi-minor axis, b; and Angle2D[*ellipse*] gives the angle of rotation,  $\theta$ . Suggested symbolic names for an Ellipse2D include the series: (e1, e2, ...), (E1, E2, ...) and (ell1, ell2, ...).

The parametric equations of a (non-rotated) Ellipse2D using the parameter  $\alpha$  are

$$\begin{aligned} x(\alpha) &= h + a \cos \alpha \\ y(\alpha) &= k + b \sin \alpha. \end{aligned}$$

Values of  $\alpha$  in the range  $0 \le \alpha < 2\pi$  generate coordinates on the complete circumference of the ellipse. Ellipse2D[{h, k},  $a, b, \theta$ ][ $\alpha$ ] returns the coordinates of the point on an ellipse at parameter  $\alpha$ . The expression Ellipse2D[{h, k},  $a, b, \theta$ ][ $\alpha$ ],  $\alpha_2$ }] when used in a plotting command, such as Sketch2D, will cause an arc of the ellipse between parameters  $\alpha_1$  and  $\alpha_2$  to be plotted.

### Hyperbola2D

Hyperbola2D[ $\{h, k\}$ ,  $a, b, \theta$ ] is the standard representation of a hyperbola in *Descarta2D* as illustrated in Figure 22.6. The first argument,  $\{h, k\}$ , is a list of coordinates representing the center of the hyperbola. The second argument is a positive scalar, a, representing the length of the semi-transverse axis. The third argument is a positive scalar, b, representing the length of the semi-conjugate axis. The fourth argument,  $\theta$ , is the angle of rotation of the hyperbola measured from the +x-axis counter-clockwise to the transverse axis of the hyperbola and is normalized to the range  $0 \le \theta < \pi$ . The underlying equation of the (non-rotated) hyperbola

is

$$\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1.$$

In an argument sequence, a hyperbola is shown as *hyperbola*, as in Angle2D[*hyperbola*]. The function Coordinates2D[*hyperbola*] returns the center point coordinates of a hyperbola; the function

#### SemiTransverseAxis2D[hyperbola]

gives the length of the semi-transverse axis, a, and the function

#### SemiConjugateAxis2D[hyperbola]

gives the length of the semi-conjugate axis, b; and Angle2D[hyperbola] gives the angle of rotation,  $\theta$ . Suggested symbolic names for a Hyperbola2D include the series: (h1, h2, ...), (H1, H2, ...) and (hyp1, hyp2, ...).

The parametric equations of a (non-rotated) Hyperbola2D using parameter t are

$$\begin{aligned} x(t) &= h + a \cosh(st) \\ y(t) &= k + b \sinh(st) \end{aligned}$$

where  $s = \cosh^{-1}(e)$  and e is the eccentricity of the hyperbola. Values of t in the range  $-\infty < t < \infty$  generate coordinates on the branch of the hyperbola opening to the right in the non-rotated position. Hyperbola2D[ $\{h, k\}, a, b, \theta$ ][t] returns the coordinates of the point on a hyperbola at parameter t. The values  $t = \pm 1$  generate coordinates at the ends of the focal chord of the hyperbola. The expression Hyperbola2D[ $\{h, k\}, a, b, \theta$ ][t],  $t_2$ }] when used in a plotting command, such as Sketch2D, will cause an arc of the hyperbola between parameters  $t_1$  and  $t_2$  to be plotted. If  $t_1 < t_2$ , the arc will be on the right branch of the (non-rotated) hyperbola; if  $t_1 > t_2$ , the arc will be on the left branch of the (non-rotated) hyperbola.

#### Line2D

Line2D[A, B, C] is the standard representation of an infinite line Ax + By + C = 0. At least one of the first two coefficients, A or B, must be non-zero. The parametric equations of a line using parameter t are

$$x(t) = ac + bt$$
 and  $y(t) = bc - at$ 

where

$$a = \frac{A}{\sqrt{A^2 + B^2}}, \ b = \frac{B}{\sqrt{A^2 + B^2}}$$
 and  $c = \frac{C}{\sqrt{A^2 + B^2}}$ 

The coordinates of the point on the line nearest the origin will be at parameter t = 0 and other coordinates, parameterized by distance  $t, -\infty < t < \infty$ , along the line are given by Line2D[A, B, C] [t]. The expression Line2D[A, B, C] [ $\{t_1, t_2\}$ ] when used in a plotting command, such as Sketch2D, will cause a segment of the line between parameters  $t_1$  and  $t_2$  to be plotted.



Figure 22.7: Standard representation of a Parabola2D.

In an argument sequence, a Line2D object is shown as *line* or *ln*, as in Angle2D[*line*] or Angle2D[*ln*]. Suggested symbolic names for a Line2D include the series: (11, 12, ...), (L1, L2, ...) and (ln1, ln2, ...).

### Parabola2D

**Parabola2D**[ $\{h, k\}, f, \theta$ ] is the standard representation of a parabola in *Descarta2D* as illustrated in Figure 22.7. The first argument,  $\{h, k\}$ , is a list of coordinates representing the vertex of the parabola. The second argument is a positive scalar, f, representing the focal length of the parabola. The third argument,  $\theta$ , is the angle of rotation of the parabola measured from the +x-axis counter-clockwise to the axis of the parabola and is normalized to the range  $0 \le \theta < 2\pi$ . The underlying equation of the (non-rotated) parabola is

$$(y-k)^2 = 4f(x-h).$$

In an argument sequence, a parabola is shown as *parabola*, as in Angle2D[*parabola*]. The function Coordinates2D[*parabola*] returns the vertex point coordinates of the parabola; FocalLength2D[*parabola*] gives the focal length of the parabola; and Angle2D[*parabola*] gives the angle of rotation,  $\theta$ . Suggested symbolic names for a Parabola2D include the series: (p1, p2, ...), (P1, P2, ...) and (pb1, pb2, ...).

The parametric equations of a Parabola2D using parameter t are

$$x(t) = h + ft^2$$
 and  $y(t) = k + 2ft$ .

Values of t in the range  $-\infty < t < \infty$  generate coordinates on the parabola opening to the right in the non-rotated position. Parabola2D[{h, k}, f,  $\theta$ ] [t] returns the coordinates of the point on a parabola at parameter t. The values  $t = \pm 1$  generate coordinates at the ends of the

focal chord of the parabola. The expression Parabola2D[ $\{h, k\}$ ,  $f, \theta$ ][ $\{t_1, t_2\}$ ] when used in a plotting command, such as Sketch2D, will cause an arc of the parabola between parameters  $t_1$  and  $t_2$  to be plotted.

### Point2D

Point2D[ $\{x, y\}$ ] (which is the same as Point2D[*coords*]) is the standard representation of a point. The coordinates define the (x, y) position of the point. In an argument sequence, a point is shown as *point* or *pt*, as in Coordinates2D[*point*] and Coordinates2D[*pt*]. Suggested symbolic names for a Point2D include the series: (p1, p2, ...), (P1, P2, ...) and (pt1, pt2, ...).

**XCoordinate2D**[*point*] and **YCoordinate2D**[*point*] return the x- and y-coordinate, respectively, of a point. **Coordinates2D**[*point*] returns the (x, y) coordinates of a point as a coordinate list.

#### Quadratic2D

Quadratic2D[A, B, C, D, E, F] is the standard representation of the quadratic

$$Ax^{2} + Bxy + Cy^{2} + Dx + Ey + F = 0.$$

At least one of the first five coefficients must be non-zero. In general, the quadratic will represent a conic curve, but certain combinations of coefficients may represent degenerate conics (lines and points) or no locus at all. In an argument sequence, a Quadratic2D is shown as quad, as in Point2D[quad]. Suggested symbolic names for a Quadratic2D include the series: (q1, q2, ...), (Q1, Q2, ...) and (quad1, quad2, ...). Descarta2D provides no parametric representation for a quadratic (the specific conics have parametric representations).

#### Segment2D

The form Segment2D[ $\{x_0, y_0\}$ ,  $\{x_1, y_1\}$ ] is the standard representation of a line segment in *Descarta2D*. The coordinates  $\{x_0, y_0\}$  and  $\{x_1, y_1\}$  are the start and end coordinates, respectively, of the line segment.

In an argument sequence, a Segment2D is shown as *lnseg*, as in Angle2D[*lnseg*]. Suggested symbolic names for a Segment2D include the series: (11, 12, ...), (L1, L2, ...) and (lnseg1, lnseg2, ...).

The parametric equations of a Segment2D using parameter t are

$$\begin{aligned} x(t) &= x_0 + t(x_1 - x_0) \\ y(t) &= y_0 + t(y_1 - y_0). \end{aligned}$$

Values of t in the range  $0 \le \theta \le 1$  generate coordinates over the complete length of the line segment. Segment2D[ $\{x_0, y_0\}$ ,  $\{x_0, y_0\}$ ][t] returns the coordinates of the point on a line segment at parameter t. The parameter value t = 0 generates the coordinates of the start point of the line segment and the value t = 1 generates the end point coordinates. The



Figure 22.8: Standard representation of a Triangle2D.

expression Segment2D[ $\{x_0, y_0\}$ ,  $\{x_1, y_1\}$ ][ $\{t_1, t_2\}$ ] when used in a plotting command, such as Sketch2D, will cause a portion of the line segment between parameters  $t_1$  and  $t_2$  to be plotted.

#### Triangle2D

Triangle2D[ $\{x_1, y_1\}$ ,  $\{x_2, y_2\}$ ,  $\{x_3, y_3\}$ ], as illustrated in Figure 22.8, is the standard representation of a triangle with vertex points  $(x_1, y_1)$ ,  $(x_2, y_2)$  and  $(x_3, y_3)$ . The vertex points cannot be coincident or collinear. Coordinates2D[*triangle*, n] returns the coordinates of vertex n of a triangle, n = 1, 2, 3.

In an argument sequence a Triangle2D is shown as *triangle*, as in Area2D[*triangle*]. Suggested symbolic names for a Triangle2D include the series: (t1, t2, ...), (T1, T2, ...) and (tri1, tri2, ...).

## **Object Queries**

Each object in *Descarta2D* responds to a set of special queries essential to the operation of *Descarta2D*. These special queries are listed below:

IsDisplay2D returns True if the object can be displayed using the Sketch2D command.

Is2D returns True if the object's head is in a given list.

IsValid2D returns True if the object is a *Descarta2D* object and each of its parameters is of the proper type and form.

ObjectNames2D returns a list of strings that are the names of all Descarta2D objects.

| D2DArc2D         | D2DHyperbola2D | D2DQuadratic2D      |
|------------------|----------------|---------------------|
| D2DArcLength2D   | D2DIntersect2D | D2DSegment2D        |
| D2DArea2D        | D2DLine2D      | D2DSketch2D         |
| D2DCircle2D      | D2DLoci2D      | D2DSolve2D          |
| D2DConic2D       | D2DMaster2D    | D2DTangentCircles2D |
| D2DConicArc2D    | D2DMedial2D    | D2DTangentConics2D  |
| D2DEllipse2D     | D2DNumbers2D   | D2DTangentLines2D   |
| D2DEquations2D   | D2DParabola2D  | D2DTangentPoints2D  |
| D2DExpressions2D | D2DPencil2D    | D2DTransform2D      |
| D2DGeometry2D    | D2DPoint2D     | D2DTriangle2D       |

Table 22.2: Reserved names in Descarta<sub>2D</sub> (packages).

If you add a new object to *Descarta2D*, the object will need to respond properly to these queries if you desire that the object behave in an integrated manner. Refer to the source code listings to determine how each of these queries can be implemented.

## 22.4 Descarta2D Packages

A relatively small number of *Mathematica* packages (\*.m files) provide support for all the *Descarta2D* functions. Your computer system must be set up to allow *Mathematica* to find these files before you can use any of the *Descarta2D* functions. In order to set up *Mathematica* to use the *Descarta2D* package files, you need copy the folder Descarta2D from the *Descarta2D* CD-ROM onto your hard drive. Descarta2D must be copied into a folder that *Mathematica* searches when loading packages. Typically, in a standard *Mathematica* installation, this will be the folder

c:\Program Files\Wolfram Research\Mathematica\3.0\AddOns\Applications\

although this directory path may be different for your installation of *Mathematica*. The master package file for *Descarta2D* will be loaded by issuing the command <<Descarta2D<sup>c</sup>. After this command is entered, *Mathematica* will automatically find and load all the packages as required to execute *Descarta2D* commands.

The package names are listed in Table 22.2. Each package defines symbols that are then *owned* by the package. The definitions in each package provide either support for *Descarta2D* objects (such as Point2D, Line2D, Circle2D, etc.) or functions (such as Radius2D[*circle*] that returns the radius of a circle, or Line2D[*point*, *point*] that constructs a line between two points).

| Angle2D           | MedialLoci2D        | SemiTransverseAxis2D   |
|-------------------|---------------------|------------------------|
| ArcLength2D       | Parameters2D        | SimplifyCoefficients2D |
| Area2D            | Perimeter2D         | Sketch2D               |
| Asymptotes2D      | Points2D            | Slope2D                |
| Bulge2D           | Polynomial2D        | SolveTriangle2D        |
| Centroid2D        | PrimaryAngle2D      | Span2D                 |
| Circumference2D   | PrimaryAngleRange2D | TangentCircles2D       |
| Coordinates2D     | Radius2D            | TangentConics2D        |
| Directrices2D     | Reflect2D           | TangentEquation2D      |
| Distance2D        | ReflectAngle2D      | TangentLines2D         |
| Eccentricity2D    | Rho2D               | TangentPoints2D        |
| Equation2D        | Rotate2D            | TangentQuadratics2D    |
| FocalChords2D     | Scale2D             | TangentSegments2D      |
| FocalLength2D     | SectorArea2D        | Translate2D            |
| Foci2D            | SegmentArea2D       | Vertices2D             |
| Length2D          | SemiConjugateAxis2D | XCoordinate2D          |
| Loci2D            | SemiMajorAxis2D     | YCoordinate2D          |
| MedialEquations2D | SemiMinorAxis2D     |                        |

Table 22.3: Reserved names in *Descarta2D* (general functions).

# 22.5 Descarta2D Functions

Descarta<sub>2D</sub> provides geometric computing facilities by introducing a number of general functions and queries whose names are listed in Table 22.3 and 22.4.

Some Descarta2D functions require keywords. A complete list of keywords is listed in Table 22.5.

There are a number of low-level functions provided by *Descarta2D* that are useful for implementing new functions. These are listed in Table 22.6.

Table 22.4: Reserved names in Descarta2D (general queries).

| IsCoincident2D | IsConcurrent2D | IsPerpendicular2D  |
|----------------|----------------|--------------------|
| IsCollinear2D  | IsOn2D         | IsTangent2D        |
| IsConcentric2D | IsParallel2D   | IsTripleParallel2D |

Table 22.5: Reserved names in Descarta2D (keywords).

| Apex2D          | Conjugate2D  | Parallel2D      |
|-----------------|--------------|-----------------|
| Circumscribed2D | Inscribed2D  | Pencil2D        |
| Complement2D    | MaxSeconds2D | Perpendicular2D |

# 22.6 Descarta2D Documentation

This entire book, including the subject matter chapters, reference chapters and exploration notebooks, is provided on the CD-ROM in two formats. The first format provided is Adobe's Portable Document Format (PDF) and is useful for passive reading and printing of typeset renderings of the book identical to the printed version of the book. PDF files may be viewed and printed using Adobe's Acrobat Reader program that can be downloaded at no charge from Abode's web site, www.adobe.com. The PDF files can be read directly off the CD-ROM or they can be copied into any convenient location on your disk drive.

The second format provided is Wolfram Research's *Mathematica* notebook format. Notebooks can be viewed interactively using *Mathematica* itself, or in a passive manner using Wolfram's *MathReader* program. *MathReader* is available at no charge and can be downloaded from Wolfram's web site, www.wolfram.com. Both *Mathematica* and *MathReader* can print notebook files.

Assuming that the Descarta2D folder has been copied from the CD-ROM into the appropriate Mathematica folder on your system, the documentation notebooks can be fully integrated into the Mathematica Help Browser. This is accomplished by clicking the Mathematica Front End menu item Help, Rebuild Help Index.... After the index has been rebuilt, all of the text and documentation for the book will be available interactively in the Help Browser. The material will be listed by clicking the AddOn button in the Help Browser and then selecting the item named Descarta2D. Notebooks whose links are clicked in the Help Browser category listing windows will be opened in the Help Browser; links clicked in the notebooks themselves

| AskCurveLength2D | IsDisplay2D       | IsZero2D           |
|------------------|-------------------|--------------------|
| ChopImaginary2D  | IsNegative2D      | IsZeroOrNegative2D |
| CurveLength2D    | IsNumeric2D       | MakePrimitives2D   |
| CurveLimits2D    | IsReal2D          | ObjectNames2D      |
| D2DPath2D        | IsScalar2D        | SetDisplay2D       |
| Is2D             | IsScalarPair2D    | Solve2D            |
| IsApproximate2D  | IsTinyImaginary2D |                    |
| IsComplex2D      | IsValid2D         |                    |

Table 22.6: Reserved names in Descarta2D (low-level functions).

will be opened at live notebooks in the Front End.

For Windows systems, installation files for Adobe Arcrobat Reader and MathReader are provided on the CD-ROM.

# Chapter 23

# **Command Browser**

This chapter is an alphabetical listing of all the commands provided by *Descarta2D*. The syntax and usage of each command is described, as well as notes outlining special options and defaults. Additionally, a cross-reference pointing to related commands is provided in each section. Commands described as being *low-level* are used in the internal implementation of *Descarta2D*. Low-level commands may be used freely, but they are not generally mentioned in the subject matter chapters of the book. Page numbers enclosed in square brackets indicate the page in the package listings where the implementation of the command is found.

### ■ Angle2D

Angle2D[*line*] computes the angle a line makes with the +x-axis, when measured counter-clockwise from the +x-axis to the line. [460]

Angles in Descarta2D are always specified and returned in radians.

Angle2D[arc] computes the angular span of an arc. [389]

Angle2D[*arc*, t] computes the angle a radial diameter passing through the point at parameter t on the arc makes with the +x-axis. [389]

Angle2D[*conic*] returns the rotation angle of a conic curve. The conic may be an ellipse [423], hyperbola [448] or parabola [481].

Angle2D[*line*, *line*] computes the angle measured counter-clockwise from the first line to the second line. [460]

Angle2D[triangle, n] computes the vertex angle at vertex n of a triangle. [547]

See also: PrimaryAngle2D, PrimaryAngleRange2D, ReflectAngle2D.

## ∎ Apex2D

Apex2D is a keyword indicating the construction of the apex control point of a conic arc. [419]

See also: Point2D.

## Arc2D

Arc2D[ $\{x_0, y_0\}$ ,  $\{x_1, y_1\}$ , B] is the standard representation of an arc. The coordinates of the start point are  $(x_0, y_0)$  and the coordinates of the end point are  $(x_1, y_1)$ . The bulge factor is the positive number B. The arc is traversed counter-clockwise from the start point to the end point. [387]

The bulge factor, B, is the ratio of the arc's height, h, to half the chord length, d/2; so B = 2h/d.

Arc2D[ $\{x_0, y_0\}$ ,  $\{x_1, y_1\}$ , B] [t] and arc[t] return the  $\{x, y\}$  coordinates of a point at parameter t on an arc. Parameter values in the range  $0 \le t \le 1$  produce coordinates covering the complete span of the arc. [388]

 $Arc2D[\{x_0, y_0\}, \{x_1, y_1\}, B][\{t_1, t_2\}]$  produces graphics primitives for a portion of an arc between parameters  $t_1$  and  $t_2$  when plotted. [388]

Arc2D[arc, Complement2D] constructs the complement of an arc. [391]

Arc2D[*point*, r, { $\theta_1$ ,  $\theta_2$ }] constructs an arc from the center point, radius and span. The angles,  $\theta_1$  and  $\theta_2$ , are measured counter-clockwise from the +x-axis. [392]

 $\operatorname{Arc2D}[point, r, \{point_1, point_2\}]$  constructs an arc from the center point, radius and the start and end points of the span. The start/end points do not need to lie on the arc, although they cannot be coincident with the center. [393]

Arc2D[point, point] constructs an arc passing through three points. The first and third points define the span of the arc. [393]

 $Arc2D[{point, \theta}, point]$  constructs an arc from a start point with entry angle and an end point. [392]

See also: Bulge2D, Complement2D.

### ArcLength2D

ArcLength2D[curve,  $\{t_1, t_2\}$ ] computes the arc length of a curve between two parameter values.

The curve may be an arc [396], circle [396], ellipse [397], hyperbola [397], line [397], line segment [397] or parabola [398].

 $N[ArcLength2D[cnarc, {t_1, t_2}]]$  numerically computes the arc length of a conic arc between two parameter values. [396]

See also: Circumference2D, Length2D, Perimeter2D, Span2D.

#### 🗖 Area2D

Area2D[curve] computes the area associated with a curve.
The curve may be an arc, circle, conic arc, ellipse or triangle.

Area2D[arc] computes the area between an arc and its chord. [399]

Area2D[circle] computes the area of a circle. [400]

Area2D[cnarc] computes the area between a conic arc and its chord. [401]

Area2D[ellipse] computes the area of an ellipse. [401]

Area2D[triangle] computes the area of a triangle. [403]

See also: SectorArea2D, SegmentArea2D.

# ■ AskCurveLength2D

AskCurveLength2D[] is a low-level function that returns the value of the CurveLength2D option of the Sketch2D command. [513]

See also: CurveLength2D, Sketch2D.

### Asymptotes2D

Asymptotes2D[hyperbola] constructs a list containing the two asymptote lines of a hyperbola. [413]

#### ■ Bulge2D

Bulge2D[arc] returns the bulge factor of an arc. [390]

See also: Arc2D.

# Centroid2D

Centroid2D is a keyword indicating the construction of a triangle's centroid point. [551]

See also: Point2D.

#### ■ ChopImaginary2D

ChopImaginary2D[*expr*, tol] is a low-level function that removes insignificant imaginary parts of complex numbers in an expression. The imaginary part is considered insignificant if its absolute value is less than the tolerance. [477]

The tolerance, if omitted, defaults to  $10^{-10}$ .

#### ■Circle2D

Circle2D[ $\{h, k\}, r$ ] is the standard representation of a circle. The coordinates of the center point of the circle are  $\{h, k\}$  and the radius is r. [405]

Circle2D[{h, k}, r][ $\theta$ ] and *circle*[ $\theta$ ] return the {x, y} coordinates of a point at parameter  $\theta$  on a circle. Parameter values in the range  $0 \le \theta < 2\pi$  produce coordinates covering the complete circumference of the circle. [406]

Circle2D[{h, k}, r][{ $\theta_1, \theta_2$ }] produces graphics primitives for the arc of the circle between parameters  $\theta_1$  and  $\theta_2$  when plotted. [406]

Circle2D[arc] constructs the circle underlying an arc. [391]

Circle2D[circle, circle, k, Pencil2D] constructs a circle, parameterized by the variable k, that represents the family (pencil) of circles passing through the intersection points of the two given circles. The family of circles is valid even if the two circles do not intersect as they will share a common radical axis. [486]

Circle2D[*lnseg*] constructs the circle whose diameter chord is a given line segment. [509]

Circle2D[point, r] constructs the circle centered at a point with a given radius. [409]

Circle2D[*point*, *point*] constructs the circle given a center point and a point on the circle. [409]

Circle2D[point, point, point] constructs a circle through three points. [410]

Circle2D[*point*, *line*] constructs a circle with a given center point and tangent to a line. [409]

Circle2D[quad] constructs the circle associated with a quadratic. [409]

Circle2D[*triangle*, Circumscribed2D] constructs a circle circumscribed about a triangle. [553]

Circle2D[*triangle*, Inscribed2D] constructs a circle inscribed inside a triangle. [553]

See also: Inscribed2D, Circumscribed2D, Pencil2D, TangentCircles2D.

### ■ Circumference2D

Circumference2D[circle] computes the circumference of a circle. [396]

Circumference[ellipse] computes the circumference of an ellipse. [397]

See also: ArcLength2D.

■ Circumscribed2D

Circumscribed2D is a keyword indicating a construction involving a triangle's circumscribed circle. [552]

See also: Circle2D, Point2D.

#### ■ Complement2D

Complement2D is a keyword indicating the construction of an arc's complement. [391]

See also: Arc2D.

### ConicArc2D

ConicArc2D[ $\{x_0, y_0\}$ ,  $\{x_A, y_A\}$ ,  $\{x_1, y_1\}$ ,  $\rho$ ] is the standard representation of a conic arc. The coordinates of the start point are  $\{x_0, y_0\}$ , the coordinates of the apex point are  $\{x_A, y_A\}$  and the coordinates of the end point are  $\{x_1, y_1\}$ . The projective discriminant is  $\rho$ . [415]

ConicArc2D[ $\{x_0, y_0\}$ ,  $\{x_A, y_A\}$ ,  $\{x_1, y_1\}$ ,  $\rho$ ][t] and cnarc[t] return the  $\{x, y\}$  coordinates of a point at parameter t on a conic arc. Parameter values in the range  $0 \le t \le 1$  produce coordinates covering the entire length of the conic arc. [416]

ConicArc2D[ $\{x_0, y_0\}$ ,  $\{x_A, y_A\}$ ,  $\{x_1, y_1\}$ ,  $\rho$ ][ $\{t_1, t_2\}$ ] produces graphics primitives representing the portion of the conic arc between parameters  $t_1$  and  $t_2$ when plotted. [416]

ConicArc2D[*line*, *conic*] constructs a conic arc defined by a conic (circle, ellipse, hyperbola or parabola) and a line containing the conic arc's chord. [419]

#### ■ Conjugate2D

Conjugate2D is a keyword indicating the construction of a conjugate hyperbola. [450]

See also: Hyperbola2D.

#### ■ Coordinates2D

Coordinates2D[args..] returns the  $\{x, y\}$  coordinates of the point that would be returned by the function Point2D[args..]. [490]

See also: Point2D, XCoordinate2D, YCoordinate2D.

#### ■ CurveLength2D

CurveLength2D is an option for the Sketch2D command specifying the approximate length that an unbounded curve should be rendered when plotted. [512]

The initial default, if not specified, is 10. The default can by changed using the *Mathematica* SetOptions command.

See also: AskCurveLength2D, Sketch2D.

#### ■ CurveLimits2D

CurveLimits2D[coords, curve] is a low-level function that computes a list of two parameter values on a curve such that the point whose coordinates are given is a distance CurveLength2D/2 from the points on the curve at the parameter values. [513]

See also: AskCurveLength2D, CurveLength2D, Sketch2D.

#### ■Directrices2D

Directrices2D[conic] returns a list of the directrix line(s) of a conic curve.

The conic may be an ellipse [413], hyperbola [413] or parabola [413]. If the conic is an ellipse or hyperbola there are two directrix lines in the list; if the conic is a parabola there is one directrix line in the list.

#### ■Distance2D

Distance2D[coords, coords] computes the distance between two positions given by coordinates. [491]

Distance2D[point, point] computes the distance between two points. [491]

Distance2D[point, line] computes the distance between a point and a line. [460]

Distance2D[*point*, *circle*] computes the distance between a point and a circle. [407]

#### ■ Eccentricity2D

Eccentricity2D[conic] computes the eccentricity of a conic.

The conic may be a ellipse [412], hyperbola [412] or parabola [412].

# ■Ellipse2D

Ellipse2D[ $\{h, k\}$ ,  $a, b, \theta$ ] is the standard representation of an ellipse. The coordinates of the center point are  $\{h, k\}$ , the length of the semi-major axis is a, the length of the semi-minor axis is b and the angle of rotation, counter-clockwise with respect to the +x-axis, is  $\theta$ . [421]

Ellipse2D[{h, k},  $a, b, \theta$ ][ $\theta_1$ ] and  $ellipse[\theta_1]$  return the {x, y} coordinates of a point at parameter  $\theta_1$  on an ellipse. Parameter values in the range  $0 \le \theta_1 < 2\pi$  produce coordinates covering the complete circumference of the ellipse. [422]

Ellipse2D[{h, k},  $a, b, \theta$ ][{ $\theta_1, \theta_2$ }] produces graphics primitives on the portion of the ellipse between parameter values  $\theta_1$  and  $\theta_2$  when plotted. [422]

Ellipse2D[*point*, *line*, *e*] constructs an ellipse defined by a focus point, directrix line and eccentricity. [426]

Ellipse2D[*point*, *point*, *e*] constructs an ellipse from two focus points and the eccentricity. [426]

Ellipse2D[{point, point}, e] constructs an ellipse from two vertex points and the eccentricity. [425]

# ■ Equation2D

Equation2D[line,  $\{x, y\}$ ] returns the equation Ax + By + C == 0, which is the equation of the line. [428]

Equation2D[quad,  $\{x, y\}$ ] returns  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F == 0$ , which is the equation of the quadratic. [428] See also: Polynomial2D.

### FocalChords2D

FocalChords2D[*conic*] returns a list containing the focal chords of a conic curve (line segments).

The conic may be an ellipse [414], hyperbola [414] or parabola [414]. If the conic is an ellipse or hyperbola the list contains two focal chords (line segments); if the conic is a parabola the list contains a single focal chord (line segment).

### FocalLength2D

FocalLength2D[parabola] returns the focal length of a parabola. [481]

See also: Parabola2D.

# ∎Foci2D

Foci2D[conic] returns a list containing the focus point(s) of a conic.

The conic may be an ellipse [412], hyperbola [412] or parabola [412]. If the conic is an ellipse or hyperbola the list contains two focus points; if the conic is a parabola the list contains a single focus point.

# ■Hyperbola2D

Hyperbola2D[ $\{h, k\}$ ,  $a, b, \theta$ ] is the standard representation of a hyperbola. The coordinates of the center point are  $\{h, k\}$ , the length of the semi-transverse axis is a, the length of the semi-conjugate axis is b and the angle of rotation, counter-clockwise with respect to the +x-axis, is  $\theta$ . [445]

Hyperbola2D[{h, k},  $a, b, \theta$ ][t] and hyperbola[t] return the {x, y} coordinates of a point at parameter t on the primary branch of a hyperbola. Parameter values in the range  $-\infty < t < +\infty$  cover the complete hyperbola branch. The primary branch opens about the +x-axis when the angle of rotation is zero. [446] Hyperbola2D[ $\{h, k\}$ ,  $a, b, \theta$ , True][t] returns the  $\{x, y\}$  coordinates of a point at parameter t on the non-primary (reflected) branch of a hyperbola (used only for graphics rendering). [446]

Hyperbola2D[{h, k},  $a, b, \theta$ ] [{ $t_1, t_2$ }] produces graphics primitives for a portion of the hyperbola between parameters values  $t_1$  and  $t_2$  when plotting. If  $t_1 < t_2$  the parameters represent a portion of the primary branch of the

hyperbola; if  $t_1 > t_2$  the parameters represent a portion of the other branch. [446]

Hyperbola2D[*hyperbola*, Conjugate2D] constructs the conjugate of a hyperbola. [450]

Hyperbola2D[*point*, *line*, *e*] constructs a hyperbola defined by a focus point, directrix line and eccentricity. [451]

Hyperbola2D[*point*, *point*, *e*] constructs a hyperbola from two focus points and the eccentricity. [450]

Hyperbola2D[{point, point}, e] constructs a hyperbola from two vertex points and the eccentricity. [450]

See also: Conjugate2D.

#### Inscribed2D

Inscribed2D is a keyword indicating a construction involving a triangle's inscribed circle. [552]

See also: Circle2D, Point2D.

# Is2D

Is2D[*object*, *objHeadList*] is a low-level function that returns True if the object is a valid Descarta2D object and its head is included in the head list; otherwise, returns False. [472]

# IsApproximate2D

IsApproximate2D[*expr*] is a low-level function that returns True if the expression contains approximate real numbers; otherwise, returns False. [431]

The function will attempt to detect if the pending evaluation will eventually be approximated using the N[expr] function. If this condition is detected the function will also return True.

#### ■ IsCoincident2D

IsCoincident2D[*obj*, *obj*] returns True if two objects are of the same type and are coincident; otherwise, returns False. The objects may be circles, coordinates, lines, points or quadratics. [439]

The function returns unevaluated if the two objects are of a different type.

**IsCoincident2D**[*objList*] returns **True** if any pair of objects in a list are of the same type and are coincident; otherwise, returns **False**. [440]

### IsCollinear2D

IsCollinear2D[*point*, *point*] returns True if three points are collinear; otherwise, returns False. [440]

IsCollinear2D[*ptList*] returns True if *any* triple of points in a list is collinear; otherwise, returns False. [440]

#### ■ IsComplex2D

IsComplex2D[*expr*, tol] is a low-level function that returns True if the expression, when evaluated, contains a complex number (a number is considered complex if the absolute value of its imaginary part is greater than the tolerance); otherwise, returns False. [431]

The tolerance, if omitted, defaults to  $10^{-10}$ .

IsComplex2D[*exprList*, *tol*] returns True if *any* expression in a list, when evaluated, contains a complex number; otherwise, returns False. [432]

IsComplex2D[*exprList*, Or, *tol*] returns True if *any* expression in a list, when evaluated, contains a complex number; otherwise, returns False. [432]

IsComplex2D[*exprList*, And, *tol*] returns True if *all* the expressions in a list, when evaluated, contain complex numbers; otherwise, returns False. [432]

#### ■ IsConcentric2D

IsConcentric2D[circle, circle] returns True if two circles are concentric; otherwise, returns
False. [440]

IsConcentric2D[*cirList*] returns True if *any* pair of circles in a list are concentric; otherwise, returns False. [440]

### ■ IsConcurrent2D

IsConcurrent2D[*line*, *line*] returns True if three lines are concurrent (intersect in a common point); otherwise, returns False. [441]

IsConcurrent2D[*lnList*] returns True if *any* triple of lines in a list is concurrent; otherwise, returns False. [441]

# IsDisplay2D

IsDisplay2D[*object*] is a low-level function that returns True if the object is a displayable *Descarta2D* object; otherwise, returns False. [512]

### ■ IsNegative2D

IsNegative2D[*expr*, *tol*] is a low-level function that returns True if the expression, when evaluated, is negative (a number is considered negative if it is less than zero and its absolute value is greater than the tolerance); otherwise, returns False. [434]

The tolerance, if omitted, defaults to  $10^{-10}$ .

IsNegative2D[*exprList*, *tol*] returns True if *any* expression in a list, when evaluated, is negative; otherwise, returns False. [434]

IsNegative2D[*exprList*, Or, *tol*] returns True if *any* expression in a list, when evaluated, is negative; otherwise, returns False. [434]

IsNegative2D[*exprList*, And, *tol*] returns True if *all* the expressions in a list, when evaluated, are negative; otherwise, returns False. [434]

See also: IsZero2D, IsZero0rNegative2D.

IsNumeric2D

IsNumeric2D[*expr*, tol] is a low-level function that returns True if all the atoms in an expression can be evaluated to real numbers (a complex number is considered real if the absolute value of its imaginary part is less than the tolerance); otherwise, returns False. [432]

The tolerance, if omitted, defaults to  $10^{-10}$ .

IsNumeric2D[*expr*, *funcName*, *tol*] returns True if all the atoms in an expression can be evaluated to real numbers; otherwise, returns False and displays a message stating that the function, *funcName*, requires numerical arguments. This form is a low-level function and is intended to be used for argument checking. [432]

### ∎IsOn2D

IsOn2D[point, curve] returns True if a point is on a curve; otherwise, returns False.

The curve may be a line [441], circle [441] or quadratic [441].

IsOn2D[point, Quadratic2D[conic]] returns True if a point is on a conic; otherwise, returns False. The conic may be a circle, ellipse, hyperbola or parabola.

### ■ IsParallel2D

IsParallel2D[*line*, *line*] returns True if two lines are parallel; otherwise, returns False. [442]

IsParallel2D[*lnList*] returns True if *any* pair of lines in a list is parallel; otherwise, returns False. [442]

See also: IsTripleParallel2D.

#### IsPerpendicular2D

IsPerpendicular2D[*line*, *line*] returns True if two lines are perpendicular; otherwise, returns False. [442]

IsPerpendicular2D[*lnList*] returns True if *any* pair of lines in a list is perpendicular; otherwise, returns False. [443]

# ■ IsReal2D

IsReal2D[*expr*, *tol*] is a low-level function that returns True if the expression, when evaluated, is a real number (a complex number is considered real if the absolute value of its imaginary part is less than the tolerance); otherwise, returns False. [433]

The tolerance, if omitted, defaults to  $10^{-10}$ .

# ■ IsScalar2D

IsScalar2D[*expr*] is a low-level function that returns True if the expression appears to be a scalar quantity—that is, it cannot be recognized as a list, a complex number or a *Descarta2D* object; otherwise, returns False. [433]

This function is used by Descarta2D for argument checking.

See also: IsScalarPair2D.

#### ■ IsScalarPair2D

IsScalarPair2D[{*expr*, *expr*}] is a low-level function that returns True if both expressions appear to be scalar quantities—that is, they cannot be recognized as lists, complex numbers or *Descarta2D* objects; otherwise, returns False. [434]

This function is used by *Descarta2D* for argument checking.

See also: IsScalar2D.

### ■ IsTangent2D

IsTangent2D[*line*, *circle*] returns True if a line is tangent to a circle; otherwise, returns False. [443]

IsTangent2D[*line*, *quad*] returns True if a line is tangent to a quadratic; otherwise, returns False. [443]

IsTangent2D[*line*, Quadratic2D[*conic*]] returns True if a line is tangent to a conic; otherwise, returns False. The conic may be a circle, ellipse, hyperbola or parabola. [443]

IsTangent2D[*circle*, *circle*] returns True if two circles are tangent to each other; otherwise, returns False. [443]

# ■ IsTinyImaginary2D

IsTinyImaginary2D[*expr*, *tol*] is a low-level function that returns True if any complex number in an expression has a tiny imaginary part (the imaginary part is considered tiny if its absolute value is less than the tolerance); otherwise, returns False. [434]

The tolerance, if omitted, defaults to  $10^{-10}$ .

# ■ IsTripleParallel2D

IsTripleParallel2D[*line*, *line*, *line*] returns True if three lines are mutually parallel; otherwise, returns False. [442]

IsTripleParallel2D[*lnList*] returns True if *any* triple of lines in a list is mutually parallel; otherwise, returns False. [442]

See also: IsParallel2D.

### ■IsValid2D

IsValid2D[*object*] is a low-level function that returns True if the object is syntactically valid; otherwise, returns False. [472]

The object may be an arc [389], circle [406], conic arc [417], ellipse [423], hyperbola [447], line [459], line segment [506], parabola [481], point [490], quadratic [498] or triangle [546].

#### ■IsZero2D

IsZero2D[*expr*, *tol*] is a low-level function that returns True if the expression, when evaluated, is zero (a number is considered zero if its absolute value is less than the tolerance); otherwise, returns False. [435]

The tolerance, if omitted, defaults to  $10^{-10}$ .

IsZero2D[*exprList*, *tol*] returns True if *any* expression in a list, when evaluated, is zero; otherwise, returns False. [435]

IsZero2D[*exprList*, Or, *tol*] returns True if *any* expression in a list, when evaluated, is zero; otherwise, returns False. [435]

IsZero2D[*exprList*, And, *tol*] returns True if *all* the expressions in a list, when evaluated, are zero; otherwise, returns False. [435]

See also: IsNegative2D, IsZeroOrNegative2D.

# IsZeroOrNegative2D

IsZeroOrNegative2D[*expr*, *tol*] returns True if the expression, when evaluated, is zero or negative; otherwise, returns False. [435]

The tolerance, if omitted, defaults to  $10^{-10}$ .

IsZeroOrNegative2D[*exprList*, *tol*] returns True if *any* expression in a list, when evaluated, is zero or negative; otherwise, returns False. [436]

IsZeroOrNegative2D[*exprList*, Or, *tol*] returns True if *any* expression in a list, when evaluated, is zero or negative; otherwise, returns False. [436]

IsZeroOrNegative2D[*exprList*, And, *tol*] returns True if *all* the expressions in a list, when evaluated, are zero or negative; otherwise, returns False. [436]

See also: IsNegative2D, IsZero2D.

### ■ Length2D

Length2D[lnseg] computes the length of a line segment. [507]

See also: ArcLength2D.

#### ■Line2D

Line2D[A, B, C] is the standard representation of the line Ax + By + C = 0. [458]

Line2D[A, B, C] [t] and line[t] return the  $\{x, y\}$  coordinates of a point at parameter t on a line. Parameter values in the range  $-\infty < t < +\infty$  produce coordinates covering the complete line. [458]

Line2D[A, B, C] [ $\{t_1, t_2\}$ ] produces graphics primitives for the line segment between parameters  $t_1$  and  $t_2$  when plotting. [458]

Line2D[circle, circle] constructs the radical axis line of two circles. [408]

Line2D[coords, coords] constructs a line through two positions specified by  $\{x, y\}$  coordinates. [462]

Line2D[*ellipse*] constructs a line which contains the major axis of an ellipse. [425]

Line2D[eqn,  $\{x, y\}$ ] constructs a line from the equation Ax + By + C == 0. [458]

Line2D[hyperbola] constructs a line which contains the transverse axis of a hyperbola. [449]

Line2D[line] constructs a line with normalized coefficients. [461]

Line2D [*line*, d] constructs a line offset a distance d from a given line. The distance may be positive or negative producing one of two possible offsets. [462]

Line2D[*line*, *line*, *k*, Pencil2D] constructs a family of lines (pencil), parameterized by k, passing through the intersection point of two given lines. [485]

Line2D[lnseg] constructs a line containing a line segment. [508]

Line2D[*lnseg*, Perpendicular2D] constructs a line that is the perpendicular bisector of a line segment. [508]

Line2D[parabola] constructs a line which contains the axis of a parabola. [483]

Line2D[*point*, *curve*] constructs the polar (line) of a curve given the pole (point). If the pole (point) is on the curve, then the polar (line) is the tangent to the curve at the pole (point). The curve may be a circle [408], ellipse [425], hyperbola [450], parabola [483] or quadratic [463].

Line2D[*point*, k, Pencil2D] constructs a family of lines (pencil), parameterized by k, passing through a point. [485]

Line2D[*point*, *line*] constructs a line through a point perpendicular to a line. [463]

Line2D[*point*, *line*, Perpendicular2D] also constructs a line through a point perpendicular to a line. [463]

Line2D[*point*, *line*, Parallel2D] constructs a line through a point parallel to a line. [463]

Line2D[point, m] constructs a line with slope m passing through a point. [462]

Line2D[point, Infinity] constructs a vertical line through a point. [462]

Line2D[point, point] constructs a line through two points. [462]

Line2D[*point*, *point*, Perpendicular2D] constructs a line equidistant from two points. This line is the perpendicular bisector of the line segment defined by the two points. [463]

Line2D[poly,  $\{x, y\}$ ] constructs a line from the polynomial Ax + By + C. [458]

Line2D[*triangle*,  $n_1$ ,  $n_2$ ] constructs a line containing vertices  $n_1$  and  $n_2$  of a triangle. [552]

See also: Parallel2D, Pencil2D, Perpendicular2D.

# ∎Loci2D

Loci2D[quad] returns a list of objects represented by a quadratic. The list may contain a conic, one or two lines, a point or it may be empty. [465]

Loci2D[cnarc] returns a list containing the curve underlying a conic arc. [419]

Loci2D[point, length, e] returns a list containing the conic defined by the vertex equation parameters. The point is the vertex point, the length is the focal length and the constant, e, is the eccentricity. The conic is constructed in standard position. [468]

Loci2D [point, length,  $e, \theta$ ] returns a list containing the conic defined by the vertex equation parameters. The point is the vertex point, the length is the focal length, the constant, e, is the eccentricity and  $\theta$  is the angle of rotation. [468] Loci2D [point, line, e] returns a list containing the conic defined by a focus point, directrix line and eccentricity. [468]

#### MakePrimitives2D

MakePrimitives2D[curve,  $\{t_1, t_2\}$ ] is a low-level function that returns a list of Mathematica graphics primitives approximating a curve between two parameter values. [513]

The curve may be an arc, circle, conic arc, ellipse, hyperbola, line, line segment or parabola.

# ■ MaxSeconds2D

MaxSeconds2D is a keyword indicating the maximum number of seconds allowed for solving equations. [516]

See also: Solve2D.

# MedialEquations2D

MedialEquations2D[{*obj*, *obj*}] returns a list of lines or quadratics equidistant from two given objects. The given objects may be points, lines or circles. [473]

See also: MedialLoci2D.

### MedialLoci2D

MedialLoci2D[{*obj*, *obj*}] returns a list of objects equidistant from two given objects. The given objects may be points, lines or circles. [474]

See also: MedialEquations2D.

#### ∎ObjectNames2D

ObjectNames2D[] returns a list of strings which are the names of all the Descarta2D objects. [472]

#### ∎Parabola2D

Parabola2D[ $\{h, k\}, f, \theta$ ] is the standard representation of a parabola. The coordinates of the vertex point are  $\{h, k\}$ , the focal length is f and the angle of rotation, counter-clockwise with respect to the +x-axis, is  $\theta$ . [479]

**Parabola2D**[{h, k},  $f, \theta$ ][t] and *parabola*[t] return the {x, y} coordinates of a point at parameter t on a parabola. Parameter values in the range  $-\infty < t < +\infty$  produce coordinates covering the complete parabola. [480]

**Parabola2D**[ $\{h, k\}, f, \theta$ ][ $\{t_1, t_2\}$ ] produces graphics primitives for the portion of the parabola between parameters  $t_1$  and  $t_2$  when plotting. [480]

Parabola2D[*point*, *line*] constructs a parabola defined by a focus point and a directrix line. [483]

# ■Parallel2D

Parallel2D is a keyword indicating a parallel construction. [463]

See also: Line2D, TangentLines2D.

#### Parameters2D

Parameters2D[*line*, *curve*] computes a list of the two parameters where a line intersects a curve.

The curve may be a circle [455], ellipse [455], hyperbola [455] or parabola [456]. The list of parameters is useful for computing areas and arc lengths defined by the line and the curve.

See also: ArcLength2D, SectorArea2D, SegmentArea2D.

#### Pencil2D

Pencil2D is a keyword indicating the construction of a pencil of curves. [485]

See also: Line2D, Circle2D, Quadratic2D.

#### Perimeter2D

Perimeter2D[triangle] computes the perimeter of a triangle. [398]

See also: ArcLength2D.

# Perpendicular2D

Perpendicular2D is a keyword indicating a perpendicular construction. [463]

See also: Line2D, TangentLines2D.

#### ∎Point2D

Point2D[{x, y}] or Point2D[*coords*] is the standard representation of a point with coordinates {x, y}. [489]

Point2D[x, y] constructs a point at coordinates (x, y). [492] Point2D[arc] constructs the center point of an arc. [391] Point2D[circle] constructs the center point of a circle. [408] Point2D[cnarc] constructs the center point of the conic underlying a conic arc. [419]

Point2D[cnarc, Apex2D] constructs the apex control point of a conic arc. [419]

**Point2D**[*conic*] constructs the center point of a central conic. The conic may be a circle [408], ellipse [424] or hyperbola [449].

Point2D[curve[t]] constructs a point at a parameter value on a curve.

**Point2D**[*line*, *curve*] constructs the pole (point) of a curve given the polar (line). If the polar (line) is tangent to the curve, then the pole (point) is the point of tangency. The curve may be a circle [408], ellipse [425], hyperbola [449], parabola [482] or quadratic [494].

Point2D[line, line] constructs the intersection point of two lines. [494]

Point2D[lnseg] constructs the midpoint of a line segment. [508]

Point2D[parabola] constructs the vertex point of a parabola. [482]

Point2D[point, line] constructs a point by projecting a point onto a line. [493]

Point2D[point, line, d] constructs a point by offsetting a point a distance, d, in the direction of a line. The distance may be positive or negative resulting in one of two possible offset points. [493]

Point2D[point, line,  $\{u, v\}$ ] constructs the point with coordinates  $\{u, v\}$  in the coordinate system defined by a point and a line. The line defines the y-axis and the point is on the +x-axis. [494]

Point2D[point, point] constructs the midpoint of two points. [493]

**Point2D**[*point, point, d*] constructs a point by offsetting a point a distance, *d*, in the direction of a second point. If the distance is negative, the point is offset in the opposite direction. [493]

Point2D[point, point,  $r_1$ ,  $r_2$ ] constructs a point dividing the segment between two points into the ratio  $r_1/r_2$ . [493]

**Point2D**[quad] constructs the center point of a quadratic, assuming the quadratic is a central conic. [494]

Point2D[*triangle*, Centroid2D] constructs a point at the centroid of a triangle. [551]

Point2D[*triangle*, Circumscribed2D] constructs the center point of a circle circumscribed about a triangle. [552]

Point2D[*triangle*, Inscribed2D] constructs the center point of a circle inscribed inside a triangle. [552]

Point2D[triangle, n] constructs a point at vertex n of a triangle. [552]

See also: Apex2D, Centroid2D, Circumscribed2D, Inscribed2D.

#### ■Points2D

Points2D[curve, curve] constructs a list containing the intersection points of two
curves. [453]

The curves may be lines, circles, ellipses, hyperbolas, parabolas or quadratics.

#### Polynomial2D

**Polynomial2D**[*line*,  $\{x, y\}$ ] returns the polynomial Ax + By + C, which is the polynomial of the line. [428]

Polynomial2D[quad,  $\{x, y\}$ ] returns  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F$ , which is the polynomial of the quadratic. [428]

See also: Equation2D.

PrimaryAngle2D

PrimaryAngle2D[ $\theta$ ] returns a primary angle in the range  $0 \le \phi < 2\pi$  where  $\phi = \text{Mod}[\theta, 2\pi]$ . [478]

**PrimaryAngle2D**[ $\theta$ ,  $2\pi$ ] returns a primary angle in the range  $0 \le \phi < 2\pi$  where  $\phi$  is given by Mod[ $\theta$ ,  $2\pi$ ]. [478]

**PrimaryAngle2D**[ $\theta$ ,  $\pi$ ] returns a primary angle in the range  $0 \le \phi < \pi$  where  $\phi$  is given by Mod[ $\theta$ ,  $\pi$ ]. [478]

See also: PrimaryAngleRange2D.

#### PrimaryAngleRange2D

PrimaryAngleRange2D[{ $\theta_1$ ,  $\theta_2$ }] returns a list of two primary angles, { $\phi_1$ ,  $\phi_2$ }, such that  $0 \le \phi_1 < 2\pi$  and  $\phi_1 < \phi_2 < (\phi_1 + 2\pi)$ . [478]

**PrimaryAngleRange2D**[*arc*] returns a list of two primary angles,  $\{\phi_1, \phi_2\}$ , which are the spanning angles of the arc. [390]

See also: PrimaryAngle2D.

### Quadratic2D

Quadratic2D[A, B, C, D, E, F] is the standard representation of the quadratic given by  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0.$  [497]

Quadratic2D[cnarc] constructs the quadratic underlying a conic arc. [416]

Quadratic2D[*conic*] constructs the quadratic associated with a conic. The conic may be a circle [405], ellipse [422], hyperbola [446] or parabola [480].

Quadratic2D[*coords*] constructs the quadratic representing a position specified by coordinates (a circle of zero radius). [500]

Quadratic2D[eqn,  $\{x, y\}$ ] constructs a quadratic from an equation given in the form  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F == 0.$  [500]

Quadratic2D[*line*, *line*] constructs the quadratic representing the product of two lines. [502]

Quadratic2D[{line, line}, {line, line}, k, Pencil2D] constructs a family of quadratics, parameterized by k, passing through the intersection points of four lines taken in pairs. [486]

Quadratic2D[line, line, line, line] constructs the quadratic tangent to five lines. [501]

Quadratic2D[*point*] constructs the quadratic representing a point (a circle of zero radius). [491]

Quadratic2D[*point*, *length*, *e*] constructs the quadratic defined by the vertex equation parameters. The point is the vertex point, the length is the focal length and the constant, *e*, is the eccentricity. The quadratic is constructed in standard position. [502]

Quadratic2D[point, length,  $e, \theta$ ] constructs the quadratic defined by the vertex equation parameters. The point is the vertex point, the length is the focal length, the constant, e, is the eccentricity and  $\theta$  is the angle of rotation. [502]

Quadratic2D[*point*, *line*, *e*] constructs the quadratic defined by a focus point, directrix line and eccentricity. [502]

Quadratic2D[*point*, *point*] constructs the quadratic representing the circle passing through three points. [501]

Quadratic2D[*point*, *point*, *point*, *point*, *k*, Pencil2D] constructs a family of quadratics, parameterized by *k*, passing through four points. [487]

Quadratic2D[point, point, point, point] constructs a quadratic passing through five points. [501]

Quadratic2D[poly,  $\{x, y\}$ ] constructs a quadratic from the polynomial given in the form  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F$ . [500]

Quadratic2D[quad] constructs a quadratic with normalized coefficients. [500]

Quadratic2D[quad, quad, k, Pencil2D] constructs a family (pencil) of quadratics, parameterized by k, and passing through the intersection points of two quadratics. [486]

See also: Pencil2D.

# Radius2D

Radius2D[circle] returns the radius of a circle. [407]

Radius2D[*arc*] returns the radius of an arc. [390] See also: Arc2D, Circle2D.

#### ■Reflect2D

Reflect2D[object, line] reflects an object in a line.

The *object* may be an arc [390], circle [407], conic arc [418], coordinates [540], ellipse [424], hyperbola [448], line [461], line segment [507], parabola [481], point [492] or triangle [550].

**Reflect2D**[*objList*, *line*] reflects a list of objects in a line, returning a list of objects. [540]

Reflect2D[eqn,  $\{x, y\}$ , line] reflects an equation in a line. [540]

The equation may be linear, Ax + By + C == 0, or quadratic,  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F == 0.$  [540]

See also: ReflectAngle2D, Rotate2D, Scale2D, Translate2D.

# ■ ReflectAngle2D

ReflectAngle2D[ $\theta$ , *line*] computes the reflection of an angle in a line. [540]

If a line L makes an angle  $\theta$  with the +x-axis and line L' is the reflection of L in the given line (the second argument to the function), then the function computes the angle  $\theta'$  that L' makes with the +x-axis.

See also: Reflect2D.

# Rho2D

Rho2D[cnarc] returns the projective discriminant of a conic arc. [417]

See also: ConicArc2D.

#### Rotate2D

Rotate2D[*object*,  $\theta$ , *coords*] rotates an object by an angle  $\theta$  (in radians) about a position whose coordinates are given. If the coordinates are omitted, the default is the origin.

The *object* may be an arc [389], circle [407], conic arc [418], coordinates [541], ellipse [424], hyperbola [448], line [461], line segment [507], parabola [482] or triangle [551].

Rotate2D[*objList*,  $\theta$ , *coords*] rotates a list of objects. [541]

Rotate2D[eqn,  $\{x, y\}$ ,  $\theta$ , coords] rotates an equation by an angle  $\theta$  (in radians) about a position whose coordinates are given. [541]

The equation may be linear, Ax + By + C == 0, or quadratic,  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F == 0.$  [541]

See also: Reflect2D, Scale2D, Translate2D.

### ■Scale2D

Scale2D[*object*, *s*, *coords*] scales an object from a position given as coordinates. If the coordinates are omitted, the default is the origin. [541]

The *object* may be an arc [391], circle [407], conic arc [418], coordinates [542], ellipse [424], hyperbola [449], line [461], line segment [507], parabola [482] or triangle [551].

Scale2D[*objList*, *s*, *coords*] scales a list of objects from a position whose coordinates are given. [542]

Scale2D[eqn,  $\{x, y\}$ , s, coords] scales an equation from a position. [542]

The equation may be linear, Ax + By + C == 0, or quadratic,

 $Ax^{2} + Bxy + Cy^{2} + Dx + Ey + F == 0.$  [542]

See also: Reflect2D, Rotate2D, Translate2D.

# SectorArea2D

SectorArea2D[curve,  $\{t_1, t_2\}$ ] computes the area of a sector of a curve between two parameters.

The curve may be a circle [400], ellipse [401] or hyperbola [402] (the sector is defined from the center point of the curve to the two points defined by the parameters on the curve).

See also: Area2D, SegmentArea2D.

#### Segment2D

Segment2D[ $\{x_0, y_0\}$ ,  $\{x_1, y_1\}$ ] is the standard representation of a line segment. The coordinates of the start point are  $\{x_0, y_0\}$  and the coordinates of the end point are  $\{x_1, y_1\}$ . [505]

Segment2D[ $\{x_0, y_0\}$ ,  $\{x_1, y_1\}$ ][t] and lnseg[t] return the  $\{x, y\}$  coordinates of a point at parameter t on a line segment. Parameter values in the range

 $0 \le t \le 1$  produce coordinates covering the entire length of the line segment. [505]

Segment2D[A, B, C] [ $\{t_1, t_2\}$ ] produces graphics primitives for the line segment between parameters  $t_1$  and  $t_2$  when plotting. [506]

Segment2D[point, point] constructs a line segment between two points. [508]

Segment2D[*triangle*,  $n_1$ ,  $n_2$ ] constructs a line segment between vertices  $n_1$  and  $n_2$  of a triangle. [552]

#### SegmentArea2D

SegmentArea2D[curve,  $\{t_1, t_2\}$ ] computes the area of a segment of a curve between two parameters.

The curve may be a circle [400], ellipse [401], hyperbola [402] or parabola [402] (the segment is the area between the curve and the chord defined by the two parameters).

See also: Area2D, SectorArea2D.

■ SemiConjugateAxis2D

SemiConjugateAxis2D[hyperbola] returns the length of the semi-conjugate axis of a hyperbola. [448]

See also: Hyperbola2D, SemiTransverseAxis2D.

SemiMajorAxis2D

SemiMajorAxis2D[ellipse] returns the length of the semi-major axis of an ellipse. [423]

See also: Ellipse2D, SemiMinorAxis2D.

SemiMinorAxis2D

SemiMinorAxis2D[ellipse] returns the length of the semi-minor axis of an ellipse. [423]

See also: Ellipse2D, SemiMajorAxis2D.

SemiTransverseAxis2D

SemiTransverseAxis2D[hyperbola] returns the length of the semi-transverse axis of a hyperbola. [448]

See also: Hyperbola2D, SemiConjugateAxis2D.

■ SetDisplay2D

SetDisplay2D[*objPatt*, *objPrim*] is a low-level function that specifies the graphics primitives to use when plotting a given object pattern. [513]

#### ■ SimplifyCoefficients2D

SimplifyCoefficients2D[*coefList*] is a low-level function that returns a list of coefficients with common factors removed. [427]

Simplify[*line*] and FullSimplify[*line*] use SimplifyCoefficients2D to simplify the coefficients of a line.

Simplify[quad] and FullSimplify[quad] use SimplifyCoefficients2D to simplify the coefficients of a quadratic.

### ■Sketch2D

Sketch2D[objList, opts] produces a plot of the objects in a list. [513]

The list of objects may be nested. Any of the options for the *Mathematica* **Graphics** command may be specified.

Sketch2D[*objList*, CurveLength2D->*n*, *opts*] produces a plot of the objects in a list, using a specified curve length for unbounded curves. [513]

See also: AskCurveLength2D, CurveLength2D.

# ■Slope2D

Slope2D[line] computes the slope of a line. [460]

Slope2D[lnseg] computes the slope of a line segment. [507]

#### ■Solve2D

Solve2D[eqnList, varList] is a low-level function that solves a list of equations for a list of variables and returns a list of rules representing the solutions. [516]

Solve2D[eqnList, varList, MaxSeconds2D->n] solves a list of equations for a list of variables with a time limit of n seconds. [516]

See also: MaxSeconds2D.

#### ■SolveTriangle2D

SolveTriangle2D[{ $\{s_1, s_2, s_3\}$ , { $a_1, a_2, a_3$ }}] computes a triangle configuration from three sides and/or angles. Unspecified arguments should be Null. [548]

SolveTriangle2D[{ $\{s_1, s_2, s_3\}$ , { $a_1, a_2, a_3$ }}, True] computes a triangle configuration from three sides and/or angles, returning an alternate solution, if one exists. [548]

The configuration is returned in the form  $\{\{s_1, s_2, s_3\}, \{a_1, a_2, a_3\}\}$ .

See also: Triangle2D.

#### ■ Span2D

Span2D[arc] computes the arc length of the complete span of an arc. [395]

N[Span2D[cnarc]] numerically computes the arc length of the complete span of a conic arc. [396]

See also: ArcLength2D.

### ■ TangentCircles2D

TangentCircles2D[ $\{pt | ln | cir, pt | ln | cir, pt | ln | cir\}$ ] constructs a list of circles tangent to three objects (points, lines or circles). [522]

For brevity of expression a circle is said to be *tangent* to a point if the point is on the circle.

**TangentCircles2D**[ $\{pt | ln | cir\}$ , pt] constructs a list of circles tangent to an object (point, line or circle) with a given center point. [521]

TangentCircles2D[ $\{pt | ln | cir\}$ , ln | cir, r] constructs a list of circles tangent to an object (point, line or circle), whose center is on a line or circle, with a given radius. [521]

TangentCircles2D[ $\{pt | ln | cir, pt | ln | cir\}$ , r] constructs a list of circles tangent to two objects (points, lines or circles), with a given radius. [522]

TangentCircles2D[{pt| ln| cir, pt| ln| cir}, ln| cir] constructs a list of circles tangent to two objects (points, lines or circles), with center on a given line or circle. [521]

See also: Circle2D.

#### ■ TangentConics2D

TangentConics2D[ $\{pt | ln, pt | ln, pt | ln, pt | ln, pt | ln\}$ ] constructs a list of conics tangent to five objects (points or lines). [526]

The expressions in the resulting conics can be very complicated and are usually practical only if evaluated numerically.

See also: TangentQuadratics2D.

#### ■ TangentEquation2D

TangentEquation2D[*line*, *quad*] returns an equation involving the coefficients of a line and a quadratic that constrains the two curves to be tangent. [532]

#### TangentLines2D

TangentLines2D[curve, curve] constructs a list of lines tangent to two curves. [533]

The curves may be circles, ellipses, hyperbolas, parabolas or quadratics.

TangentLines2D[*line*, *curve*] constructs a list of lines parallel to a line and tangent to a curve. [532]

TangentLines2D[*line*, *curve*, Parallel2D] also constructs a list of lines parallel to a line and tangent to a curve. [532]

TangentLines2D[*line*, *curve*, Perpendicular2D] also constructs a list of lines perpendicular to a line and tangent to a curve. [532]

TangentLines2D[*point*, *curve*] constructs a list of lines from a point and tangent to a curve. [532]

See also: Parallel2D, Perpendicular2D, TangentSegments2D.

#### ■ TangentPoints2D

TangentPoints2D[*point*, *curve*] constructs a list of points that are the points of tangency of lines from a point to a curve. [537]

The curve may be a circle, ellipse, hyperbola, parabola or quadratic.

#### ■ TangentQuadratics2D

TangentQuadratics2D[ $\{pt | ln, pt | ln\}$ ] constructs a list of quadratics tangent to five objects (points or lines). [526]

The expressions in the resulting quadratics can be very complicated and are usually practical only if evaluated numerically.

See also: TangentConics2D.

#### ■ TangentSegments2D

TangentSegments2D[curve, curve] constructs a list of line segments tangent to two curves. [534]

The curves may be circles, ellipses, hyperbolas, parabolas or quadratics.

See also: TangentLines2D.

# ■Translate2D

Translate2D[object, {u, v}] translates an object delta distance.

The *object* may be an arc [391], circle [408], conic arc [418], coordinates [542], ellipse [424], hyperbola [449], line [461], line segment [508], parabola [482], quadratic [499] or triangle [551].

Translate2D[objList, {u, v}] translates a list of objects. [543]

**Translate2D**[eqn,  $\{x, y\}$ ,  $\{u, v\}$ ] translates an equation delta distance. [543]

The equation may be linear, Ax + By + C == 0, or quadratic,

 $Ax^{2} + Bxy + Cy^{2} + Dx + Ey + F == 0.$  [543]

See also: Reflect2D, Rotate2D, Scale2D.

# ■Triangle2D

Triangle2D[ $\{x_1, y_1\}$ ,  $\{x_2, y_2\}$ ,  $\{x_3, y_3\}$ ] is the standard representation of a triangle defined by three vertex coordinates. [546]

**Triangle2D**[ $\{s_1, s_2, s_3\}$ ] constructs a triangle from three side lengths. The first vertex of the triangle will be the origin and the second vertex will be on the +x-axis. [554]

Triangle2D[{ $\{s_1, s_2, s_3\}$ , { $a_1, a_2, a_3$ }}] constructs a triangle from three sides and/or angles. Unspecified arguments should be Null. The first vertex of the triangle will be the origin and the second vertex will be on the +x-axis. [554]

Triangle2D[{ $\{s_1, s_2, s_3\}$ , { $a_1, a_2, a_3$ }}, True] constructs a triangle from three sides and/or angles, returning an alternate solution, if one exists. [554]

Triangle2D[*line*, *line*] constructs a triangle whose sides are specified by three lines. [553]

Triangle2D[*point*, *point*] constructs a triangle whose vertices are specified by three points. [553]

See also: SolveTriangle2D.

### Vertices2D

Vertices2D[conic] returns a list containing the vertex point(s) of a conic curve.

The conic may be an ellipse [412], hyperbola [413] or parabola [413]. If the conic is an ellipse or hyperbola the list contains two vertex points; if the conic is a parabola the list contains a single vertex point.

#### XCoordinate2D

XCoordinate2D[point] returns the x-coordinate of a point. [491]

XCoordinate2D[coords] returns the x-coordinate of a location. [491]

See also: Coordinates2D, Point2D, YCoordinate2D.

# ■ YCoordinate2D

YCoordinate2D[point] returns the y-coordinate of a point. [491]

YCoordinate2D[coords] returns the y-coordinate of a location. [491]

See also: Coordinates2D, Point2D, XCoordinate2D.

# Chapter 24

# **Error Messages**

This chapter is a listing of all the error messages that can be generated by *Descarta2D* during computations. *Mathematica* may generate additional error messages. The messages are listed alphabetically by message name. The number in square brackets indicates the page where the error is defined in the packages.

# Arc2D

An invalid arc of the form  $arc_1$  has been detected; the arguments cannot be imaginary.

The arguments of an arc cannot involve imaginary numbers. *Descarta2D* will return the **\$Failed** symbol whenever the arguments of an object are determined to be invalid.

An invalid arc of the form *arc* has been detected; the bulge factor must be positive and the defining points must be distinct.

The bounding points of an arc cannot be coincident and the bulge factor must be positive. *Descarta2D* will return the **\$Failed** symbol whenever the arguments of an object are determined to be invalid.

The defining points are coincident; an arc cannot be constructed.

The defining points of an arc must be distinct. *Descarta2D* will return the **\$Failed** symbol if coincident points are detected.

The three defining points are collinear; an arc cannot be constructed.

An arc cannot be constructed through three collinear points. *Descarta2D* will return the **\$Failed** symbol if collinear points are detected.

The entry angle of the arc is invalid; the entry angle cannot be an integer multiple of Pi radians.

The entry angle of an arc cannot be an integer multiple of  $\pi$  radians. *Descarta2D* will return the **\$Failed** symbol if invalid entry angle is detected.

The radius of an arc must be positive. *Descarta2D* will return the **\$Failed** symbol if a non-positive radius is detected.

The angular span of the arc is invalid; the span cannot be an integer multiple of 2Pi radians.

The angular span of an arc cannot be a multiple of  $2\pi$  radians. The **\$Failed** symbol will be returned when an invalid span is specified.

# Circle2D

The points  $\{pt_1, pt_2\}$  are coincident; no valid circle exists.

When specifying a circle by two points, the points cannot be coincident. *Descarta2D* will return the **\$Failed** symbol if two coincident points are specified.

When specifying a circle through three points, the points cannot be collinear. *Descarta2D* will return the **\$Failed** symbol if it detects that the three specified points lie on a line.

An invalid circle of the form  $cir_1$  has been detected; the arguments cannot be imaginary.

The arguments defining a circle cannot be imaginary numbers. *Descarta2D* will return the **\$Failed** symbol if the arguments of an object involve imaginary numbers.

An invalid circle of the form  $\operatorname{cir}_1$  has been detected; the radius must be positive.

When defining a circle the radius must be a positive number. *Descarta2D* will return the **\$Failed** symbol whenever the arguments to an object are determined to be invalid.

The curve represented by  $quad_1$  is not a circle.

Descarta2D has detected that the curve represented by a Quadratic2D is not a circle. The **\$Failed** symbol will be returned.

When specifying a circle tangent to a line with a given center point, the point cannot be on the line. If *Descarta2D* detects that the point lies on the line, it will return the **\$Failed** symbol.

The radius argument, r, is invalid; the radius must be positive.

When specifying a circle by center point and radius, the radius must be positive. *Descarta2D* will return the **\$Failed** symbol if a non-positive radius is specified during a circle construction.

# ConicArc2D

The chord defined by  $ln_1$  passes through the center of  $crv_1$ ; a conic arc cannot be constructed.

The chord of a conic arc cannot pass through the center of a central conic because this configuration is invalid. If the line defining the chord passes through the center of the conic, then the **\$Failed** symbol will be returned.

An invalid conic arc of the form  $cnarc_1$  has been detected; the arguments cannot be imaginary.

The arguments defining a conic arc cannot be imaginary. If *Descarta2D* detects an invalid object the **\$Failed** symbol will be returned.

ConicArc2D::noChord......[419]

No chord exists between  $ln_1$  and  $crv_1$ ; a conic arc cannot be constructed.

When constructing a conic arc from a line and a conic curve, the line must intersect the conic in two points that form the chord of the conic arc. If the intersection consists of less than two points, or it is on opposite branches of a hyperbola, then the **\$Failed** symbol will be returned.

An invalid conic arc of the form  $cnarc_1$  has been detected; the control points cannot be collinear.

The three control points defining a conic arc cannot be collinear. *Descarta2D* will return the **\$Failed** symbol whenever the arguments to an object are determined to be invalid.

An invalid conic arc of the form  $cnarc_1$  has been detected; the value of rho must be in the range 0<rho<1.

The value of  $\rho$  determines the shape and type of the conic arc. When  $0 < \rho < 1/2$  an elliptic conic arc is created, when  $\rho = 1/2$  a parabolic conic arc is created and when  $1/2 < \rho < 1$  a hyperbolic conic arc is created. *Descarta2D* will return the **\$Failed** symbol whenever the arguments to an object are determined to be invalid.

# D2DExpressions2D

D2DExpressions2D:badTol......[431]

The tolerance tol is not a valid tolerance specification; the default tolerance,  $10^{-10},\,\rm will$  be used.

Tolerance values used to query expressions must be numbers greater than or equal to zero.

# D2DMaster\$2D

D2DMaster\$2D::loaded.....[469]

The package 'D2DMaster2D' has already been loaded.

The package D2DMaster2D.m defines the symbol names associated with each *Descarta2D* package so that it can be loaded automatically when referenced. This file only needs to be loaded once; subsequent requests to load the file will be ignored and will cause no harm.

The path to 'D2DMaster2D.m' cannot be found; unable to initialize Descarta2D.

The package D2DMaster2D.m defines the symbol names associated with each *Descarta2D* package so that it can be loaded automatically when referenced. This error indicates that the software has not been installed correctly.

```
D2DMaster$2D::tooManyPaths......[469]
More than one path to 'D2DMaster2D.m' was found; using path-name.
```

The package D2DMaster2D.m defines the symbol names associated with each *Descarta2D* package so that they can be loaded automatically when referenced. This error indicates that the software has found more than one copy of this file and may suggest that the software has not been installed correctly.

# Directrices2D

Directrices2D::circular.....[413] The ellipse *ellipse*<sub>1</sub> is circular; it has no (finite) directrix lines.

An ellipse whose semi-major and semi-minor axes are equal in length has no (finite) directrix lines. *Descarta2D* will return an empty list.

# Ellipse2D

An invalid ellipse of the form  $ellipse_1$  has been detected; the arguments of an ellipse cannot involve imaginary numbers.

When constructing an ellipse *Descarta2D* verifies that none of the arguments involve imaginary numbers. *Descarta2D* will return the **\$Failed** symbol whenever the arguments to an object are determined to be invalid.

Ellipse2D::invalid.....[423]

An invalid ellipse of the form  $ellipse_1$  has been detected; the length of both the semi-major and semi-minor axes must be positive.

When constructing an ellipse *Descarta2D* verifies that both the semi-major and semi-minor axes have positive lengths. *Descarta2D* will return the **\$Failed** symbol whenever the arguments to an object are determined to be invalid.

The defining geometry or eccentricity is invalid; the eccentricity of an ellipse must be in the range 0<e<1, the foci and vertices cannot be coincident, and the focus cannot lie on the directrix.

An invalid ellipse was specified and Descarta2D will return the \$Failed symbol.

# Hyperbola2D

An invalid hyperbola of the form  $hyp_1$  has been detected; the arguments cannot be imaginary.

When constructing a hyperbola the arguments cannot be imaginary. If imaginary arguments are detected *Descarta2D* will return the **\$Failed** symbol.

An invalid hyperbola of the form  $hyp_1$  has been detected; the lengths of the semi-transverse and semi-conjugate axes must be positive.

When constructing a hyperbola the lengths of both the semi-transverse and the semi-conjugate axes must be positive. *Descarta2D* will return the **\$Failed** symbol whenever the arguments to an object are determined to be invalid.

The defining geometry or eccentricity is invalid; the eccentricity of a hyperbola must be greater than 1, the foci and vertices cannot be coincident and the focus cannot lie on the directrix.

An invalid hyperbola was specified and Descarta2D will return the \$Failed symbol.

# IsNumeric2D

The *funcName* function requires numerical arguments; symbolic arguments are not allowed.

Some *Descarta2D* functions require that their arguments be numeric. These functions will not allow symbolic arguments.

# Line2D

The circles  $\{cir_1, cir_2\}$  are concentric; no radical axis exists.

When specifying the two circles for the construction of a radical axis, the two circles cannot be concentric. If *Descarta2D* detects that concentric circles have been specified in the radical axis construction, it will return the **\$Failed** symbol.

An invalid line of the form  $ln_1$  has been detected; the arguments cannot be imaginary.

The arguments defining a line cannot be imaginary. If *Descarta2D* detects that an object is invalid the **\$Failed** symbol will be returned.

An invalid line of the form  $ln_1$  has been detected; at least one of the first two coefficients must be non-zero.

When defining a line at least one of the first two coefficients, A or B, must be non-zero. *Descarta2D* will return the **\$Failed** symbol whenever the arguments to an object are determined to be invalid.

Since  $pt_1$  is at the center of the conic, no polar line exists.

When creating the polar line of a quadratic with respect to a point, *Descarta2D* verifies that the point is not coincident with the center of the conic curve represented by the quadratic. If the point is at the center of the conic represented by the quadratic *Descarta2D* returns the **\$Failed** symbol.

The expression expr cannot be recognized as a linear polynomial or equation in variables x and y.

When converting a polynomial or equation to a line, the expression representing the line must be recognizable as a linear polynomial or equation. If the expression is not recognizable *Descarta2D* returns the **\$Failed** symbol.

The coordinates  $\{x_1\,,\,y_1\}$  and  $\{x_2\,,\,y_2\}$  are coincident; no valid line can be constructed.

When creating a line through a pair of coordinates or a pair of points, the positions cannot be coincident. *Descarta2D* will return the **\$Failed** symbol if it detects the coordinates are coincident.

# Loci2D

The quadratic is a central conic, but its type cannot be determined.

Due to the nature of the coefficients of the quadratic, the specific conic type cannot be determined; an empty list will be returned.

The eccentricity, e, is invalid; the eccentricity must be positive.

The eccentricity of a conic must be positive; the **\$Failed** symbol will be returned.

The equation represented by the quadratic has no real points; an empty list will be returned.

# MedialEquations2D

MedialEquations2D::coincident......[473]

The objects  $\{obj_1\,,\,obj_2\}$  are coincident; no finite number of medial curves exist.

When two objects are identical the medial points include all the points in the plane and no unique curve locus exists. When this situation occurs *Descarta2D* will return an empty list indicating that no unique curves satisfy the geometric constraints specified.

# Parabola2D

| Parabola2D::imaginary[480]   |
|--|
| An invalid parabola of the form $parabola_1$ has been detected; the arguments cannot be imaginary.   |
| The arguments of a parabola cannot be imaginary. <i>Descarta2D</i> will return the <b>\$Failed</b> symbol whenever the arguments to an object are determined to be invalid.      |
| Parabola2D::invalid[480]   |
| An invalid parabola of the form $parabola_1$ has been detected; the focal length cannot be zero.   |
| The focal length, $f$ , of a parabola cannot be zero. <i>Descarta2D</i> will return the <b>\$Failed</b> symbol whenever the arguments to an object are determined to be invalid. |
| Parabola2D::invptln[483]   |
| The focus $pt_1$ is on the directrix $ln_1$ ; no valid parabola can be constructed.  |
| The focus point of a parabola cannot be on the directrix line. <i>Descarta2D</i> returns the <b>\$Failed</b> symbol when it detects an invalid construction.                     |

# Parameters2D

No chord exists between  $ln_1$  and  $crv_2$ .

The Parameters2D function requires that the defining line intersect the curve in two points. If the line does not intersect the chord, *Descarta2D* will return the **\$Failed** symbol.

# Point2D

No unique intersection point exists; lines  $ln_1$  and  $ln_2$  are coincident.

Coincident lines cannot be intersected. *Descarta2D* will return the **\$Failed** symbol if it detects an attempt to intersect coincident lines.

An invalid point of the form  $pt_1$  has been detected; the coordinates of a point cannot be imaginary.

The coordinates of a point cannot be imaginary. If *Descarta2D* detects an invalid object the **\$Failed** symbol will be returned.

When defining a point that is offset in a direction specified by two points, the direction points cannot be coincident. *Descarta2D* returns the **\$Failed** symbol if the two points are coincident.

Since  $ln_1$  passes through the center of the conic, no pole point exists.

When creating the pole point of a quadratic with respect to a line, *Descarta2D* verifies that the line does not pass through the center of the conic curve represented by the quadratic. If the line does pass through the center *Descarta2D* returns the **\$Failed** symbol.

The sum of the ratio numbers  $\{r_1, r_2\}$  cannot be zero.

When defining a point that divides a segment into a given ratio, the ratio numbers  $r_1$  and  $r_2$  cannot sum to zero. Descarta2D will return the **\$Failed** symbol if the ratio numbers are invalid.

, 1

The quadratic is not a central conic and has no center point. *Descarta2D* will return the **\$Failed** symbol.

The conic underlying  $\mathit{cnarc}$  is not a central conic; it has no center point.

The conic underlying a conic arc is not a central conic and has no center point. *Descarta2D* will return the **\$Failed** symbol.

No intersection point exists; lines  $ln_1$  and  $ln_2$  are parallel.

Parallel lines cannot be intersected. *Descarta2D* will return the **\$Failed** symbol if its detects an attempt to intersect parallel lines.

# Quadratic2D

Two or more of the points are coincident; no valid quadratic pencil exists.

When constructing a quadratic pencil from four points, no pair of points may be coincident. The **\$Failed** symbol with be returned if any pair of points is detected to be coincident.

When defining a quadratic using a point, a line and an eccentricity, *Descarta2D* will report an error if the eccentricity is not positive and return the **\$Failed** symbol.

An invalid quadratic of the form  $quad_1$  has been detected; the arguments cannot be imaginary.

The arguments defining a quadratic cannot be imaginary. If *Descarta2D* detects that an object is invalid the **\$Failed** symbol will be returned.

An invalid quadratic of the form  $quad_1$  has been detected; at least one of the first five coefficients must be non-zero.

At least one of the first five coefficients of a quadratic must be non-zero. The **\$Failed** symbol is returned whenever the arguments to an object are determined to be invalid.

A negative eccentricity,  $expr_1$ , is invalid; no valid quadratic can be constructed.

The eccentricity of a conic must be non-negative. *Descarta2D* will return the **\$Failed** symbol if an invalid eccentricity is specified.

A non-positive focal chord length,  $expr_1$ , is invalid; no valid quadratic can be constructed.

The length of a conic's focal chord must be positive. *Descarta2D* will return the **\$Failed** symbol if an invalid length is specified.

The expression expr cannot be recognized as a quadratic polynomial or equation in variables x and y.

When converting a polynomial or equation to a quadratic, the expression representing the quadratic must be recognizable as a quadratic polynomial or equation. If the expression is not recognizable *Descarta2D* returns the **\$Failed** symbol.
## Segment2D

An invalid line segment of the form  $lnseg_1$  has been detected; the arguments cannot be imaginary.

A line segment with imaginary arguments has been detected. *Descarta2D* will return the **\$Failed** symbol whenever the arguments to an object are determined to be invalid.

An invalid line segment of the form  $lnseg_1$  has been detected; the defining coordinates cannot be coincident.

In order to be valid, a line segment must have two distinct end points, they cannot be coincident. *Descarta2D* will return the **\$Failed** symbol whenever the arguments to an object are determined to be invalid.

## Sketch2D

Setting CurveLength2D $\rightarrow n_1$  is invalid; 'CurveLength2D' must be positive; the current value of CurveLength2D $\rightarrow n_2$  will be retained.

When using the *Mathematica* SetOptions command, any attempt to set the CurveLength2D parameter of the Sketch2D function to a non-positive value will be rejected. The current value of the CurveLength2D parameter will be retained.

No valid objects to sketch.

If there are no valid geometric objects in the list of objects to sketch, *Descarta2D* will output the Sketch2D::noObj message to indicate no graphical output will be plotted.

n object(s) cannot be sketched.

When plotting objects using the Sketch2D command, *Descarta2D* will count the number of objects that have symbolic arguments. Such objects cannot be plotted and will not be included in the graphics that are displayed.

### Solve2D

An infinite number of solutions exist; only independent solutions will be returned.

When solving a system of equations some solutions may exist in which the solutions are interrelated functions of each other. Such solutions will not be returned.

Option MaxSeconds2D-> $n_1$  is invalid; 'MaxSeconds2D' must be positive; the current value of MaxSeconds2D-> $n_2$  will be retained.

When setting the MaxSeconds2D option of the Solve2D command, the option value must be positive.

The equations could not be solved in MaxSeconds2D-> $n_1$ , an empty list of solutions will be returned; using approximate numbers may produce a more complete list of solutions.

Some equations are too complex to be solved in the time allowed by the *Descarta2D* Solve2D command. An empty list of solutions will be returned if the maximum time elapses before a solution is found. To increase the maximum time allowed use the SetOptions command. For example,

SetOptions[Solve2D, MaxSeconds2D->60].

will set the time limit to 60 seconds.

## SolveTriangle2D

Two valid solutions exist for this configuration; set the alternate solution option to *logical* to compute the other configuration.

When computing a triangle configuration, *Descarta2D* will display this warning if more than one solution is valid. The *logical* will either be **True** or **False** indicating the setting required to produce the alternate configuration.

The triangle configuration is under-constrained; a valid configuration with the triangle's perimeter arbitrarily set to 1 will be computed.

When computing a triangle configuration consisting of angles only, *Descarta2D* will display this warning to indicate that the length of the sides are arbitrarily set, being correct for the given angles.

The triangle configuration is under-constrained; three constraints are expected.

At least three parameters are needed to compute a triangle configuration. *Descarta2D* will return **\$Failed** if a configuration is under-constrained.

The configuration of sides and/or angles specified is invalid; no triangle can be constructed.

An invalid triangle configuration has been specified. Descarta2D will return \$Failed.

## TangentConics2D

Two or more of the defining points or lines are coincident; no proper conic can be constructed.

When constructing a tangent conic from defining points, all of the points must be unique; if any of the points are coincident, *Descarta2D* will return an empty list.

Three or more of the defining points are collinear; no proper conic can be constructed.

When constructing a tangent conic from defining points, no triple of three points may be collinear; if any triple is collinear *Descarta2D* will return an empty list.

Three or more of the tangent lines are concurrent; no proper conic can be constructed.

When constructing a tangent conic from defining lines, no triple of lines can be concurrent (meet in a point); if any triple is concurrent *Descarta2D* will return an empty list.

One of the points is on more than one of the tangent lines; no proper conic can be constructed.

When constructing a tangent conic from points and lines, each point is allowed to be on at most one of the tangent lines; if any point is on more than one line, *Descarta2D* will return an empty list.

Three or more of the defining lines are parallel; no proper conic can be constructed.

When constructing a tangent conic from defining lines, no triple of lines can be parallel; if any triple is parallel *Descarta2D* will return an empty list.

Two or more of the points are on a tangent line; no proper conic can be constructed.

When constructing a tangent conic from points and lines, each line can have at most one point on it; if any line has more than one point one it, *Descarta2D* will return an empty list.

# Transform2D

The scale factor  $\boldsymbol{s}$  is invalid; the scale factor must be positive.

The scale factor, s, for a scaling transformation must be positive. Descarta2D will return the **\$Failed** symbol if a non-positive scale factor is specified.

## **Triangle2D**

An invalid triangle of the form  $triangle_1$  has been detected; the arguments cannot be imaginary.

The arguments of a triangle cannot be imaginary. *Descarta2D* will return the **\$Failed** symbol whenever the arguments to an object are determined to be invalid.

An invalid triangle of the form  $triangle_1$  has been detected; the vertex points cannot be collinear.

The vertex points of a triangle cannot be collinear. *Descarta2D* will return the **\$Failed** symbol whenever the arguments to an object are determined to be invalid.

Two of the lines  $\{ln_1, ln_2, ln_3\}$  are parallel, or the three are concurrent; no triangle exists.

When defining a triangle by three lines, the lines must intersect in three distinct points. If any pair of lines are parallel, or the three lines are concurrent, *Descarta2D* will return the **\$Failed** symbol.

Part VII

Packages

# D2DArc2D

The package D2DArc2D implements the Arc2D object.

# Initialization

```
BeginPackage["D2DArc2D`", {"D2DCircle2D`", "D2DExpressions2D`",
"D2DGeometry2D`", "D2DLine2D`", "D2DMaster2D`", "D2DNumbers2D`",
"D2DPoint2D`", "D2DSketch2D`", "D2DTransform2D`"}];
D2DArc2D::usage=
    "D2DArc2D is a package that implements the Arc2D object.";
Arc2D::usage=
    "Arc2D[{x0,y0},{x1,y1},B] is the standard form of an arc with start
point (x0,y0), end point (x1,y1) and positive bulge factor 'B'.";
Bulge2D::usage=
    "Bulge2D[arc] returns the bulge factor of an arc.";
Complement2D::usage=
    "Complement2D is a keyword required in Arc2D[arc, Complement2D].";
Begin["'Private'"];
```

# Description

#### Representation

Arc2D[ $\{x_0, y_0\}, \{x_1, y_1\}, B$ ]  $\blacksquare$  Standard representation of an arc in *Descarta2D*. The first argument is a list of coordinates representing the start point of the arc. The second argument is a list of coordinates representing the end point of the arc. The third argument is a scalar representing the bulge factor of the arc, B > 0. The arc is traversed counter-clockwise from  $P_1$  to  $P_2$ . The bulge factor is the ratio of the arc's height, h, to half the chord length, d/2; so B = 2h/d.

#### **Evaluation**

Arc2D[ $\{x_0, y_0\}$ ,  $\{x_1, y_1\}$ , B] [t]  $\blacksquare$  Evaluates an arc at a parameter value, t, and returns a list of coordinates  $\{x, y\}$ . Parameters in the range  $0 \le t \le 1$  cover the complete span of the arc.

```
Arc2D[{x0_,y0_}, {x1_,y1_}, B_][t_?IsScalar2D] :=
Module[{arc,h,k,beta},
arc=Arc2D[{x0,y0}, {x1,y1},B];
{h,k}=Coordinates2D[arc];
beta=Angle2D[arc];
{h+(x0-h)*Cos[beta*t]-(y0-k)*Sin[beta*t],
k+(x0-h)*Sin[beta*t]+(y0-k)*Cos[beta*t]}];
```

#### Graphics

Provides graphics primitives for an arc by extending the *Mathematica* Display command. Executed when the package is loaded.

```
SetDisplay2D[
Arc2D[{x0_,y0_},{x1_,y1_},B_][{t1_?IsScalar2D,t2_?IsScalar2D}],
Circle[Coordinates2D[Arc2D[{x0,y0},{x1,y1},B]],
Radius2D[Arc2D[{x0,y0},{x1,y1},B]],
PrimaryAngleRange2D[{
            Angle2D[Arc2D[{x0,y0},{x1,y1},B],t1],
            Angle2D[Arc2D[{x0,y0},{x1,y1},B],t2]]]];
SetDisplay2D[
Arc2D[{x0_,y0_},{x1_,y1_},B_],
Circle[Coordinates2D[Arc2D[{x0,y0},{x1,y1},B]],
Radius2D[Arc2D[{x0,y0},{x1,y1},B]],
PrimaryAngleRange2D[{x0,y0},{x1,y1},B]],
PrimaryAngleRange2D[Arc2D[{x0,y0},{x1,y1},B]],
```

#### Validation

 $\operatorname{Arc2D}[\{x_0, y_0\}, \{x_1, y_1\}, B] \blacksquare$  Detects an arc with imaginary arguments and returns the **\$Failed** symbol. If the imaginary parts are insignificant, they are removed.

```
Arc2D::imaginary=
    "An invalid arc of the form 'Arc2D['1', '2', '3']' has been detected;
the arguments cannot be imaginary.";
Arc2D[{x0_,y0_}, {x1_,y1_},B_] :=
    (Arc2D @@ ChopImaginary2D[Arc$2D[{x0,y0}, {x1,y1},B]]) /;
(FreeQ[{x0,y0,x1,y1,B},_Pattern] &&
    IsTinyImaginary2D[{x0,y0,x1,y1,B}]);
Arc2D[{x0_,y0_}, {x1_,y1_},B_] :=
    (Message[Arc2D::imaginary, {x0,y0}, {x1,y1},B];$Failed) /;
(FreeQ[{x0,y0,x1,y1,B},_Pattern] &&
    IsComplex2D[{x0,y0,x1,y1,B},0]);
```

 $\operatorname{Arc2D}[\{x_0, y_0\}, \{x_1, y_1\}, B] \blacksquare$  Detects an arc with a negative bulge factor and returns an arc with the defining points interchanged and the positive bulge factor.

```
Arc2D[ {x0_,y0_}, {x1_,y1_}, B_?IsNegative2D] :=
Arc2D[ {x1,y1}, {x0,y0}, -B];
```

 $\operatorname{Arc2D}[\{x_0, y_0\}, \{x_1, y_1\}, B] \blacksquare$  Detects an arc with a zero bulge factor and returns the **\$Failed** symbol.

```
Arc2D::invalid=
    "An invalid arc of the form 'Arc2D['1', '2', '3']' has been detected;
the bulge factor must be positive and the defining points must be
distinct.";
Arc2D[{x0_,y0_},{x1_,y1_},B_] :=
    (Message[Arc2D::invalid,{x0,y0},{x1,y1},B];$Failed) /;
(FreeQ[{x0_,y0,x1,y1,B},_Pattern] &&
    IsZero2D[B,0]);
```

 $\operatorname{Arc2D}[\{x_0, y_0\}, \{x_1, y_1\}, B] \square$  Detects an arc whose defining points are coincident and returns the **\$Failed** symbol.

```
Arc2D[{x0_,y0_},{x1_,y1_},B_] :=
   (Message[Arc2D::invalid,{x0,y0},{x1,y1},B];$Failed) /;
   (FreeQ[{x0,y0,x1,y1,B},_Pattern] &&
   IsZero2D[Distance2D[{x0,y0},{x1,y1}]]);
```

**IsValid2D**[*arc*] ■ Returns **True** for a syntactically valid arc.

# Scalars

#### Angular Span of an Arc

Angle2D[arc] Computes the angular span of an arc. The result is returned in radians.

Angle2D[Arc2D[{x0\_,y0\_},{x1\_,y1\_},B\_]] := 4\*ArcTan[B];

#### Angle at Parameter on an Arc

Angle2D[*arc*, *t*]  $\blacksquare$  Computes the angle between a line through the arc center parallel to the +*x*-axis and a line through a point at a parameter value, *t*, on the arc. For example, Angle2D[*arc*, 0] gives the start angle,  $\theta_1$ , and Angle2D[*arc*, 1] gives the end angle,  $\theta_2$ .

#### **Bulge Factor of an Arc**

**Bulge2D**[*arc*] ■ Returns the bulge factor of an arc.

```
Bulge2D[Arc2D[{x0_,y0_},{x1_,y1_},B_]] := B;
```

#### Primary Angle Range

**PrimaryAngleRange2D**[arc]  $\blacksquare$  Computes a list of two primary angles measured counterclockwise from the +x-axis to the defining points of an arc. The arc is traversed counterclockwise from the first angle to the second.

#### **Radius of an Arc**

**Radius2D**[*arc*] ■ Computes the radius of an arc.

```
Radius2D[Arc2D[{x0_,y0_},{x1_,y1_},B_]] :=
Sqrt[(x0-x1)^2+(y0-y1)^2]*(B+1/B)/4;
```

# Transformations

#### Reflect

**Reflect2D**[*arc*, *line*] ■ Reflects an arc in a line.

```
Reflect2D[Arc2D[{x0_,y0_}, {x1_,y1_},B_],L:Line2D[a_,b_,c_]] :=
Arc2D[Reflect2D[{x1,y1},L],Reflect2D[{x0,y0},L],B];
```

#### Rotate

Rotate2D[*arc*,  $\theta$ , *coords*]  $\blacksquare$  Rotates an arc by an angle  $\theta$  about a position given by a coordinate list. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

#### Scale

Scale2D[*arc*, *s*, *coords*] ■ Scales an arc from a position given by coordinates. If the position is omitted, it defaults to the origin (see D2DTransform2D.nb).

#### Translate

Translate2D[arc,  $\{u, v\}$ ]  $\blacksquare$  Translates an arc delta distance.

# Construction

#### Center Point of an Arc

 $Point2D[arc] \blacksquare Constructs the center point of the arc.$ 

```
Point2D[Arc2D[ {x0_,y0_}, {x1_,y1_}, B_]] :=
Module[ {K},
K=(1/B-B)/4;
Point2D[ { (x0+x1)/2+K*(y0-y1), (y0+y1)/2-K*(x0-x1) } ];
```

## **Circle from Arc**

 $Circle2D[arc] \blacksquare$  Constructs the circle associated with an arc.

```
Circle2D[A:Arc2D[{x0_,y0_}, {x1_,y1_},B_]] :=
Circle2D[Coordinates2D[A],Radius2D[A]];
```

#### **Complement Arc**

Arc2D[arc, Complement2D] Constructs an arc that is the complement of a given arc.

```
Arc2D[Arc2D[{x0_,y0_},{x1_,y1_},B_],Complement2D] :=
Arc2D[{x1,y1},{x0,y0},1/B];
```

#### Arc from Center Point, Radius and Span

Arc2D[*point*, r, { $\theta_1$ ,  $\theta_2$ }]  $\blacksquare$  Constructs an arc from a center point, radius and angular span range. The arc is defined counter-clockwise from the start point associated with  $\theta_1$  to the end point associated with  $\theta_2$ .

```
Arc2D::invalidSpan=
   "The angular span of the arc is invalid; the span cannot be an integer
multiple of 2Pi radians.";
Arc2D::invalidRadius=
   "The radius, '1', of the arc is invalid; the radius must be positive.";
Arc2D[Point2D[c:{h_,k_}], r_?IsZeroOrNegative2D,
      {t0_?IsScalar2D,t1_?IsScalar2D}] :=
   (Message[Arc2D::invalidRadius,r]; $Failed);
Arc2D[Point2D[c:{h_,k_}], r_?IsScalar2D,
      {t0_?IsScalar2D,t1_?IsScalar2D}] :=
   (Message[Arc2D::invalidSpan];$Failed) /;
IsZero2D[Distance2D[Circle2D[c,r][t0],Circle2D[c,r][t1]]];
Arc2D[Point2D[c:{h_,k_}], r_?IsScalar2D,
      {t0_?IsScalar2D,t1_?IsScalar2D}] :=
   Module[{T0,T1,p0,p1,d,pm,H,B},
      {T0,T1}=PrimaryAngleRange2D[{t0,t1}];
      p0=Circle2D[c,r][T0];
      p1=Circle2D[c,r][T1];
      d=Distance2D[p0,p1];
      pm=Circle2D[c,r][(T0+T1)/2];
      H=Distance2D[(p0+p1)/2,pm];
      B=2*H/d;
      Arc2D[p0,p1,B] ];
```

#### Arc from Defining Points and Entry Angle

 $\operatorname{Arc2D}[\{point, \theta\}, point] \blacksquare$  Constructs an arc from the start and end points and the angle between the chord and the entry vector. The angle cannot be an integer multiple of  $\pi$  radians. The angle is positive for counter-clockwise arcs and negative for clockwise arcs.

```
Arc2D::invalidEntryAngle=
    "The entry angle of the arc is invalid; the entry angle cannot be an
integer multiple of Pi radians.";
Arc2D::invalidCoincident=
    "The defining points are coincident; an arc cannot be constructed.";
```

```
Arc2D[{P0:Point2D[p0:{x0_,y0_}],A_?IsScalar2D},
    P1:Point2D[p1:{x1_,y1_}]] :=
Which[
    IsZero2D[PrimaryAngle2D[A,Pi]],
    Message[Arc2D::invalidEntryAngle];$Failed,
    IsCoincident2D[P0,P1],
    Message[Arc2D::invalidCoincident];$Failed,
    True,
        Arc2D[p0,p1,Tan[A/2]] ];
```

### Arc from Three Points

 $\operatorname{Arc2D}[point, point, point] \blacksquare$  Constructs an arc through three points. The first and the third points are the start and end points of the arc, respectively, and the second point is a general point on the arc. The private function Minor\$2D is the 2D vector cross-product.

```
Arc2D::invalidCollinear=
    "The three defining points are collinear; an arc cannot be constructed.";
Minor$2D[{u1_,v1_},{u2_,v2_}] := u1*v2-u2*v1;
Arc2D[P0:Point2D[p0:{x0_,y0_}],
    Pon:Point2D[p0:{x1_,y1_}]] :=
    Module[{s,c,B},
    Which[
        IsCollinear2D[P0,Pon,P1],
        Message[Arc2D::invalidCollinear];$Failed,
        True,
        s=Minor$2D[pon-p0,p1-pon];
        c=Dot[pon-p0,p1-pon];
        B=s/(c+Sqrt[c^2+s^2]);
        Arc2D[p0,p1,B]] ];
```

#### Arc from Center, Radius and Ray End Points

 $\operatorname{Arc2D}[point, r, \{point, point\}] \blacksquare$  Constructs an arc given the center point, radius and ray end points. The ray end points do not have to be on the arc, but they cannot be coincident with the center point.

```
Arc2D[P:Point2D[{h_,k_}],r_?IsScalar2D,
        {P0:Point2D[{x0_,y0_}],P1:Point2D[{x1_,y1_}]}] :=
    Which[
        IsZeroOrNegative2D[r],
        Message[Arc2D::invalidRadius,r];$Failed,
        IsCoincident2D[P,P0] || IsCoincident2D[P,P1],
        Message[Arc2D::invalidCoincident];$Failed,
        True,
        Arc2D[Point2D[{h,k}],r,
             {ArcTan[x0-h,y0-k],ArcTan[x1-h,y1-k]}] ]
```

# Epilogue

End[ ]; (\* end of "`Private" \*)
EndPackage[ ]; (\* end of "D2DArc2D`" \*)

# D2DArcLength2D

The package D2DArcLength2D provides functions for computing the arc length of *Descarta2D* objects.

# Initialization

```
BeginPackage["D2DArcLength2D'", {"D2DArc2D'", "D2DCircle2D'",
"D2DConicArc2D'", "D2DEllipse2D'", "D2DExpressions2D'", "D2DGeometry2D'",
"D2DHyperbola2D'", "D2DLine2D'", "D2DNumbers2D'", "D2DParabola2D'",
"D2DSegment2D'", "D2DTriangle2D'"}];
D2DArcLength2D::usage=
    "D2DArcLength2D is a package for computing the arc length of curves.";
ArcLength2D[:usage=
    "ArcLength2D[curve,{t0,t1}] computes the arc length of a curve between
two parameters.";
Circumference2D[circle] computes the circumference of a circle.
Circumference2D[ellipse] computes the circumference of an ellipse.";
Perimeter2D::usage=
    "Perimeter2D[triangle] computes the perimeter of a triangle.";
Span2D::usage=
    "Span2D[arc] computes the span (arc length) of an arc; N[Span2D[cnarc]]
numerically computes the span (arc length) of a conic arc.";
```

```
Begin["'Private'"];
```

# Arc Length

#### Arc

**Span2D**[*arc*]  $\blacksquare$  Computes the arc length of a complete span of an arc.

 $\operatorname{ArcLength2D}[\operatorname{arc}, \{\theta_1, \theta_2\}] \blacksquare$  Computes the arc length of an arc between two parameters.

#### Circle

**Circumference2D**[*circle*] ■ Computes the circumference of a circle.

Circumference2D[Circle2D[{h\_,k\_},r\_]] := 2\*Pi\*r;

ArcLength2D[*circle*,  $\{\theta_1, \theta_2\}$ ]  $\blacksquare$  Computes the arc length of a circle between two parameters.

```
ArcLength2D[Circle2D[{h_,k_},r_],{t1_?IsScalar2D,t2_?IsScalar2D}] :=
Module[{T1,T2},
        {T1,T2}=PrimaryAngleRange2D[{t1,t2}];
        r*(T2-T1) ];
```

#### **Conic Segment**

IsNumeric2D[C1,Span2D];

Span2D[cnarc] //N  $\blacksquare$  Computes the arc length of a complete span of a conic arc numerically.

```
N[Span2D[C1:ConicArc2D[{x0_,y0_}, {xA_,yA_}, {x1_,y1_},p_]]] :=
NArcLength$2D[C1,{0,1},$MachinePrecision] /;
IsNumeric2D[C1,Span2D];
N[Span2D[C1:ConicArc2D[{x0_,y0_}, {xA_,yA_}, {x1_,y1_},p_]],n_] :=
NArcLength$2D[C1,{0,1},n] /;
```

ArcLength2D[*cnarc*,  $\{\theta_1, \theta_2\}$ ] //N  $\blacksquare$  Computes the arc length of a conic arc between two parameters numerically.

```
N[ArcLength2D[C1:ConicArc2D[{x0_,y0_},{xA_,yA_},{x1_,y1_},p_],
	{t1_?IsScalar2D,t2_?IsScalar2D}]] :=
NArcLength$2D[C1,{t1,t2},$MachinePrecision] /;
IsNumeric2D[{C1,t1,t2},ArcLength2D];
N[ArcLength2D[C1:ConicArc2D[{x0_,y0_},{xA_,yA_},{x1_,y1_},p_],
	{t1_?IsScalar2D,t2_?IsScalar2D}],
n_] :=
NArcLength$2D[C1,{t1,t2},n] /;
IsNumeric2D[{C1,t1,t2},ArcLength2D];
```

#### Ellipse

Circumference2D[*ellipse*] ■ Computes the circumference of an ellipse.

```
Circumference2D[E1:Ellipse2D[{h_,k_},a_,b_,theta_]] :=
ArcLength2D[E1,{0,2Pi}];
```

ArcLength2D[*ellipse*,  $\{\theta_1, \theta_2\}$ ]  $\blacksquare$  Computes the arc length of an ellipse between two parameters.

#### Hyperbola

The private function ArcLengthHyperbola $2D[hyperbola, \{0, t\}]$  computes the arc length of a hyperbola between parameter values 0 and t. The result may be positive or negative, depending on the value given for t.

```
ArcLengthHyperbola$2D[Hyperbola2D[{h_,k_},a_,b_,theta_],{0,t_}] :=
Re[-I*b*EllipticE[I*ArcCosh[Sqrt[a^2+b^2]/a]*t,1+a^2/b^2]];
```

ArcLength2D[hyperbola,  $\{\theta_1, \theta_2\}$ ]  $\blacksquare$  Computes the arc length of a hyperbola between two parameters.

#### Line

 $\operatorname{ArcLength2D}[line, \{t_1, t_2\}] \blacksquare$  Computes the arc length of a line between two parameters.

```
ArcLength2D[Line2D[a_,b_,c_], {t1_?IsScalar2D,t2_?IsScalar2D}] :=
Module[{L},
L=t2-t1;
If[IsNegative2D[L],-L,L] ];
```

#### Line Segment

ArcLength2D[lnseg, { $t_1$ ,  $t_2$ }]  $\blacksquare$  Computes the arc length of a line segment between two parameters. The function Length2D[lnseg] computes the length of a complete line segment (defined in package D2DSegment2D).

### Parabola

ArcLength2D[parabola,  $\{t_1, t_2\}$ ]  $\blacksquare$  Computes the arc length of a parabola between two parameter values.

#### Triangle

**Perimeter2D**[*triangle*] ■ Computes the perimeter of a triangle.

```
Perimeter2D[Triangle2D[{x1_,y1_},{x2_,y2_},{x3_,y3_}]] :=
    Sqrt[(x1-x2)^2+(y1-y2)^2]+
    Sqrt[(x1-x3)^2+(y1-y3)^2]+
    Sqrt[(x2-x3)^2+(y2-y3)^2];
```

# Arc Length (Numerical)

#### **Parametric Curves**

The private function NArcLength\$2D[curve,  $\{t_1, t_2\}$ ] numerically computes the arc length of a parametric curve between two parameter values. The function uses numerical integration, so the arguments of the function must be numerical. The third argument, n, specifies the numerical precision.

```
NArcLength$2D[obj_,{t1_,t2_},n_] :=
Module[{t,Dx,Dy,L},
        {Dx,Dy}=Map[D[#,t]&,obj[t]];
        L=NIntegrate[Sqrt[Dx^2+Dy^2],{t,t1,t2},WorkingPrecision->n];
        If[IsNegative2D[L],-L,L]];
```

# Epilogue

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DArcLength2D`" *)
```

# D2DArea2D

The package D2DArea2D computes areas associated with Descarta2D objects.

# Initialization

```
BeginPackage["D2DArea2D`", {"D2DArc2D`", "D2DCircle2D`", "D2DConicArc2D`",
"D2DEllipse2D`", "D2DExpressions2D`", "D2DGeometry2D`", "D2DHyperbola2D`",
"D2DLine2D`", "D2DNumbers2D`", "D2DParabola2D`", "D2DPoint2D`",
"D2DArea2D::usage=
    "D2DArea2D is a package for computing areas.";
Area2D::usage=
    "Area2D[object] computes the area of a closed object";
SectorArea2D::usage=
    "SectorArea2D[object,{t1,t2}] computes the area of a sector of an
    object.";
SegmentArea2D[object,{t1,t2}] computes the area of a segment of an
    object.";
Begin["`Private`"];
```

# Areas Associated with an Arc

#### Area

**Area2D**[*arc*] ■ Computes the area between an arc and its chord.

```
Area2D[A:Arc2D[{x0_,y0_},{x1_,y1_},B_]] :=
SegmentArea2D[Circle2D[A],PrimaryAngleRange2D[A]];
```

## Areas Associated with a Circle

#### Area

**Area2D**[*circle*] ■ Computes the area of a circle.

Area2D[Circle2D[{h\_,k\_},r\_]] := Pi\*r^2;

#### Sector Area

SectorArea2D[*circle*,  $\{\theta_1, \theta_2\}$ ]  $\blacksquare$  Computes the area of a circle sector defined by two parameters.

```
SectorArea2D[Circle2D[{h_,k_},r_],{t1_?IsScalar2D,t2_?IsScalar2D}] :=
Module[{T1,T2},
    {T1,T2}=PrimaryAngleRange2D[{t1,t2}];
    (T2-T1)*r^2/2 ];
```

#### Segment Area

SegmentArea2D[*circle*,  $\{\theta_1, \theta_2\}$ ]  $\blacksquare$  Computes the area of a circle segment defined by two parameters.

```
SegmentArea2D[Circle2D[{h_,k_},r_],{t1_?IsScalar2D,t2_?IsScalar2D}] :=
Module[{T1,T2,theta},
        {T1,T2}=PrimaryAngleRange2D[{t1,t2}];
        theta=T2-T1;
        r^2*(theta-Sin[theta])/2 ];
```

# Areas Associated with a Conic Arc

#### Area

 $Area2D[cnarc] \blacksquare$  Computes the area between a conic arc and its chord for a conic arc in a standard position. The first case is for parabolas; the second case is for ellipses and hyperbolas.

```
Area2D[ConicArc2D[{0,0}, {a_,b_}, {d_,0},p_]] :=
Module[{A},
A=d*b/3;
If[IsNegative2D[A],-A,A] ] /;
IsZero2D[p-1/2];
Area2D[ConicArc2D[{0,0}, {a1_,b1_}, {d1_,0},p_]] :=
Module[{b,d,r},
b*d*p*(p*r+(-1+p)^2*Log[(1-p)/(p+r)])/(2*r^3) //.
{r->Sqrt[-1+2p],b->Sqrt[b1^2],d->Sqrt[d1^2]}] /;
Not[IsZero2D[p-1/2]];
```

**Area2D**[*cnarc*]  $\blacksquare$  Computes the area between a conic arc and its chord. Notice that the *x*-coordinate of the apex point in standard position has no bearing on the area, and, therefore, is not computed.

```
Area2D[ConicArc2D[p0:{x0_,y0_},pA:{xA_,yA_},p1:{x1_,y1_},p_]] :=
Module[{a,b,d},
b=Distance2D[Point2D[pA],Line2D[p0,p1]];
d=Distance2D[p0,p1];
Area2D[ConicArc2D[{0,0},{a,b},{d,0},p]] ];
```

## Areas Associated with an Ellipse

#### Area

**Area2D**[*ellipse*] ■ Computes the area of a complete ellipse.

```
Area2D[Ellipse2D[{h_,k_},a_,b_,alpha_]] := Pi*a*b;
```

#### Sector Area

SectorArea2D[*ellipse*,  $\{0, \theta\}$ ]  $\blacksquare$  Computes the area of an ellipse sector between parameter values 0 and  $\theta$ .

```
SectorArea2D[E1:Ellipse2D[{h_,k_},a_,b_,alpha_],{0,t_?IsScalar2D}] :=
Module[{T=PrimaryAngle2D[t]},
Which[
IsZero2D[T], Pi*a*b,
IsNegative2D[Pi-T], Pi*a*b/2+SectorArea2D[E1,{0,T-Pi}],
True, a*b*(Pi-2*ArcSin[Cos[t]])/4]];
```

SectorArea2D[*ellipse*,  $\{\theta_1, \theta_2\}$ ]  $\blacksquare$  Computes the area of an ellipse sector between two parameter values.

#### Segment Area

SegmentArea2D[*ellipse*,  $\{\theta_1, \theta_2\}$ ]  $\blacksquare$  Computes the area of an ellipse segment between two parameter values.

# Areas Associated with a Hyperbola

#### Sector Area

SectorArea2D[hyperbola,  $\{t_1, t_2\}$ ]  $\blacksquare$  Computes the area of a hyperbola sector between two parameter values.

#### Segment Area

SegmentArea2D[hyperbola,  $\{t_1, t_2\}$ ]  $\blacksquare$  Computes the area of a hyperbola segment between two parameters.

## Areas Associated with a Parabola

#### Segment Area

SegmentArea2D[parabola,  $\{t_1, t_2\}$ ]  $\blacksquare$  Computes the area of a segment of a parabola between two parameters.

# Areas Associated with a Triangle

## Area

**Area2D**[*triangle*] ■ Computes the area of a triangle.

```
Area2D[Triangle2D[{x1_,y1_},{x2_,y2_},{x3_,y3_}]] :=
Module[{A},
A=Det[{{x1,y1,1},{x2,y2,1},{x3,y3,1}}]/2;
If[IsNegative2D[A],-A,A]];
```

# **Epilogue**

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DArea2D`" *)
```

# D2DCircle2D

The package D2DCircle2D implements the Circle2D object.

# Initialization

```
BeginPackage["D2DCircle2D`", {"D2DExpressions2D`", "D2DGeometry2D`",
"D2DLine2D`", "D2DMaster2D`", "D2DNumbers2D`", "D2DPoint2D`",
"D2DQuadratic2D`", "D2DSketch2D`", "D2DTransform2D`"}];
D2DCircle2D::usage=
    "D2DCircle2D is a package that implements the Circle2D object.";
Circle2D::usage=
    "Circle2D[{h,k},r] is the standard form of a circle with radius 'r'
centered at {h,k}.";
Radius2D::usage=
    "Radius2D[circle] gives the radius of a circle.";
Begin["`Private`"];
```

# Description

#### Representation

 $Circle2D[{h, k}, r] \blacksquare$  Standard representation of a circle in *Descarta2D*. The center of the circle is (h, k) and the radius is r.

#### Equation

Quadratic2D[*circle*] ■ Constructs the quadratic representing the equation of a circle.

```
Quadratic2D[Circle2D[{h_,k_},r_]] :=
Quadratic2D[1,0,1,-2*h,-2*k,h^2+k^2-r^2];
```

#### **Evaluation**

Circle2D[{h, k}, r][ $\theta$ ]  $\blacksquare$  Evaluates a parameter,  $\theta$ , on a circle and returns a coordinate list {x, y}. Parameters in the range  $0 \le \theta < 2\pi$  cover a complete circle.

```
Circle2D[{h_,k_},r_][t_?IsScalar2D] := {h+r*Cos[t],k+r*Sin[t]};
```

#### Graphics

Provides graphics primitives for a circle by extending the *Mathematica* Display command. Executed when the package is loaded.

```
SetDisplay2D[
    Circle2D[{h_,k_},r_][{t1_?IsScalar2D,t2_?IsScalar2D}],
    Circle[{h,k},r,PrimaryAngleRange2D[{t1,t2}]] ];
SetDisplay2D[
    Circle2D[{h_,k_},r_],
    Circle[{h,k},r] ];
```

#### Validation

 $Circle2D[{h, k}, r] \blacksquare$  Detects a circle with imaginary arguments and returns the **\$Failed** symbol. If the imaginary parts are insignificant, they are removed.

```
Circle2D::imaginary=
    "An invalid circle of the form 'Circle2D['1','2']' has been detected;
the arguments cannot be imaginary.";
Circle2D[{h_,k_},r_] :=
    (Circle2D @@ ChopImaginary2D[Circle$2D[{h,k},r]]) /;
(FreeQ[{h,k,r},_Pattern] && IsTinyImaginary2D[{h,k,r}]);
Circle2D[{h_,k_},r_] :=
    (Message[Circle2D::imaginary,{h,k},r];$Failed) /;
(FreeQ[{h,k,r},_Pattern] && IsComplex2D[{h,k,r},0]);
```

Circle2D[ $\{h, k\}, r$ ]  $\blacksquare$  Detects a circle whose radius is non-positive and returns the **\$Failed** symbol.

```
Circle2D::invalid=
  "An invalid circle of the form 'Circle2D['1', '2']' has been detected;
the radius must be positive.";
Circle2D[{h_,k_},r_] :=
   (Message[Circle2D::invalid,{h,k},r];$Failed) /;
(FreeQ[{h,k,r},_Pattern] && IsZeroOrNegative2D[r,0]);
```

**IsValid2D**[*circle*] ■ Verifies that a circle is syntactically valid.

```
IsValid2D[Circle2D[{h_?IsScalar2D,k_?IsScalar2D},r_?IsScalar2D]] := True;
```

## Scalars

#### Distance Point/Circle

Distance2D[*point*, *circle*] Computes the distance between a point and a circle.

```
Distance2D[Point2D[{x_,y_}],Circle2D[{h_,k_},r_]] :=
Sqrt[(Distance2D[{x,y},{h,k}]-r)^2];
```

#### Radius

**Radius2D**[*circle*] ■ Returns the radius of a circle.

 $Radius2D[Circle2D[{h_,k_},r_]] := r;$ 

# Transformations

#### Reflect

**Reflect2D**[*circle*, *line*] ■ Reflects a circle in a line.

```
Reflect2D[Circle2D[{h_,k_},r_],L:Line2D[a_,b_,c_]] :=
Circle2D[Reflect2D[{h,k},L],r];
```

#### Rotate

Rotate2D[*circle*,  $\theta$ , *coords*]  $\blacksquare$  Rotates a circle by an angle  $\theta$  about a position specified by a coordinate list. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

#### Scale

Scale2D[*circle*, *s*, *coords*] ■ Scales a circle from a coordinate position. If the position is omitted, it defaults to the origin (see D2DTransform2D.nb).

#### Translate

Translate2D[*circle*,  $\{u, v\}$ ]  $\blacksquare$  Translates a circle delta distance.

## **Point Construction**

#### **Center Point**

**Point2D**[*circle*] ■ Constructs the center point of the circle.

```
Point2D[Circle2D[{h_,k_},r_]] := Point2D[{h,k}];
```

#### **Pole Point**

**Point2D**[*line*, *circle*]  $\blacksquare$  Constructs a point that is the pole point of a line with respect to a circle. If the line is tangent to the circle then the point is the tangency point.

```
Point2D[L1:Line2D[a1_,b1_,c1_],C2:Circle2D[{h2_,k2_},r2_]] :=
Point2D[L1,Quadratic2D[C2]];
```

# Line Construction

#### **Polar Line**

Line2D[*point*, *circle*] Constructs a line that is the polar line of a point with respect to a circle. If the point is on the circle then the line is tangent to the circle.

```
Line2D[P1:Point2D[{x1_,y1_}],C2:Circle2D[{h2_,k2_},r2_]] := Line2D[P1,Quadratic2D[C2]];
```

### **Radical Axis**

Line2D[*circle*, *circle*] Constructs a line that is the radical axis of two circles.

```
Line2D::concentric=
   "The circles {`1`, `2`} are concentric; no radical axis exists.";
Line2D[C1:Circle2D[{h1_,k1_},r1_],C2:Circle2D[{h2_,k2_},r2_]] :=
   If[IsConcentric2D[C1,C2],
    Message[Line2D::concentric,C1,C2];$Failed,
    Line2D[2*(h2-h1),2*(k2-k1),(h1^2-h2^2)+(k1^2-k2^2)+(r2^2-r1^2)]];
```

## **Circle Construction**

#### **Circle from Quadratic Equation**

 $Circle2D[quad] \blacksquare$  Constructs a circle from a quadratic. The quadratic must be recognizable as a circle.

```
Circle2D::noCircle=
   "The curve represented by `1` is not a circle.";
Circle2D[Q:Quadratic2D[a_,b_,c_,d_,e_,f_]] :=
   Module[{s},
    If[IsZero2D[{a-c,b},And] && Not[IsZero2D[{a,c},Or]],
        s=(d^2+e^2-4*a*f)/a^2;
    If[IsZeroOrNegative2D[s],
        Message[Circle2D::noCircle,Q];$Failed,
        Circle2D[{-d/(2a),-e/(2a)},Sqrt[s]/2]],
   Message[Circle2D::noCircle,Q];$Failed] ];
```

### **Circle from Center Point and Radius**

Circle2D[*point*, r]  $\blacksquare$  Constructs a circle from a center point and radius.

```
Circle2D::radius=
   "The radius argument, `1`, is invalid; the radius must be positive.";
Circle2D[Point2D[{h_,k_}],r_?IsScalar2D] :=
   If[IsZeroOrNegative2D[r],
        Message[Circle2D::radius,r];$Failed,
        Circle2D[{h,k},r]];
```

#### Circle from Center Point and Point on Circle

Circle2D[*point*, *point*] Constructs a circle from a center point and a point on the circle.

```
Circle2D::coincident=
    "The points {`1`, `2`} are coincident; no valid circle exists.";
Circle2D[P1:Point2D[{x1_,y1_}],P2:Point2D[{x2_,y2_}]] :=
    If[IsCoincident2D[P1,P2],
        Message[Circle2D::coincident,P1,P2];$Failed,
        Circle2D[{x1,y1},Sqrt[(x1-x2)^2+(y1-y2)^2]]];
```

#### **Circle from Center and Tangent Line**

**Circle2D**[*point*, *line*] Constructs a circle from a center point and a tangent line.

```
Circle2D::on=
    "'1' is on '2'; no valid circle exists.";
Circle2D[P1:Point2D[{x1_,y1_}],L2:Line2D[A2_,B2_,C2_]] :=
    If[IsOn2D[P1,L2],
        Message[Circle2D::on,P1,L2];$Failed,
        Circle2D[{x1,y1},Sqrt[(A2*x1+B2*y1+C2)^2/(A2^2+B2^2)]]];
```

## **Circle Through Three Points**

Circle2D[*point*, *point*, *point*] Constructs a circle through three points.

```
Circle2D::collinear=
  "The points {`1`, `2`, `3`} are collinear; no valid circle exists.";
Circle2D[P1:Point2D[{x1_,y1_}],P2:Point2D[{x2_,y2_}],
        P3:Point2D[{x3_,y3_}]] :=
        If[IsCollinear2D[P1,P2,P3],
        Message[Circle2D::collinear,P1,P2,P3];$Failed,
        Circle2D[Quadratic2D[P1,P2,P3]] ];
```

## **Epilogue**

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DCircle2D`" *)
```

# D2DConic2D

The package D2DConic2D provides functions for constructing various points, lines and line segments associated with conic curves.

# Initialization

```
BeginPackage["D2DConic2D`", {"D2DCircle2D`", "D2DEllipse2D`",
"D2DExpressions2D'", "D2DHyperbola2D'", "D2DLine2D'", "D2DParabola2D'",
"D2DPoint2D'", "D2DSegment2D'", "D2DTransform2D'"}];
D2DConic2D::usage=
  "D2DConic2D is a package for constructing geometry associated with conic
curves.";
Asymptotes2D::usage=
   "Asymptotes2D[hyperbola] constructs a list containing the two asymptote
lines of a hyperbola.";
Directrices2D::usage=
   "Directrices2D[conic] constructs a list containing the directrix line(s)
of a conic curve (one for a parabola, two for ellipses and hyperbolas).";
Eccentricity2D::usage=
   "Eccentricity2D[conic] computes the eccentricity of a conic curve
(parabola, ellipse or hyperbola).";
FocalChords2D::usage=
   "FocalChords2D[conic] constructs a list containing the focal chords
(line segments) of a conic curve (one for a parabola, two for ellipses and
hyperbolas).";
Foci2D::usage=
  "Foci2D[conic] constructs a list containing the focus point(s) of a
conic curve (one for a parabola, two for ellipses and hyperbolas).";
Vertices2D::usage=
   "Vertices2D[conic] contructs a list containing the vertex point(s) of a
conic curve (one for a parabola, two for ellipses and hyperbolas).";
Begin["'Private'"];
```

### **Scalars**

#### Eccentricity

**Eccentricity2D**[*ellipse*] ■ Computes the eccentricity of an ellipse.

Eccentricity2D[Ellipse2D[{h\_,k\_},a\_,b\_,theta\_]] := Sqrt[a^2-b^2]/a;

Eccentricity2D[*hyperbola*] ■ Computes the eccentricity of a hyperbola.

 $\label{eq:ccentricity2D[Hyperbola2D[{h_,k_},a_,b_,theta_]] := Sqrt[a^2+b^2]/a;$ 

Eccentricity2D[parabola]  $\blacksquare$  Computes the eccentricity of a parabola (e = 1).

 $\label{eq:contricity2D[Parabola2D[{h_,k_},f_,theta_]] := 1;$ 

# **Point Construction**

#### **Focus Points**

Foci2D[*ellipse*] Constructs a list containing the two focus points of an ellipse.

Foci2D[hyperbola] Constructs a list containing the two focus points of a hyperbola.

```
Foci2D[H1:Hyperbola2D[{h_,k_},a_,b_,theta_]] :=
Module[{e=Eccentricity2D[H1]},
        {Point2D[Rotate2D[{h+a*e,k},theta,{h,k}]],
        Point2D[Rotate2D[{h-a*e,k},theta,{h,k}]]};
```

**Foci2D**[*parabola*] Constructs a list containing the single focus point of a parabola.

```
Foci2D[Parabola2D[{h_,k_},f_,theta_]] :=
    {Point2D[Rotate2D[{h+f,k},theta,{h,k}]]};
```

#### **Vertex Points**

**Vertices2D**[*ellipse*] ■ Constructs a list containing the two vertex points of an ellipse.

```
Vertices2D[Ellipse2D[{h_,k_},a_,b_,theta_]] :=
  {Point2D[Rotate2D[{h+a,k},theta,{h,k}]],
    Point2D[Rotate2D[{h-a,k},theta,{h,k}]];
```

**Vertices2D**[*hyperbola*] Constructs a list containing the two vertex points of a hyperbola.

```
Vertices2D[Hyperbola2D[{h_,k_},a_,b_,theta_]] :=
  {Point2D[Rotate2D[{h+a,k},theta,{h,k}]],
    Point2D[Rotate2D[{h-a,k},theta,{h,k}]]};
```

**Vertices**2D[*parabola*] ■ Constructs a list containing the single vertex point of a parabola.

 $Vertices2D[Parabola2D[{h_,k_},f_,theta_]] := {Point2D[{h,k}]};$ 

# Line Construction

#### Asymptote Lines

Asymptotes2D[hyperbola] Constructs a list containing the two asymptote lines of a hyperbola.

```
Asymptotes2D[Hyperbola2D[{h_,k_},a_,b_,theta_]] :=
{Rotate2D[Line2D[b, a,-a*k-b*h],theta,{h,k}],
Rotate2D[Line2D[b,-a, a*k-b*h],theta,{h,k}]};
```

#### **Directrix Lines**

**Directrices2D**[*ellipse*] ■ Constructs a list containing the two directrix lines of an ellipse.

```
Directrices2D::circular=
   "The ellipse `1` is circular; it has no (finite) directrix lines.";
Directrices2D[E1:Ellipse2D[{h_,k_},a_,b_,theta_]] :=
   Module[{e=Eccentricity2D[E1]},
    If[IsZero2D[e],
      Message[Directrices2D::circular,E1];{},
      {Rotate2D[Line2D[1,0,-(h+a/e)],theta,{h,k}],
      Rotate2D[Line2D[1,0,-(h-a/e)],theta,{h,k}]}] ];
```

Directrices2D[*hyperbola*] ■ Constructs a list containing the two directrix lines of a hyperbola.

```
 \begin{array}{l} \label{eq:definition} Directrices2D[H1:Hyperbola2D[\{h_,k_\},a_,b_,theta_]] &:= \\ Module[\{e=Eccentricity2D[H1]\}, \\ & \{Rotate2D[Line2D[1,0,-(h+a/e)],theta,\{h,k\}], \\ & Rotate2D[Line2D[1,0,-(h-a/e)],theta,\{h,k\}]\} \ ]; \end{array}
```

Directrices2D[*parabola*] Constructs a list containing the single directrix line of a parabola.

```
\label{eq:discrete} \begin{split} & \text{Directrices2D[Parabola2D[}\{h_,k_\},f_,theta_]] \ := \\ & \{\text{Rotate2D[Line2D[}1,0,-h+f],theta,\{h,k\}]\}; \end{split}
```

## Line Segment Construction

### **Focal Chords**

**FocalChords2D**[*ellipse*] ■ Constructs a list containing two line segments that are the focal chords of an ellipse.

```
FocalChords2D[E1:Ellipse2D[{h_,k_},a_,b_,theta_]] :=
Module[{e,l1,l2},
    e=Eccentricity2D[E1];
    l1=Segment2D[{h+a*e,k+b^2/a},{h+a*e,k-b^2/a}];
    l2=Segment2D[{h-a*e,k+b^2/a},{h-a*e,k-b^2/a}];
    Map[Rotate2D[#,theta,{h,k}]&,{l1,l2}] ];
```

FocalChords2D[hyperbola]  $\blacksquare$  Constructs a list containing two line segments that are the focal chords of a hyperbola.

```
FocalChords2D[H1:Hyperbola2D[{h_,k_},a_,b_,theta_]] :=
Module[{e,11,12},
    e=Eccentricity2D[H1];
    l1=Segment2D[{h+a*e,k+b^2/a},{h+a*e,k-b^2/a}];
    l2=Segment2D[{h-a*e,k+b^2/a},{h-a*e,k-b^2/a}];
    Map[Rotate2D[#,theta,{h,k}]&,{l1,12}] ];
```

 $FocalChords2D[parabola] \blacksquare$  Constructs a list containing one line segment that is the single focal chord of the parabola.

```
\label{eq:solution} \begin{split} & FocalChords2D[Parabola2D[ \{h_,k_\},f_,theta_] ] := \\ & \{Rotate2D[Segment2D[ \{h+f,k+2*f\}, \{h+f,k-2*f\}],theta, \{h,k\}] \}; \end{split}
```

# **Epilogue**

```
End[ ]; (* end of "'Private" *)
EndPackage[ ]; (* end of "D2DConic2D'" *)
```
# D2DConicArc2D

The package D2DConicArc2D implements the ConicArc2D object.

## Initialization

```
BeginPackage["D2DConicArc2D`", {"D2DCircle2D`", "D2DEllipse2D`",
"D2DEquations2D`", "D2DExpressions2D`", "D2DGeometry2D`",
"D2DHyperbola2D`", "D2DIntersect2D`", "D2DLine2D`", "D2DLoci2D`",
"D2DMaster2D`", "D2DNumbers2D`", "D2DParabola2D`", "D2DPoint2D`",
"D2DQuadratic2D`", "D2DSketch2D`", "D2DTransform2D`"}];
D2DConicArc2D::usage=
   "D2DConicArc2D is a package providing the conic arc object.";
Apex2D::usage=
   "Apex2D is a keyword used in Point2D[cnarc,Apex2D] to construct the apex
control point of a conic arc.";
ConicArc2D::usage=
   "ConicArc2D[{x0,y0},{xA,yA},{x1,y1},p] is the standard form of conic arc
with start point (x0,y0), end point (x1,y1), apex point (xA,yA) and
projective discriminant 'p'.";
Rho2D::usage=
    "Rho2D[cnarc] returns the rho value of a conic arc; 0<rho<1/2 is an
ellipse; rho=1/2 is a parabola; 1/2<rho<1 is a hyperbola.";
Begin["'Private'"];
```

## Description

#### Representation

ConicArc2D[ $\{x_0, y_0\}, \{x_A, y_A\}, \{x_1, y_1\}, \rho$ ] Standard representation of a conic arc in *Descarta2D*. The first and third arguments are the coordinates of the start and end points of the conic arc, respectively. The second argument is the coordinates of the apex point of the conic arc (the apex point is the intersection of the start/end point tangents). The fourth argument is a scalar representing the  $\rho$  value of the conic arc ( $0 < \rho < 1/2$ , ellipse;  $\rho = 1/2$ , parabola;  $1/2 < \rho < 1$ , hyperbola).

#### Equation

 $Quadratic2D[cnarc] \blacksquare$  Constructs a quadratic representing the equation of the curve associated with a conic arc.

```
Quadratic2D[ConicArc2D[{x0_,y0_}, {xA_,yA_}, {x1_,y1_},p_]] :=
Module[{eqn,a,b,k,x,y},
eqn=a*b==k*(1-a-b)^2 /.
{k->(1-p)^2/(4*p^2),
a->((y-yA)*(x1-xA)-(x-xA)*(y1-yA))/
((y0-yA)*(x1-xA)-(x0-xA)*(y1-yA)),
b->((y-yA)*(x0-xA)-(x-xA)*(y0-yA))/
((y1-yA)*(x0-xA)-(x1-xA)*(y0-yA))};
Quadratic2D[eqn, {x,y}] ];
```

#### **Evaluation**

ConicArc2D[ $\{x_0, y_0\}$ ,  $\{x_A, y_A\}$ ,  $\{x_1, y_1\}$ ,  $\rho$ ][t]  $\blacksquare$  Evaluates a conic arc at a parameter, t, and returns a list of coordinates  $\{x, y\}$ . Parameter values in the range  $0 \le t \le 1$  cover the complete span of the conic arc.

```
ConicArc2D[p0:{x0_,y0_},pA:{xA_,yA_},p1:{x1_,y1_},p_][t_?IsScalar2D] :=
Module[{b0,b1,b2},
b0=(1-t)^2; b1=2*t*(1-t); b2=t^2;
(b0*(1-p)*p0+b1*p*pA+b2*(1-p)*p1)/(b0*(1-p)+b1*p+b2*(1-p))];
```

#### Graphics

Provides graphics primitives for a conic arc by extending the *Mathematica* Display command. Executed when the package is loaded.

#### Validation

ConicArc2D[ $\{x_0, y_0\}$ ,  $\{x_A, y_A\}$ ,  $\{x_1, y_1\}$ ,  $\rho$ ]  $\blacksquare$  Detects a conic arc with imaginary arguments and returns the **\$Failed** symbol. If the imaginary parts are insignificant, they are removed.

```
ConicArc2D::imaginary=
    "An invalid conic arc of the form 'ConicArc2D['1', '2', '3', '4']' has
been detected; the arguments cannot be imaginary.";
```

```
ConicArc2D[p0:{x0_,y0_},pA:{xA_,yA_},p1:{x1_,y1_},p_] :=
  (ConicArc2D @@ ChopImaginary2D[ConicArc$2D[p0,pA,p1,p]]) /;
 (FreeQ[{p0,pA,p1,p},_Pattern] && IsTinyImaginary2D[{p0,pA,p1,p}]);
ConicArc2D[p0:{x0_,y0_},pA:{xA_,yA_},p1:{x1_,y1_},p_] :=
  (Message[ConicArc2D::imaginary,p0,pA,p1,p];$Failed) /;
 (FreeQ[{p0,pA,p1,p},_Pattern] && IsComplex2D[{p0,pA,p1,p},0]);
```

ConicArc2D[ $\{x_0, y_0\}$ ,  $\{x_A, y_A\}$ ,  $\{x_1, y_1\}$ ,  $\rho$ ]  $\blacksquare$  Detects a conic arc with collinear control points and returns the **\$Failed** symbol.

```
ConicArc2D::points=
   "An invalid conic arc of the form 'ConicArc2D['1', '2', '3', '4']' has
been detected; the control points cannot be collinear.";
ConicArc2D[p0:{x0_,y0_},pA:{xA_,yA_},p1:{x1_,y1_},p_] :=
   (Message[ConicArc2D::points,p0,pA,p1,p];$Failed) /;
(FreeQ[{p0,pA,p1,p},_Pattern] &&
   IsCollinear2D[Point2D[p0],Point2D[pA],Point2D[p1]]);
```

ConicArc2D[ $\{x_0, y_0\}$ ,  $\{x_A, y_A\}$ ,  $\{x_1, y_1\}$ ,  $\rho$ ]  $\blacksquare$  Detects a conic arc with an invalid  $\rho$  value and returns the **\$Failed** symbol.

```
ConicArc2D::rho=
   "An invalid conic arc of the form 'ConicArc2D['1', '2', '3', '4']' has
been detected; the value of rho must be in the range 0<rho<1.";
ConicArc2D[p0:{x0_,y0_},pA:{xA_,yA_},p1:{x1_,y1_},p_] :=
   (Message[ConicArc2D::rho,p0,pA,p1,p];$Failed) /;
(FreeQ[{p0,pA,p1,p},_Pattern] &&
   (IsZeroOrNegative2D[p,0] || IsZeroOrNegative2D[1-p,0]));</pre>
```

**IsValid2D**[*cnarc*] ■ Verifies that a conic arc is syntactically valid.

## Scalars

Rho

**Rho2D**[*cnarc*]  $\blacksquare$  Returns the  $\rho$  value of a conic arc.

```
Rho2D[ConicArc2D[{x0_,y0_}, {xA_,yA_}, {x1_,y1_},p_]] := p;
```

## Transformations

#### Reflect

**Reflect2D**[*cnarc*, *line*] ■ Reflects a conic arc in a line.

```
Reflect2D[ConicArc2D[{x0_,y0_}, {xA_,yA_}, {x1_,y1_},p_],
L:Line2D[A2_,B2_,C2_]] :=
ConicArc2D[Reflect2D[{x0,y0},L],
Reflect2D[{xA,yA},L],
Reflect2D[{x1,y1},L],p];
```

#### Rotate

Rotate2D[*cnarc*,  $\theta$ , *coords*]  $\blacksquare$  Rotates a conic arc by an angle  $\theta$  about a position specified by a coordinate list. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

#### Scale

 $Scale2D[cnarc, s, coords] \blacksquare$  Scales a conic arc by a scale factor, s, from a position given by coordinates. If the position is omitted, it defaults to the origin (see D2DTransform2D.nb).

#### Translate

Translate2D[cnarc, {u, v}]  $\blacksquare$  Translates a conic arc delta distance.

## Construction

#### **Apex Point**

**Point2D**[*cnarc*, Apex2D] ■ Constructs the apex control point of a conic arc.

#### **Center Point**

**Point2D**[*cnarc*] Constructs the center point of the central conic underlying a conic arc.

```
Point2D::notCentral1=
    "The conic underlying `1` is not a central conic; it has no center
point.";
Point2D[C1:ConicArc2D[p0:{x0_,y0_},pA:{xA_,yA_},p1:{x1_,y1_},p_]] :=
    If[IsZero2D[p-1/2],
        Message[Point2D::notCentral1,C1];$Failed,
        Point2D[(-p^2*pA+(p-1)^2*(p0+p1)/2)/(1-2*p)] ];
```

## **Conic from Conic Arc**

Loci2D[*cnarc*] Constructs a list containing the conic curve associated with a conic arc.

```
Loci2D[C1:ConicArc2D[{x0_,y0_}, {xA_,yA_}, {x1_,y1_},p_]] :=
Loci2D[Quadratic2D[C1]];
```

#### **Conic Arc from Conic**

**ConicArc2D**[*line*, *curve*] ■ Constructs a conic arc from a portion of a conic curve defined by a chordal line.

```
ConicArc2D::noChord=
    "No chord exists between '1' and '2'; a conic arc cannot be
constructed.";
ConicArc2D::center=
    "The chord defined by '1' passes through the center of '2'; a conic arc
cannot be constructed.";
```

The private function FindRho\$2D[curve, point, {point, point}] computes  $\rho$  for a conic arc from the apex point and start/end points.

```
FindRho$2D[Circle2D[{h_,k_},r_] |
Ellipse2D[{h_,k_},a_,b_,theta_] |
Hyperbola2D[{h_,k_},a_,b_,theta_],
Point2D[{xA_,yA_}],
{Point2D[{x0_,y0_}],Point2D[{x1_,y1_}]}] :=
Module[{xM,yM},
{xM,yM}={x0+x1,y0+y1}/2;
If[IsZero2D[h-xM],
1/(1+Sqrt[k-yA]/Sqrt[k-yM]),
1/(1+Sqrt[h-xA]/Sqrt[h-xM])] //Simplify ];
```

For central conics (circles, ellipses and hyperbolas) there is a restriction that the center point cannot be on the line defining the chord of the conic arc.

Non-central conics (parabolas) have no restrictions on the position of the line defining the chord of the conic arc.

```
ConicArc2D[L1:Line2D[a1_,b1_,c1_],C2:Parabola2D[{h_,k_},f_,theta_]] :=
CnArc$2D[L1,C2,Points2D[L1,C2]];
```

Both end points of the chord of the conic arc must be on the same branch of a hyperbola.

The private function CnArc\$2D[*line*, *curve*, {*point*, *point*}] completes the computation of the conic arc.

No conic arc exists if the intersection of the line and the curve does not result in two points.

```
CnArc$2D[L1:Line2D[a1_,b1_,c1_],C2_,pts_] :=
(Message[ConicArc2D::noChord,L1,C2];$Failed)
```

## Epilogue

```
End[ ]; (* end of "'Private" *)
EndPackage[ ]; (* end of "D2DConicArc2D'" *)
```

# D2DEllipse2D

The package D2DEllipse2D implements the Ellipse2D object.

## Initialization

```
BeginPackage["D2DEllipse2D`", {"D2DExpressions2D`", "D2DGeometry2D`",
"D2DLine2D'", "D2DMaster2D'", "D2DNumbers2D'", "D2DPoint2D'",
"D2DQuadratic2D'", "D2DSegment2D'", "D2DSketch2D'", "D2DTransform2D'"}];
D2DEllipse2D::usage=
   "D2DEllipse2D is a package that implements the Ellipse2D object.";
Ellipse2D::usage=
   "Ellipse2D[{h,k},a,b,theta] is the standard form of an ellipse centered
at (h,k), semi-major axis length 'a', semi-minor axis length 'b' and
rotation angle 'theta'.";
SemiMajorAxis2D::usage=
   "SemiMajorAxis2D[ellipse] returns the length of the semi-major axis of
an ellipse.";
SemiMinorAxis2D::usage=
   "SemiMinorAxis2D[ellipse] returns the length of the semi-minor axis of
an ellipse.";
Begin["'Private'"];
```

## Description

#### Representation

Ellipse2D[ $\{h, k\}$ ,  $a, b, \theta$ ]  $\blacksquare$  Standard representation of an ellipse in *Descarta2D*. The first argument is a list of coordinates representing the center of the ellipse. The second and third arguments are (positive) scalars representing the lengths of the semi-major and semi-minor axes of the ellipse. The fourth argument is the counter-clockwise rotation angle (in radians) of the ellipse about the center point.

#### Equation

Quadratic2D[*ellipse*] ■ Constructs a quadratic representing the equation of an ellipse.

```
Quadratic2D[Ellipse2D[{h_,k_},a_,b_,theta_]] :=
Rotate2D[
Quadratic2D[b^2,0,a^2,-2*b^2*h,-2*a^2*k,-a^2*b^2+b^2*h^2+a^2*k^2],
theta,
{h,k}];
```

#### **Evaluation**

Ellipse2D[{h, k},  $a, b, \theta$ ][ $\theta_1$ ]  $\blacksquare$  Evaluates a position on an ellipse at parameter  $\theta_1$  and returns a list of coordinates {x, y}. Parameters in the range  $0 \le \theta_1 < 2\pi$  cover the entire ellipse.

```
Ellipse2D[{h,k_},a_,b_,theta_][t_?IsScalar2D] :=
Rotate2D[{h+a*Cos[t],k+b*Sin[t]},theta,{h,k}];
```

### Graphics

Provides the graphics primitives for an ellipse by extending the *Mathematica* Display command. Executed when the package is loaded.

```
SetDisplay2D[
Ellipse2D[{h_,k_},a_,b_,t_][{t1_?IsScalar2D,t2_?IsScalar2D}],
MakePrimitives2D[Ellipse2D[{h,k},a,b,t],
PrimaryAngleRange2D[{t1,t2}]] ];
SetDisplay2D[
Ellipse2D[{h_,k_},a_,b_,t_],
MakePrimitives2D[Ellipse2D[{h,k},a,b,t],{0,2Pi}] ];
```

#### Validation

Ellipse2D[ $\{h, k\}$ ,  $a, b, \theta$ ]  $\blacksquare$  Detects an ellipse with imaginary arguments and returns the **\$Failed** symbol. If the imaginary parts are insignificant, they are removed.

```
Ellipse2D::imaginary=
   "An invalid ellipse of the form 'Ellipse2D['1', '2', '3', '4']' has been
detected; the arguments of an ellipse cannot involve imaginary numbers.";
Ellipse2D[{h_,k_},a_,b_,theta_] :=
   (Ellipse2D @@ ChopImaginary2D[Ellipse$2D[{h,k},a,b,theta]]) /;
(FreeQ[{h,k,a,b,theta},_Pattern] && IsTinyImaginary2D[{h,k,a,b,theta}]);
Ellipse2D[{h_,k_},a_,b_,theta_] :=
   (Message[Ellipse2D::imaginary,{h,k},a,b,theta];$Failed) /;
(FreeQ[{h,k,a,b,theta},_Pattern] && IsComplex2D[{h,k,a,b,theta},0]);
```

Ellipse2D[ $\{h, k\}$ ,  $a, b, \theta$ ]  $\blacksquare$  Detects an ellipse with invalid arguments and returns the **\$Failed** symbol.

```
Ellipse2D::invalid=
   "An invalid ellipse of the form 'Ellipse2D['1', '2', '3', '4']' has been
detected; the length of both the semi-major and semi-minor axes must be
positive.";
Ellipse2D[{h_,k_},a_,b_,theta_] :=
   (Message[Ellipse2D::invalid,{h,k},a,b,theta];$Failed) /;
```

```
(FreeQ[{h,k,a,b,theta},_Pattern] && IsZeroOrNegative2D[{a,b},0]);
```

Ellipse2D[{h, k},  $a, b, \theta$ ]  $\blacksquare$  Detects a y-axis ellipse and rotates it  $\pi/2$  radians.

```
Ellipse2D[{h,k_},a_,b_,theta_] :=
    Ellipse2D[{h,k},b,a,theta+Pi/2] /;
(FreeQ[{h,k,a,b,theta},_Pattern] && IsNegative2D[a-b,0]);
```

Ellipse2D[{h, k},  $a, b, \theta$ ]  $\blacksquare$  Normalizes the rotation angle on all ellipses to the range  $0 \le \theta < \pi$ .

```
Ellipse2D[{h,k],a_,b_,theta_] :=
Ellipse2D[{h,k},a,b,PrimaryAngle2D[theta,Pi]] /;
(FreeQ[{h,k,a,b,theta},_Pattern] && (theta=!=PrimaryAngle2D[theta,Pi]));
```

**IsValid2D**[*ellipse*] ■ Verifies that an ellipse is syntactically valid.

## Scalars

#### Angle of Rotation

**Angle2D**[*ellipse*] ■ Returns the rotation angle of an ellipse.

Angle2D[Ellipse2D[{h\_,k\_},a\_,b\_,theta\_]] := theta;

#### Semi-major Axis Length

SemiMajorAxis2D[*ellipse*] ■ Returns the length of the semi-major axis of an ellipse.

```
SemiMajorAxis2D[Ellipse2D[{h_,k_},a_,b_,theta_]] := a;
```

#### Semi-minor Axis Length

**SemiMinorAxis2D**[*ellipse*] Returns the length of the semi-minor axis of an ellipse.

```
\texttt{SemiMinorAxis2D[Ellipse2D[\{h_,k_\},a_,b_,theta_]]} \mathrel{:=} b;
```

## Transformations

#### Reflect

**Reflect2D**[*ellipse*, *line*] ■ Reflects an ellipse in a line.

```
Reflect2D[Ellipse2D[{h_,k_},a_,b_,theta_],L:Line2D[p_,q_,r_]] :=
Ellipse2D[Reflect2D[{h,k},L],a,b,ReflectAngle2D[theta,L]];
```

#### Rotate

Rotate2D[*ellipse*,  $\theta$ , *coords*]  $\blacksquare$  Rotates an ellipse by an angle  $\theta$  about a position specified by a coordinate list. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

#### Scale

 $Scale2D[ellipse, s, coords] \blacksquare$  Scales an ellipse by a scale factor, s, from a position given by coordinates. If the position is omitted, it defaults to the origin (see D2DTransform2D.nb).

#### Translate

Translate2D[*ellipse*,  $\{u, v\}$ ]  $\blacksquare$  Translates an ellipse delta distance.

## **Point Construction**

#### **Center Point**

**Point2D**[*ellipse*] ■ Constructs the center point of an ellipse.

```
\texttt{Point2D[Ellipse2D[\{h\_,k\_\},a\_,b\_,theta\_]]} \mathrel{:=} \texttt{Point2D[\{h,k\}];}
```

#### **Pole Point**

Point2D[line, ellipse]  $\blacksquare$  Constructs the pole point of a line with respect to an ellipse. If the line is tangent to the ellipse then the point is the point of tangency.

```
Point2D[L1:Line2D[a1_,b1_,c1_],E2:Ellipse2D[{h2_,k2_},a2_,b2_,theta2_]] :=
Point2D[L1,Quadratic2D[E2]];
```

## Line Construction

#### **Axis Line**

Line2D[*ellipse*] Constructs a line containing the major axis of an ellipse.

```
Line2D[Ellipse2D[{h_,k_},a_,b_,theta_]] :=
Rotate2D[Line2D[0,1,-k],theta,{h,k}];
```

#### **Polar Line**

Line2D[*point*, *ellipse*]  $\blacksquare$  Constructs the polar (line) of a pole (point) with respect to an ellipse. If the point is on the ellipse then the line is tangent to the ellipse at the point.

```
Line2D[P1:Point2D[{x1_,y1_}],E2:Ellipse2D[{h2_,k2_},a2_,b2_,theta2_]] := Line2D[P1,Quadratic2D[E2]];
```

## **Ellipse Construction**

#### **Ellipse from Vertices and Eccentricity**

```
Ellipse2D::invdef=
    "The defining geometry or eccentricity is invalid; the eccentricity of
an ellipse must be in the range 0<e<1, the foci and vertices cannot be
coincident, and the focus cannot lie on the directrix.";
Ellipse2D[{P1:Point2D[{x1_,y1_}],P2:Point2D[{x2_,y2_}]},e_?IsScalar2D] :=
Module[{a,b,h,k},
    If[IsZeroOrNegative2D[{e,1-e},Or] || IsCoincident2D[P1,P2],
    Message[Ellipse2D::invdef];$Failed,
    a=Distance2D[P1,P2]/2;
    b=a*Sqrt[1-e^2];
    {h,k}={(x1+x2)/2,(y1+y2)/2};
    Ellipse2D[{h,k},a,b,ArcTan[x2-x1,y2-y1]]] ];
</pre>
```

#### **Ellipse from Foci and Eccentricity**

Ellipse2D[*point*, *point*, *e*] ■ Constructs an ellipse from the foci and eccentricity.

```
Ellipse2D[P1:Point2D[{x1_,y1_}],P2:Point2D[{x2_,y2_}],e_?IsScalar2D] :=
Module[{a,b,h,k},
If[IsZeroOrNegative2D[{e,1-e},Or] || IsCoincident2D[P1,P2],
Message[Ellipse2D::invdef];$Failed,
a=Distance2D[P1,P2]/(2*e);
b=a*Sqrt[1-e^2];
{h,k}={(x1+x2)/2,(y1+y2)/2};
Ellipse2D[{h,k},a,b,ArcTan[x2-x1,y2-y1]]] ];
```

#### Ellipse from Focus, Directrix and Eccentricity

Ellipse2D[*point*, *line*, *e*] ■ Constructs an ellipse from a focus point, directrix line and eccentricity.

```
Ellipse2D[P1:Point2D[{x1_,y1_}],L2:Line2D[p_,q_,r_],e_?IsScalar2D] :=
Module[{d,s,a,b,h,k},
If[IsZeroOrNegative2D[{e,1-e},Or] || IsOn2D[P1,L2],
Message[Ellipse2D::invdef];$Failed,
d=Distance2D[P1,L2];
s=(p*x1+q*y1+r)/(p^2+q^2);
a=d*e/(1-e^2);
b=a*Sqrt[1-e^2];
{h,k}={x1,y1}+{p,q}*(a*s*e)/d;
Ellipse2D[{h,k},a,b,ArcTan[p,q]]] ];
```

## **Epilogue**

```
End[ ]; (* end of "'Private" *)
EndPackage[ ]; (* end of "D2DEllipse2D'" *)
```

# **D2DEquations2D**

The package D2DEquations2D provides functions for converting *Mathematica* equations and polynomials into *Descarta2D* lines and quadratics, and vice versa.

## Initialization

```
BeginPackage["D2DEquations2D'", {"D2DExpressions2D'", "D2DLine2D'",
"D2DQuadratic2D`"}];
D2DEquations2D::usage=
   "D2DEquations2D is a package that provides functions for converting
Mathematica polynomials and equations into lines and quadratics, and vice
versa.";
Equation2D::usage=
   "Equation2D[line, \{x,y\}] forms a linear equation in two unknowns, a*x +
b*y + c == 0; Equation2D[quad, \{x,y\}] forms a quadratic equation in two
unknowns, a*x^2 + b*x*y + c*y^2 + d*x + e*y + f == 0.";
Polynomial2D::usage=
   "Polynomial2D[line, \{x\,,y\}\,] forms a linear polynomial in two unknowns,
a*x + b*y + c; Polynomial2D[quad, {x,y}] forms a quadratic polynomial in
two unknowns, a*x^2 + b*x*y + c*y^2 + d*x + e*y + f.";
SimplifyCoefficients2D::usage=
   "SimplifyCoefficients2D[coefList] returns a list of coefficients with
common factors removed.";
```

Begin["'Private'"];

## Coefficients

#### **Simplify Coefficients**

SimplifyCoefficients2D[*coefList*] ■ Returns a list of coefficients with common factors removed.

```
SimplifyCoefficients2D[coef:{c1_?IsScalar2D,c2__?IsScalar2D}] :=
Module[{gcd,coef1},
gcd=PolynomialGCD[Sequence @@ Rationalize[coef]];
If[IsZero2D[gcd], gcd=1];
coef1=Map[Simplify[#/gcd]&,coef];
Map[If[IsZero2D[Round[#]-#],Round[#],#]&,coef1] ];
```

## Equations

#### Linear

Equation2D[*line*,  $\{x, y\}$ ]  $\blacksquare$  Forms ax + by + c == 0 from a line.

Equation2D[Line2D[a\_,b\_,c\_],{x\_?IsScalar2D,y\_?IsScalar2D}] := a\*x+b\*y+c==0;

#### Quadratic

Equation2D[quad,  $\{x, y\}$ ]  $\blacksquare$  Forms  $ax^2 + bxy + cy^2 + dx + ey + f == 0$  from a quadratic.

```
Equation2D[Quadratic2D[a_,b_,c_,d_,e_,f_],{x_?IsScalar2D,y_?IsScalar2D}] :=
    a*x^2+b*x*y+c*y^2+d*x+e*y+f==0;
```

## **Polynomials**

#### Linear

Polynomial2D[*line*,  $\{x, y\}$ ]  $\blacksquare$  Forms ax + by + c from a line.

Polynomial2D[Line2D[a\_,b\_,c\_], {x\_?IsScalar2D,y\_?IsScalar2D}] := a\*x+b\*y+c;

#### Quadratic

Polynomial2D[quad,  $\{x, y\}$ ]  $\blacksquare$  Forms  $ax^2 + bxy + cy^2 + dx + ey + f$  from a quadratic.

## Epilogue

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DEquations2D`" *)
```

# D2DExpressions2D

The package D2DExpressions2D provides functions for querying Mathematica expressions.

## Initialization

BeginPackage["D2DExpressions2D`", {"D2DMaster2D`", "D2DNumbers2D`"}];

D2DExpressions2D::usage=
 "D2DExpressions2D is a package for querying Mathematica expressions.";

#### IsApproximate2D::usage=

"IsApproximate2D[expr] returns 'True' if the expression contains approximate real numbers, or if the evaluation will eventually be approximated using the 'N' function.";

#### IsComplex2D::usage=

"IsComplex2D[expr,(tol)] returns 'True' if the expression evaluates to a complex number; IsComplex2D[List,(tol)] and IsComplex2D[List,Or,(tol)] return 'True' if any expressions in the list evaluate to complex numbers; IsComplex2D[List,And,(tol)] returns 'True' if all the expressions in the list evaluate to complex numbers; the default tolerance, if omitted, is 10^(-10).";

#### IsNegative2D::usage=

"IsNegative2D[expr,(tol)] returns 'True' if the expression is negative; otherwise, returns 'False'; IsNegative2D[List,(tol)] and IsNegative2D[List,Or,(tol)] return 'True' if any expressions in the list are negative; IsNegative2D[List,And,(tol)] returns 'True' if all the expressions in the list are negative; the default tolerance, if omitted, is 10^(-10).";

#### IsNumeric2D::usage=

"IsNumeric2D[expr,(tol)] returns 'True' if the expression consists of atoms that can be evaluated to real numbers; IsNumeric2D[expr,funcName,(tol)] provides the same function with a message; the default tolerance, if omitted, is 10^(-10).";

#### IsReal2D::usage=

"IsReal2D[expr,(tol)] returns 'True' if the expression is real-valued; otherwise, returns 'False'; the default tolerance, if omitted, is 10^(-10).";

```
IsScalar2D::usage=
   "IsScalar2D[expr] returns 'True' if the expression appears to be a
scalar (not a List, object, or complex number); otherwise, returns
'False'.";
IsScalarPair2D::usage=
   "IsScalarPair2D[{expr1,expr2}] returns 'True' if both expressions in a
list appear to be a scalars (not a Lists, objects, or complex numbers);
otherwise, returns 'False'.";
IsTinyImaginary2D::usage=
   "IsTinyImaginary2D[expr,(tol)] returns 'True' if any complex numbers in
the expression have tiny imaginary parts; the default tolerance, if
omitted, is 10^(-10).";
IsZero2D::usage=
   "IsZero2D[expr,(tol)] returns 'True' if the expression is zero;
otherwise, returns 'False'; IsZero2D[List,(tol)] and
IsZero2D[List,Or,(tol)] return 'True' if any expressions in the list are
zero; IsZero2D[List,And,(tol)] returns 'True' if all the expressions in the
list are zero; the default tolerance, if omitted, is 10^(-10).";
IsZeroOrNegative2D::usage=
   "IsZeroOrNegative2D[expr,(tol)] returns 'True' if the expression is zero
or negative; otherwise, returns 'False'; IsZeroOrNegative2D[List,(tol)] and
IsZeroOrNegative2D[List,Or,(tol)] return 'True' if any expressions in the
list are zero or negative; IsZeroOrNegative2D[List,And,(tol)] returns
'True' if all the expressions in the list are zero or negative; the default
tolerance, if omitted, is 10^(-10).";
Begin["'Private'"];
```

## Utilities

#### Chop

The built-in *Mathematica* function Chop issues an error message if the tolerance is zero. These modifications to the Chop function allow a zero tolerance to be specified. Executed when the package is loaded.

```
protected=Unprotect[Chop];
Chop[expr_,0] := expr;
Chop[expr_,0.] := expr;
Protect[Evaluate[protected]];
```

#### **Random Evaluation**

The private function RandomEvaluation\$2D substitutes random numbers for the non-numeric symbols in the expression and applies the N function to the result. This is useful for determining whether a symbolic expression represents some specific numerical value (such as zero). There is a small probability that an erroneous conclusion may be reached if an unfortunate combination of random numbers arises.

```
RandomEvaluation$2D[expr_] :=
Module[{atoms,symbols,rules},
    atoms=Level[N[expr],{-1}];
    symbols=Union[Select[atoms,(Head[#]===Symbol)&]];
    rules=Map[Rule[#,Random[Real,{0.1,0.9}]]&,symbols];
    N[expr /. rules] ];
```

#### Tolerance

The private function Tolerance\$2D[tol] returns tol if it is a valid tolerance value; otherwise, issues a warning message and returns the default tolerance value,  $10^{-10}$ . The special cases are provided to improve the performance of heavily used tolerance values.

```
D2DExpressions2D::badTol=
    "The tolerance `1` is not a valid tolerance specification; the default
tolerance, 10^(-10), will be used.";
Tolerance$2D[10^(-10)] := 10^(-10);
Tolerance$2D[0] := 0;
Tolerance$2D[tol_] :=
    If[TrueQ[N[tol]>=0],
        tol,
        Message[D2DExpressions2D::badTol,tol];10^(-10)];
```

## Number Queries

#### **Approximate Query**

 $\texttt{IsApproximate2D[expr]} \blacksquare \text{Returns True if any of the atoms in an expression are approximate real numbers or if the N function will eventually be applied to the expression; otherwise, returns False.$ 

```
IsApproximate2D[expr_] :=
    (Not[FreeQ[expr,_Real]] || Stack[N[___]]=!={});
```

#### **Complex Query**

 $IsComplex2D[expr, (tol)] \blacksquare$  Returns True if an expression evaluates numerically to a complex number; otherwise, returns False. The heavily used cases are provided to improve performance. The default tolerance, if omitted, is  $10^{-10}$ .

```
IsComplex2D[n_Real,tol_:(10^(-10))] := False;
IsComplex2D[n_Integer,tol_:(10^(-10))] := False;
IsComplex2D[n_Rational,tol_:(10^(-10))] := False;
```

```
IsComplex2D[sym_Symbol,tol_:(10^(-10))] := False;
IsComplex2D[n_Complex,tol_:(10^(-10))] := Abs[Im[n]]>Tolerance$2D[tol];
IsComplex2D[expr_,tol_:(10^(-10))] :=
Module[{n,tol1},
tol1=Tolerance$2D[tol];
n=Chop[N[expr],tol1];
Head[n]==Complex ] /;
(Head[expr] =!= List);
```

#### **Complex Query (List)**

**IsComplex2D**[*exprList*, **Or** | **And**, (tol)]  $\blacksquare$  With the default option, **Or**, returns **True** if *any* expression in the list evaluates to a complex number; with the **And** option, returns **True** if *all* the expressions in the list evaluate to complex numbers; otherwise, returns **False**. The default tolerance, if omitted, is  $10^{-10}$ .

```
IsComplex2D[expr_List,bool_:Or,tol_:(10^(-10))] :=
Module[{tol1},
        tol1=Tolerance$2D[tol];
        bool @@ Map[IsComplex2D[#,tol1]&,expr] ] /;
(bool==And || bool==Or);
```

#### **Numeric Query**

IsNumeric2D[*expr*, (tol)]  $\blacksquare$  Returns True if all the atoms in an expression can be evaluated to real numbers; otherwise, returns False. The default tolerance, if omitted, is  $10^{-10}$ .

#### Numeric Query (with Message)

**IsNumeric2D**[*expr*, *funcName*] ■ Returns **True** if all the atoms in an expression can be evaluated to real numbers; otherwise, returns **False**. Outputs a message with the function name if the result is **False**.

```
IsNumeric2D::notNumeric=
    "The '1' function requires numerical arguments; symbolic arguments are
not allowed.";
IsNumeric2D[expr_,funcName_Symbol,tol_:(10^(-10))] :=
    If[IsNumeric2D[expr,Tolerance$2D[tol]],
        True,
        Message[IsNumeric2D::notNumeric,funcName];False];
```

#### **Real Query**

IsReal2D[*expr*, (tol)]  $\blacksquare$  Returns True if the expression can be evaluated to a real number; otherwise, returns False. A complex number with an insignificant imaginary component will return True. The default tolerance, if omitted, is  $10^{-10}$ .

```
IsReal2D[expr_Real,___] := True;
IsReal2D[expr_Integer,___] := True;
IsReal2D[expr_Symbol,___] := True;
IsReal2D[expr_Rational,___] := True;
IsReal2D[expr_Complex,tol_:(10^(-10))] :=
Chop[Im[expr]//N,Tolerance$2D[tol]]===0;
IsReal2D[expr_,tol_:(10^(-10))] :=
Module[{n},
n=Chop[N[expr],Tolerance$2D[tol]];
(NumberQ[n] && (Im[n]==0))];
```

#### Scalar Query

**IsScalar2D**[n] Returns **True** if an expression appears to be a scalar quantity—that is, it cannot be recognized as Null, a list, a complex number or a *Descarta2D* object; otherwise, returns **False**. The special cases for **Real**, **Integer** and **Symbol** are provided to improve the performance of heavily used queries.

```
IsScalar2D[_Real] := True;
IsScalar2D[_Integer] := True;
IsScalar2D[_Symbol] := True;
IsScalar2D[_List] := False;
IsScalar2D[_?IsComplex2D] := False;
IsScalar2D[Null] := False;
IsScalar2D[Null] := False;
IsScalar2D[expr_] := False /;
MemberQ[ObjectNames2D[],ToString[Head[expr]]];
IsScalar2D[expr_] := False /;
Not[FreeQ[expr,_Pattern]];
IsScalar2D[expr_] := True;
```

#### Scalar Pair Query

**IsScalarPair2D**[ $\{n_1, n_2\}$ ]  $\blacksquare$  Returns **True** if a list of two expressions appears to be a scalar pair—that is, neither expression can be recognized as Null, a list, a complex number or a valid *Descarta2D* object; otherwise, returns False.

```
IsScalarPair2D[{n1_,n2_}] := IsScalar2D[n1] && IsScalar2D[n2];
IsScalarPair2D[___] := False;
```

#### **Tiny Imaginary Query**

IsTinyImaginary2D[*expr*, (tol)]  $\blacksquare$  Returns True if any atoms in an expression involve complex numbers with tiny imaginary parts; otherwise, returns False. The default tolerance, if omitted, is  $10^{-10}$ .

```
IsTinyImaginary2D[expr_,tol_:(10^(-10))] :=
Module[{tol1},
tol1=Tolerance$2D[tol];
Or @@ Map[(Head[#]===Complex && Chop[Im[#],tol1]===0)&,
Level[expr,{-1}]] ];
```

## **Sign Queries**

#### **Negative Query**

IsNegative2D[*expr*, (tol)]  $\blacksquare$  Returns True if a number is numerically negative; otherwise returns False. The default tolerance, if omitted, is  $10^{-10}$ .

```
IsNegative2D[expr_,tol_:(10^(-10))] :=
Module[{n},
    n=Chop[N[expr],Tolerance$2D[tol]];
    If[MemberQ[{Real,Integer},Head[n]], n<0, False] ] /;
(Head[expr] =!= List);
```

#### Negative Query (List)

IsNegative2D[*exprList*, Or | And, *(tol)*]  $\blacksquare$  With the default option, Or, returns True if *any* expression in the list is numerically negative; with the And option, returns True if *all* the expressions in the list are numerically negative; otherwise, returns False. The default tolerance, if omitted, is  $10^{-10}$ .

```
IsNegative2D[expr_List,bool_:Or,tol_:(10^(-10))] :=
Module[{toll},
tol1=Tolerance$2D[tol];
bool @@ Map[IsNegative2D[#,tol1]&,expr] ] /;
(bool==And || bool==Or);
```

#### **Zero Query**

IsZero2D[*expr*, (tol)]  $\blacksquare$  Returns True if an expression is numerically zero; otherwise, returns False. The heavily used cases are provided as special cases to improve performance. The default tolerance, if omitted or invalid, is  $10^{-10}$ .

```
IsZero2D[n_Real,tol_:(10^(-10))] := (Abs[n]<=Tolerance$2D[tol]);</pre>
IsZero2D[n_Integer,tol_:(10^(-10))] := (Abs[n]<=Tolerance$2D[tol]);</pre>
IsZero2D[expr_Symbol,tol_:(10^(-10))] := False;
IsZero2D[n_Complex,tol_:(10^(-10))] :=
   Module[{tol1},
      tol1=Tolerance$2D[tol];
      (Abs[Re[n]]<=tol1 && Abs[Im[n]]<=tol1)];
IsZero2D[h_[___],tol_:(10^(-10))] := False /;
MemberQ[ObjectNames2D[],ToString[h]];
IsZero2D[expr_,tol_:(10^(-10))] :=
   Module[{n,tol1},
      tol1=Tolerance$2D[tol];
      n=Chop[N[expr],tol1];
      If[MemberQ[{Real,Integer,Complex},Head[n]],
         IsZero2D[n,tol1],
         Chop[RandomEvaluation$2D[n],tol1]===0] ] /;
(Head[expr] =!= List);
```

## Zero Query (List)

IsZero2D[*exprList*, Or | And, (tol)]  $\blacksquare$  With the default option, Or, returns True if any expression in the list is numerically zero; with the And option, returns True if all the expressions in the list are numerically zero; otherwise, returns False. The default tolerance, if omitted, is  $10^{-10}$ .

```
IsZero2D[expr_List,bool_:Or,tol_:(10^(-10))] :=
Module[{tol1},
        tol1=Tolerance$2D[tol];
        bool @@ Map[IsZero2D[#,tol1]&,expr] ] /;
(bool==And || bool==Or);
```

#### Zero or Negative Query

IsZeroOrNegative2D[*expr*, (tol)]  $\blacksquare$  Returns True if the expression is numerically zero or negative; otherwise, returns False. The default tolerance, if omitted, is  $10^{-10}$ .

```
IsZeroOrNegative2D[expr_,tol_:(10^(-10))] :=
Module[{tol1},
tol1=Tolerance$2D[tol];
(IsZero2D[expr,tol1] || IsNegative2D[expr,tol1])] /;
(Head[expr] =!= List);
```

## Zero or Negative Query (List)

IsZeroOrNegative2D[*exprList*, Or | And, (tol)]  $\blacksquare$  With the default option, Or, returns True if *any* expression in the list is numerically zero or negative; with the And option, returns True if *all* the expressions in the list are numerically zero or negative; otherwise, returns False. The default tolerance, if omitted, is  $10^{-10}$ .

```
IsZeroOrNegative2D[expr_List,bool_:Or,tol_:(10^(-10))] :=
   Module[{toll},
      toll=Tolerance$2D[tol];
      bool @@ Map[IsZeroOrNegative2D[#,toll]&,expr] ] /;
(bool==And || bool==Or);
```

## **Epilogue**

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DExpressions2D`" *)
```

# D2DGeometry2D

The package D2DGeometry2D provides geometric query functions.

## Initialization

```
BeginPackage["D2DGeometry2D`", {"D2DCircle2D`", "D2DExpressions2D`",
"D2DLine2D`", "D2DMaster2D`", "D2DPoint2D`", "D2DQuadratic2D`"}];
D2DGeometry2D::usage=
   "D2DGeometry2D is a package providing various geometric queries.";
IsCoincident2D::usage=
   "IsCoincident2D[obj1,obj2] returns 'True' if the two objects are
coincident; IsCoincident2D[obj_List] returns 'True' if any pair of objects
in the list is coincident (coordinates, points, lines, circles or
quadratics).";
IsCollinear2D::usage=
   "IsCollinear2D[point,point] returns 'True' if the three points are
collinear; IsCollinear2D[pt_List] returns 'True' if any triple of points in
the list are collinear.";
IsConcentric2D::usage=
   "IsConcentric2D[circle,circle] returns 'True' if the two circles are
concentric; IsConcentric2D[cir_List] returns 'True' if any pair of circles
in the list are concentric.";
IsConcurrent2D::usage=
   "IsConcurrent2D[line,line,line] returns 'True' if the three lines are
concurrent; IsConcurrent2D[ln_List] returns 'True' if any triple of lines
in the list is concurrent.";
IsOn2D::usage=
   "IsOn2D[point,line | circle | quad] returns 'True' if the point is on
the line, circle or quadratic.";
IsParallel2D::usage=
"IsParallel2D[line,line] returns 'True' if two lines are parallel;
IsParallel2D[ln_List] returns 'True' if any pair of lines in a list is
parallel.";
```

```
IsTripleParallel2D::usage=
    "IsTripleParallel2D[line,line,line] returns 'True' if three lines are
parallel; IsTripleParallel2D[ln_List] returns 'True' if any triple of lines
in a list is parallel.";
IsPerpendicular2D::usage=
    "IsPerpendicular2D[line,line] returns 'True' if two lines are
perpendicular; IsPerpendicular2D[ln_List] returns 'True' if any pair of
lines in a list is perpendicular.";
IsTangent2D::usage=
    "IsTangent2D[line,circle] returns 'True' if a line is tangent to a
circle; IsTangent2D[circle,circle] returns 'True' if two circles are
tangent to each other; IsTangent2D[line,quad] returns 'True' if a line is
tangent to a quadratic.";
Begin["'Private'"];
```

## Utilities

#### Combinations

The private functions PairQ\$2D, Pairs\$2D, TripleQ\$2D and Triples\$2D are used to apply queries over lists of objects.

```
Pairs$2D[{a_,b_}] := {{a,b}};
Pairs$2D[{a_,b_,c__}] :=
   Flatten[{Map[{a,#}&, {b,c}], Pairs$2D[{b,c}]},1];
Pairs$2D[L_List /; Length[L]<2] := L;</pre>
Triples2D[\{a_,b_,c_\}] := \{\{a,b,c\}\};
Triples2D[\{a_,b_,c_,d_\}] :=
   Flatten[{Map[({a,#[[1]],#[[2]]})&,Pairs$2D[{b,c,d}]],
            Triples$2D[{b,c,d}]},1];
Triples$2D[L_List /; Length[L]<3] := L;</pre>
PairQ$2D[L:{a_,b_},func_] :=
   MemberQ[Map[func[#[[1]],#[[2]]]&,
               Pairs$2D[L]],
           True];
PairQ$2D[{___},func_] := False;
TripleQ\$2D[L:{a_,b_,c_},func_] :=
   MemberQ[Map[func[#[[1]],#[[2]],#[[3]]]&,
               Triples$2D[L]],
           True];
TripleQ$2D[{___},func_] := False;
```

## **Coincident Queries**

## **Two Coordinates**

**IsCoincident2D**[*coords*, *coords*] ■ Returns **True** if two coordinates are coincident; otherwise, returns **False**.

#### **Two Points**

IsCoincident2D[point, point] ■ Returns True if two points are coincident; otherwise, returns
False.

```
IsCoincident2D[Point2D[{x1_,y1_}],Point2D[{x2_,y2_}]] :=
IsZero2D[x1-x2] && IsZero2D[y1-y2];
```

#### Two Lines

IsCoincident2D[*line*, *line*] ■ Returns True if two lines are coincident; otherwise, returns False.

```
IsCoincident2D[Line2D[a1_,b1_,c1_],Line2D[a2_,b2_,c2_]] :=
IsZero2D[{Det[{{ a1, b1},{ a2, b2}}],
Det[{-c1, b1},{-c2, b2}],
Det[{{ a1,-c1},{ a2,-c2}}],And]
```

### **Two Circles**

IsCoincident2D[*circle*, *circle*] ■ Returns True if two circles are coincident; otherwise, returns False.

```
IsCoincident2D[Circle2D[{h1_,k1_},r1_],Circle2D[{h2_,k2_},r2_]] :=
IsCoincident2D[{h1,k1}, {h2,k2}] && IsZero2D[r1-r2];
```

#### **Two Quadratics**

IsCoincident2D[quad, quad] ■ Returns True if two quadratics are coincident; otherwise, returns False.

#### List of Objects

**IsCoincident2D**[*objList*] ■ Returns **True** if any pair of objects (points, lines, circles or quadratics) in a list are coincident; otherwise, returns **False**.

IsCoincident2D[obj\_List] := PairQ\$2D[obj,IsCoincident2D];

## **Collinear Queries**

#### **Three Points**

**IsCollinear2D**[*point*, *point*] ■ Returns **True** if three points are collinear; otherwise, returns **False**.

```
IsCollinear2D[Point2D[{x1_,y1_}],Point2D[{x2_,y2_}],Point2D[{x3_,y3_}]] :=
IsZero2D[Det[{{x1,y1,1},{x2,y2,1},{x3,y3,1}}];
```

## List of Points

**IsCollinear2D**[*ptsList*] ■ Returns **True** if any combination of three points in a list are collinear; otherwise, returns **False**.

```
IsCollinear2D[pts_List] := TripleQ$2D[pts,IsCollinear2D];
```

## **Concentric Queries**

#### **Two Circles**

IsConcentric2D[*circle*, *circle*] ■ Returns True if two circles are concentric; otherwise, returns False.

```
IsConcentric2D[Circle2D[{h1_,k1_},r1_],Circle2D[{h2_,k2_},r2_]] :=
IsCoincident2D[{h1,k1}, {h2,k2}];
```

## List of Circles

**IsConcentric2D**[*cirList*] ■ Returns **True** if any combination of two circles in a list are concentric; otherwise, returns **False**.

```
IsConcentric2D[cir_List] := PairQ$2D[cir,IsConcentric2D];
```

## **Concurrent Queries**

#### **Three Lines**

**IsConcurrent2D**[*line*, *line*] ■ Returns **True** if three given lines are concurrent; otherwise, returns **False**. Coincident and parallel lines are not considered to be concurrent and will return **False**.

```
IsConcurrent2D[L1:Line2D[a1_,b1_,c1_],L2:Line2D[a2_,b2_,c2_],
L3:Line2D[a3_,b3_,c3_]] :=
IsZero2D[Det[{{a1,b1,c1},{a2,b2,c2},{a3,b3,c3}}]] &&
Not[IsParallel2D[L1,L2]] && Not[IsParallel2D[L2,L3]] &&
Not[IsParallel2D[L2,L3]];
```

#### List of Lines

**IsConcurrent2D**[*lnsList*] ■ Returns **True** if any combination of three lines in a list are concurrent; otherwise, returns **False**. Coincident and parallel lines are not considered to be concurrent and will return **False**.

```
IsConcurrent2D[lns_List] := TripleQ$2D[lns,IsConcurrent2D];
```

## **On Queries**

#### **Point On Line**

**IsOn2D**[*point*, *line*] ■ Returns **True** if a point is on a line; otherwise, returns **False**.

```
IsOn2D[Point2D[{x1_,y1_}],Line2D[a2_,b2_,c2_]] := IsZero2D[a2*x1+b2*y1+c2];
```

#### **Point On Circle**

**IsOn2D**[*point*, *circle*] ■ Returns **True** if a point is on a circle; otherwise, returns **False**.

```
IsOn2D[Point2D[{x1_,y1_}],Circle2D[{h2_,k2_},r2_]] :=
IsZero2D[(x1-h2)^2+(y1-k2)^2-r2^2];
```

#### **Point On Quadratic**

**IsOn2D**[*point*, *quad*] Returns **True** if a point is on a quadratic; otherwise, returns **False**.

```
IsOn2D[Point2D[{x_,y_}],Quadratic2D[a_,b_,c_,d_,e_,f_]] :=
IsZero2D[a*x^2+b*x*y+c*y^2+d*x+e*y+f];
```

## **Parallel Queries**

#### **Two Lines**

IsParallel2D[*line*, *line*] Returns True if two lines are parallel, otherwise, returns False.

```
IsParallel2D[Line2D[a1_,b1_,c1_],Line2D[a2_,b2_,c2_]] :=
IsZero2D[a1*b2-a2*b1];
```

### List of Lines (by Pairs)

**IsParallel2D**[*lnsList*] ■ Returns **True** if any combination of two lines in a list are parallel; otherwise, returns **False**.

```
IsParallel2D[lns_List] := PairQ$2D[lns,IsParallel2D];
```

#### **Three Lines**

**IsTripleParallel2D**[*line*, *line*] ■ Returns **True** if three lines are mutually parallel; otherwise, returns **False**.

#### List of Lines (by Triples)

**IsTripleParallel2D**[*lnsList*] ■ Returns **True** if any combination of three lines in a list is parallel; otherwise, returns **False**.

```
IsTripleParallel2D[lns_List] := TripleQ$2D[lns,IsTripleParallel2D];
```

## **Perpendicular Queries**

#### **Two Lines**

**IsPerpendicular2D**[*line*, *line*] ■ Returns **True** if two lines are perpendicular; otherwise, returns **False**.

```
IsPerpendicular2D[Line2D[a1_,b1_,c1_],Line2D[a2_,b2_,c2_]] :=
IsZero2D[a1*a2+b1*b2];
```

#### List of Lines

**IsPerpendicular2D**[*lnsList*] ■ Returns **True** if any combination of two lines in a list is perpendicular; otherwise, returns **False**.

IsPerpendicular2D[lns\_List] := PairQ\$2D[lns,IsPerpendicular2D];

## **Tangent Queries**

#### Line and Circle

**IsTangent2D**[*line*, *circle*] ■ Returns **True** if a line is tangent to a circle; otherwise, returns **False**.

```
IsTangent2D[Line2D[A1_,B1_,C1_],Circle2D[{h_,k_},r_]] :=
IsZero2D[r^2*(A1^2+B1^2)-(C1+A1*h+B1*k)^2];
```

### **Two Circles**

**IsTangent2D**[*circle*, *circle*] ■ Returns **True** if two circles are tangent to each other; otherwise, returns **False**.

```
IsTangent2D[C1:Circle2D[{h1_,k1_},r1_],C2:Circle2D[{h2_,k2_},r2_]] :=
Module[{d},
    d=(h1-h2)^2+(k1-k2)^2;
    IsZero2D[{d-(r1+r2)^2,d-(r1-r2)^2},Or] &&
Not[IsCoincident2D[C1,C2]] ];
```

## Line and Quadratic

**IsTangent2D**[*line*, *quad*] ■ Returns **True** if a line is tangent to a quadratic; otherwise, returns **False**.

## **Epilogue**

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DGeometry2D`" *)
```

# D2DHyperbola2D

The package D2DHyperbola2D implements the Hyperbola2D object.

## Initialization

```
BeginPackage["D2DHyperbola2D`", {"D2DExpressions2D`", "D2DGeometry2D`",
"D2DLine2D'", "D2DMaster2D'", "D2DNumbers2D'", "D2DPoint2D'",
"D2DQuadratic2D'", "D2DSegment2D'", "D2DSketch2D'", "D2DTransform2D'"}];
D2DHyperbola2D::usage=
   "'D2DHyperbola2D' is a package providing support for hyperbolas.";
Conjugate2D::usage=
   "Conjugate2D is a keyword used to construct a conjugate hyperbola in
Hyperbola2D[hyperbola, Conjugate2D].";
Hyperbola2D::usage=
   "Hyperbola2D[{h,k},a,b,theta] is the standard form of a hyperbola,
centered at (h,k), semi-transverse axis length 'a', semi-conjugate axis 'b'
and rotation angle 'theta'.";
SemiConjugateAxis2D::usage=
   "SemiConjugateAxis2D[hyperbola] returns the length of the semi-conjugate
axis of a hyperbola.";
SemiTransverseAxis2D::usage=
   "SemiTransverseAxis2D[hyperbola] returns the length of the
semi-transverse axis of a hyperbola.";
Begin["'Private'"];
```

## Description

#### Representation

Hyperbola2D[ $\{h, k\}$ ,  $a, b, \theta$ ] Standard representation of a hyperbola in Descarta2D. The first argument is a list of coordinates representing the center of the hyperbola. The second and third arguments are (positive) scalars representing the lengths of the semi-transverse and semi-conjugate axes. The fourth argument is the counter-clockwise rotation (in radians) of the hyperbola about the center point.

#### Equation

Quadratic2D[hyperbola] ■ Constructs the quadratic representing the equation of a hyperbola.

```
Quadratic2D[Hyperbola2D[{h_,k_},a_,b_,theta_]] :=
Rotate2D[
    Quadratic2D[b^2,0,-a^2,-2*b^2*h,2*a^2*k,-a^2*b^2+b^2*h^2-a^2*k^2],
    theta,{h,k}];
```

#### **Evaluation**

Hyperbola2D[ $\{h, k\}$ ,  $a, b, \theta$ , False | True][t]  $\blacksquare$  Evaluates the primary branch of a hyperbola (when the keyword is False or omitted) or its reflection (when the keyword is True). The primary branch is the one opening about the +x-axis when the rotation angle is zero. The end points of the focal chords are at parameter values -1 and +1.

## Graphics

Provides graphics for a hyperbola by extending the *Mathematica* Display command. Executed when the package is loaded.

```
SetDisplay2D[
   \label{eq:hyperbola2D[{h_,k_},a_,b_,t_][{t1_?IsScalar2D,t2_?IsScalar2D}] /;
      t1<=t2,
   MakePrimitives2D[
      Hyperbola2D[{h,k},a,b,t,False],{t1,t2}]];
SetDisplay2D[
   \label{eq:hyperbola2D[{h_,k_},a_,b_,t_][{t1_?IsScalar2D,t2_?IsScalar2D}] \ /;
      t1>t2,
   MakePrimitives2D[
      Hyperbola2D[\{h,k\},a,b,t,True\},\{t2,t1\}];
SetDisplay2D[
   Hyperbola2D[{h_,k_},a_,b_,t_],
   Map[MakePrimitives2D[
           Hyperbola2D[\{h,k\},a,b,t,\#\},
           \texttt{CurveLimits2D[{a,0}, Hyperbola2D[{0,0}, a, b, 0]]]\&,}
        {False,True}] ];
```

#### Validation

Hyperbola2D[ $\{h, k\}$ ,  $a, b, \theta$ ]  $\blacksquare$  Detects a hyperbola with imaginary arguments and returns the **\$Failed** symbol. If the imaginary parts are insignificant, they are removed.

```
Hyperbola2D::imaginary=
    "An invalid hyperbola of the form 'Hyperbola2D['1', '2', '3', '4']' has
been detected; the arguments cannot be imaginary.";
Hyperbola2D[{h_,k_},a_,b_,theta_] :=
    (Hyperbola2D @@ ChopImaginary2D[Hyperbola$2D[{h,k},a,b,theta]]) /;
(FreeQ[{h,k,a,b,theta},_Pattern] && IsTinyImaginary2D[{h,k,a,b,theta}]);
Hyperbola2D[{h_,k_},a_,b_,theta_] :=
    (Message[Hyperbola2D::imaginary,{h,k},a,b,theta];$Failed) /;
(FreeQ[{h,k,a,b,theta},_Pattern] && IsComplex2D[{h,k,a,b,theta},0]);
```

Hyperbola2D[ $\{h, k\}$ ,  $a, b, \theta$ ]  $\blacksquare$  Detects a hyperbola with invalid arguments and returns the **\$Failed** symbol.

```
Hyperbola2D::invalid=
    "An invalid hyperbola of the form 'Hyperbola2D['1', '2', '3', '4']' was
encountered; the lengths of the semi-transverse and semi-conjugate axes
must be positive.";
Hyperbola2D[{h_,k_},a_,b_,theta_] :=
    (Message[Hyperbola2D::invalid,{h,k},a,b,theta];$Failed) /;
```

```
(FreeQ[{h,k,a,b,theta},_Pattern] && IsZeroOrNegative2D[{a,b},Or,0]);
```

Hyperbola2D[{h, k},  $a, b, \theta$ ]  $\blacksquare$  Normalizes the rotation angle on a hyperbola to the range  $0 \le \theta < \pi$ .

```
Hyperbola2D[{h_,k_},a_,b_,theta_] :=
Hyperbola2D[{h,k},a,b,PrimaryAngle2D[theta,Pi]] /;
(FreeQ[{h,k,a,b,theta},_Pattern] && (theta=!=PrimaryAngle2D[theta,Pi]));
```

**IsValid2D**[*hyperbola*] ■ Verifies that a hyperbola is syntactically valid.

```
IsValid2D[Hyperbola2D[{h_,k_},a_,b_,theta_,True]] :=
IsValid2D[Hyperbola2D[{h,k},a,b,theta]];
IsValid2D[Hyperbola2D[{h_,k_},a_,b_,theta_,False]] :=
IsValid2D[Hyperbola2D[{h,k},a,b,theta]];
IsValid2D[
Hyperbola2D[{h_?IsScalar2D,k_?IsScalar2D},
a_?IsScalar2D,b_?IsScalar2D,
theta_?IsScalar2D]] := True;
```

## Scalars

#### Angle of Rotation

Angle2D[*hyperbola*] ■ Returns the rotation angle of a hyperbola.

Angle2D[Hyperbola2D[{h\_,k\_},a\_,b\_,theta\_]] := theta;

#### Semi-transverse Axis Length

SemiTransverseAxis2D[hyperbola] ■ Returns the length of the semi-transverse axis of a hyperbola.

SemiTransverseAxis2D[Hyperbola2D[{h\_,k\_},a\_,b\_,theta\_]] := a;

#### Semi-conjugate Axis Length

SemiConjugateAxis2D[hyperbola] ■ Returns the length of the semi-conjugate axis of a hyperbola.

SemiConjugateAxis2D[Hyperbola2D[{h\_,k\_},a\_,b\_,theta\_]] := b;

## Transformations

#### Reflect

**Reflect2D**[*hyperbola*, *line*] ■ Reflects a hyperbola in a line.

```
\label{eq:response} \begin{split} & \texttt{Reflect2D[Hyperbola2D[\{h\_,k\_\},a\_,b\_,theta\_],L:Line2D[p\_,q\_,r\_]]} & := \\ & \texttt{Hyperbola2D[Reflect2D[\{h,k\},L],a,b,ReflectAngle2D[theta,L]];} \end{split}
```

#### Rotate

Rotate2D[hyperbola,  $\theta$ , coords]  $\blacksquare$  Rotates a hyperbola by an angle  $\theta$  about a position specified by a coordinate list. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

```
\label{eq:response} \begin{split} & \texttt{Rotate2D[Hyperbola2D[\{h_,k_\},a_,b_,theta_],alpha_?IsScalar2D, \\ & \{x0_?IsScalar2D,y0_?IsScalar2D\}\} := \\ & \texttt{Hyperbola2D[Rotate2D[\{h,k\},alpha,\{x0,y0\}],a,b,alpha+theta];} \end{split}
```

#### Scale

Scale2D[*hyperbola*, *s*, *coords*] ■ Scales a hyperbola from a position given by coordinates. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

#### Translate

```
\label{eq:constraint} \begin{split} & \text{Translate2D[Hyperbola2D[ {h_,k_} , a_,b_,theta_],} \\ & \left\{ u_? \text{IsScalar2D,v_? IsScalar2D} \right\} \end{bmatrix} := \\ & \text{Hyperbola2D[ {h+u,k+v },a,b,theta];} \end{split}
```

## **Point Construction**

#### Center Point of a Hyperbola

**Point2D**[*hyperbola*] ■ Constructs the center point of a hyperbola.

```
\label{eq:point2D[Hyperbola2D[{h_,k_},a_,b_,theta_]] := Point2D[{h,k}];
```

#### **Pole Point**

Point2D[line, hyperbola] Constructs the pole (point) of a polar (line) with respect to a hyperbola. If the line is tangent to the hyperbola then the point is the point of tangency.

```
Point2D[L1:Line2D[a1_,b1_,c1_],H2:Hyperbola2D[{h_,k_},a_,b_,theta_]] :=
Point2D[L1,Quadratic2D[H2]];
```

## Line Construction

#### Axis of a Hyperbola

Line2D[hyperbola] Constructs a line that contains the transverse axis of a hyperbola.

```
Line2D[Hyperbola2D[{h_,k_},a_,b_,theta_]] :=
Rotate2D[Line2D[0,1,-k],theta,{h,k}];
```

#### **Polar Line**

Line2D[*point*, *hyperbola*]  $\blacksquare$  Constructs the polar (line) of a pole (point) with respect to a hyperbola. If the point is on the hyperbola then the line is tangent to the hyperbola at the point.

```
Line2D[P1:Point2D[{x1_,y1_}],H2:Hyperbola2D[{h_,k_},a_,b_,theta_]] := Line2D[P1,Quadratic2D[H2]];
```

## Hyperbola Construction

#### Conjugate Hyperbola

Hyperbola2D[*hyperbola*, Conjugate2D] ■ Constructs the conjugate hyperbola of a given hyperbola.

```
Hyperbola2D[Hyperbola2D[{h_,k_},a_,b_,theta_],Conjugate2D] :=
Hyperbola2D[{h,k},b,a,theta+Pi/2]
```

### Hyperbola from Vertices/Eccentricity

```
Hyperbola2D::invdef=
   "The defining geometry or eccentricity is invalid; the eccentricity of a
hyperbola must be greater than 1, the foci and vertices cannot be
coincident and the focus cannot lie on the directrix.";
Hyperbola2D[{P1:Point2D[{x1_,y1_}],P2:Point2D[{x2_,y2_}]},e_?IsScalar2D] :=
   Module[{a,b,h,k},
    If[IsZeroOrNegative2D[e-1] || IsCoincident2D[P1,P2],
        Message[Hyperbola2D::invdef];$Failed,
        a=Distance2D[P1,P2]/2;
        b=a*Sqrt[e^2-1];
        {h,k}={(x1+x2)/2,(y1+y2)/2};
        Hyperbola2D[{h,k},a,b,ArcTan[x2-x1,y2-y1]]] ];
```

#### Hyperbola from Foci/Eccentricity

Hyperbola2D[*point*, *point*, *e*] ■ Constructs a hyperbola from the foci and eccentricity.

```
Hyperbola2D[P1:Point2D[{x1_,y1_}],P2:Point2D[{x2_,y2_}],e_?IsScalar2D] :=
Module[{a,b,h,k},
If[IsZeroOrNegative2D[e-1] || IsCoincident2D[P1,P2],
Message[Hyperbola2D::invdef];$Failed,
a=Distance2D[P1,P2]/(2*e);
b=a*Sqrt[e^2-1];
{h,k}={(x1+x2)/2,(y1+y2)/2};
Hyperbola2D[{h,k},a,b,ArcTan[x2-x1,y2-y1]]] ];
```
## Hyperbola from Focus/Directrix/Eccentricity

**Hyperbola2D**[*point*, *line*, *e*] ■ Constructs a hyperbola from focus point, directrix line and eccentricity.

```
Hyperbola2D[P1:Point2D[{x1_,y1_}],L2:Line2D[p_,q_,r_],e_?IsScalar2D] :=
Module[{d,s,a,b,h,k},
If[IsZeroOrNegative2D[e-1] || IsOn2D[P1,L2],
Message[Hyperbola2D::invdef];$Failed,
d=Distance2D[P1,L2];
s=(p*x1+q*y1+r)/(p^2+q^2);
a=d*e/(e^2-1);
b=a*Sqrt[e^2-1];
{h,k}={x1,y1}-{p,q}*(a*s*e)/d;
Hyperbola2D[{h,k},a,b,ArcTan[p,q]]] ];
```

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DHyperbola2D`" *)
```

# D2DIntersect2D

The package D2DIntersect2D constructs points that are the intersection points of two curves. It also computes the parameter values where a chordal line intersects a conic curve.

# Initialization

```
BeginPackage["D2DIntersect2D`",{"D2DCircle2D`", "D2DEllipse2D`",
"D2DEquations2D`", "D2DExpressions2D`", "D2DGeometry2D`",
"D2DHyperbola2D`", "D2DLine2D`", "D2DMaster2D`", "D2DNumbers2D`",
"D2DParabola2D`", "D2DPoint2D`", "D2DQuadratic2D`", "D2DSolve2D`",
"D2DTransform2D`"}];
D2DIntersect2D::usage=
    "D2DIntersect2D is a package for constructing the intersection points
between curves.";
Parameters2D::usage=
    "Parameters2D[line, conic] constructs a list containing the two
parameters where a line intersects a conic (circle, ellipse, parabola or
hyperbola).";
Points2D::usage=
    "Points2D[curve ,curve] constructs a list of intersection points of two
curves.";
Begin["`Private`"];
```

# **Intersection Points**

## Intersection Point of Two Lines

**Points2D**[*line*, *line*] ■ Constructs a list containing up to one point that is the intersection point of two lines.

```
Points2D[L1:Line2D[a1_,b1_,c1_],L2:Line2D[a2_,b2_,c2_]] :=
If[IsParallel2D[L1,L2], {}, {Point2D[L1,L2]}];
```

#### Intersection Points of a Line and a Circle

Points2D[*line*, *circle*]  $\blacksquare$  Constructs a list containing at most two points that are the intersection points of a line and a circle.

## Intersection Points of Two Circles

 $Points2D[circle, circle] \blacksquare$  Constructs a list containing at most two points that are the intersection points of two circles.

```
Points2D[Circle2D[{h1_,k1_},r1_],Circle2D[{h2_,k2_},r2_]] :=
Module[{s,d,R,x0,y0,cos,sin,z,mapList},
If[IsCoincident2D[{h1,k1},{h2,k2}],{},
s=(h1-h2)^2+(k1-k2)^2; d=Sqrt[s]; R=s+r1^2-r2^2;
{x0,y0}={R,Sqrt[z=(4*s*r1^2-R^2)]}/(2d);
{cos,sin}={h2-h1,k2-k1}/d;
mapList=Which[IsZero2D[z], {0},
IsNegative2D[z], {},
True, {-1,1}];
Map[Point2D[{h1+cos*x0+#*sin*y0,k1+sin*x0-#*cos*y0}]&,mapList]] ];
```

### Intersection Points of Two Curves

Points2D[curve, curve] ■ Constructs a list containing at most four points that are the intersection points of two curves. Valid curve forms are lines, circles, parabolas, ellipses, hyperbolas and quadratics. Coincident point solutions appear only once in the list returned. The private function Eqn\$2D returns a line or quadratic representing a curve.

```
Eqn$2D[crv_] :=
	If[Is2D[crv, {Line2D,Quadratic2D}],crv,Quadratic2D[crv]];
Points2D[crv1_,crv2_] :=
	Module[{x,y,eqns,roots},
	eqns=Map[Equation2D[#,{x,y}]&,Map[Eqn$2D,{crv1,crv2}]];
	roots=Select[Solve2D[eqns,{x,y}],Not[IsComplex2D[{x,y} /. #]]&];
	Union[Map[(Point2D[{x,y}] /. #)&,roots]] ] /;
(Is2D[crv1,
	{Circle2D,Ellipse2D,Hyperbola2D,Line2D,Parabola2D,Quadratic2D}] &&
	Is2D[crv2,
	{Circle2D,Ellipse2D,Hyperbola2D,Line2D,Parabola2D,Quadratic2D}]);
```

## **Chordal Parameter Range**

### Sort Numerically

The private function SortNumeric\$2D sorts a list of two numbers into ascending order, retaining the exact form of the numbers, and returns the sorted list. If the ascending order cannot be determined (for example, when the arguments are symbolic) then the original ordering is returned.

```
\label{eq:sortNumeric} SortNumeric \\ \\ 2D[ \\ \{n1\_,n2\_ \} ] \ := \ If[IsNegative2D[n2-n1], \\ \\ \{n2,n1\}, \\ \\ \{n1,n2\} ]; \\
```

### Circle

Parameters2D[line, circle]  $\blacksquare$  Constructs a list containing the two parameters on a circle where a line intersects the circle. The parameters will be primary angles and sorted in increasing order  $(0 \le \theta_1 < \theta_2 < 2\pi)$ .

## Ellipse

**Parameters2D**[*line*, *ellipse*]  $\blacksquare$  Returns a list of the two parameters on an ellipse where a line intersects the ellipse. The parameters will be primary angles and sorted in increasing order  $(0 \le \theta_1 < \theta_2 < 2\pi)$ .

### Hyperbola

**Parameters2D** [*line*, *hyperbola*]  $\blacksquare$  Returns a list of the two parameters on a hyperbola where a line intersects the hyperbola. The parameters are sorted in the order  $(-\infty < t_1 < t_2 < +\infty)$ .

The line must intersect the hyperbola's primary branch (in standard position) in two points for a parameter range to be returned.

```
Parameters2D[L1:Line2D[a1_,b1_,c1_],
             \texttt{H2:Hyperbola2D[\{h_,k_\},a_,b_,theta_]]} :=
   Module[{ln,pts,t1,t2,x1,x2,y1,y2},
      ln=Rotate2D[Translate2D[L1, {-h,-k}], -theta];
      pts=Points2D[ln,Hyperbola2D[{0,0},a,b,0]];
      If[Length[pts]==2,
         \{\{x1,y1\},\{x2,y2\}\}=Map[Coordinates2D,pts];
         t1=ArcSinh[y1/b]/ArcCosh[Sqrt[a^2+b^2]/a];
         t2=ArcSinh[y2/b]/ArcCosh[Sqrt[a^2+b^2]/a];
         If[IsZero2D[t1-t2],
            Message[Parameters2D::noChord,L1,H2];$Failed,
            If[IsNumeric2D[{t1,t2}] &&
               Not[IsCoincident2D[L1,Line2D[H2[t1],H2[t2]]]],
               Message[Parameters2D::noChord,L1,h2];$Failed,
               SortNumeric$2D[{t1,t2}]] ],
         Message[Parameters2D::noChord,L1,H2];$Failed] ];
```

## Parabola

**Parameters2D**[*line*, *parabola*]  $\blacksquare$  Returns a list of two parameters on a parabola where a line intersects the parabola. The parameters are sorted in increasing order ( $\infty < t_1 < t_2 < +\infty$ ).

```
Parameters2D[L1:Line2D[a1_,b1_,c1_],P2:Parabola2D[{h_,k_},f_,theta_]] :=
Module[{t,x,y,ans},
        {x,y}=P2[t];
        ans=Select[Solve[a1*x+b1*y+c1==0,t],Not[IsComplex2D[t /. #]]&];
        If[Length[ans]==2,
            SortNumeric$2D[Map[(t /. #)&,ans]],
        Message[Parameters2D::noChord,L1,P2];$Failed] ];
```

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DIntersect2D`" *)
```

# D2DLine2D

The package D2DLine2D implements the Line2D object.

# Initialization

```
BeginPackage["D2DLine2D`", {"D2DEquations2D`", "D2DExpressions2D`",
"D2DGeometry2D`", "D2DMaster2D`", "D2DNumbers2D`", "D2DPoint2D`",
"D2DQuadratic2D`", "D2DSketch2D`", "D2DTransform2D`"}];
D2DLine2D::usage=
   "D2DLine2D is a package that implements the Line2D object.";
Angle2D::usage=
   "Angle2D[line] gives the between the +x-axis and a line;
Angle2D[line,line] gives the angle measured counter-clockwise from the
first line to the second line. Angle2D[conic] gives the angle of rotation
of a conic.";
Line2D::usage=
  "Line2D[A,B,C] is the standard form of a line with the equation
Ax+By+C=0.";
Parallel2D::usage=
   "Parallel2D is the keyword required in Line2D[point, line, Parallel2D].";
Perpendicular2D::usage=
   "Perpendicular2D is the keyword required in Line2D[point, point,
Perpendicular2D]; it is also required in Line2D[point, line,
Perpendicular2D].";
Slope2D::usage=
   "Slope2D[line] gives the slope of a line. Slope[lnseg] gives the slope
of a line segment.";
Begin["'Private'"];
```

## Description

#### Representation

Line2D[A, B, C] Standard representation of a line in Descarta2D. The three arguments are the coefficients of the line in general form, Ax + By + C = 0. The normal form of a line,  $x \cos \theta + y \sin \theta - \rho = 0$ , is also by provided by using the form Line2D[ $\cos \theta$ ,  $\sin \theta$ ,  $-\rho$ ], where  $\theta$  is the angle the normal to the line makes with the +x-axis, and  $\rho$  is the distance of the line from the origin.

### Equations

Line2D[*expr*,  $\{x, y\}$ ] Constructs a line from a linear polynomial in two unknowns. For example, the polynomial ax+by+c will return Line2D[*a*, *b*, *c*]; the equation ax+by+c == 0 will also return Line2D[*a*, *b*, *c*]. The  $\{x, y\}$  arguments are assumed to be the names of the variables.

### **Evaluation**

Line2D[A, B, C] [t]  $\blacksquare$  Evaluates a parameter, t, on a line. Returns a coordinate list  $\{x, y\}$ . The point nearest the origin is at parameter t = 0. Other points are parameterized by distance along the line.

```
Line2D[a1_,b1_,c1_][t_?IsScalar2D] :=
Module[{a,b,c},
{a,b,c}={a1,b1,c1}/Sqrt[a1^2+b1^2];
-{a*c,b*c}+{b,-a}*t ];
```

### Graphics

Provides graphics for a line by extending the *Mathematica* **Display** command. Executed when the package is loaded.

```
SetDisplay2D[
Line2D[a_,b_,c_][{t1_?IsScalar2D,t2_?IsScalar2D}],
Line[{Line2D[a,b,c][t1],
Line2D[a,b,c][t2]}] ];
SetDisplay2D[
Line2D[a_,b_,c_],
Line[{Line2D[a,b,c][-AskCurveLength2D[]/2],
Line2D[a,b,c][ AskCurveLength2D[]/2]}];
```

### Validation

Line2D[A, B, C]  $\blacksquare$  Detects a line with imaginary coefficients and returns the **\$Failed** symbol. If the imaginary parts are insignificant, they are removed.

```
Line2D::imaginary=
   "An invalid line of the form 'Line2D['1', '2', '3']' has been detected;
the arguments cannot be imaginary.";
Line2D[a_,b_,c_] :=
   (Line2D @@ ChopImaginary2D[Line$2D[a,b,c]]) /;
(FreeQ[{a,b,c},_Pattern] && IsTinyImaginary2D[{a,b,c}]);
Line2D[a_,b_,c_] :=
   (Message[Line2D::imaginary,a,b,c];$Failed) /;
(FreeQ[{a,b,c},_Pattern] && IsComplex2D[{a,b,c},0]);
```

Line2D[A, B, C]  $\blacksquare$  Returns the \$Failed symbol when an invalid line is detected (the first two coefficients are zero). Also, normalizes lines with tiny coefficients to improve numerical stability.

```
Line2D::invalid=
   "An invalid line of the form 'Line2D['1', '2', '3']' was encountered; at
least one of the first two coefficients must be non-zero.";
Line2D[a_,b_,c_] :=
   (Message[Line2D::invalid,a,b,c];$Failed) /;
(FreeQ[{a,b,c},_Pattern] && IsZero2D[{a,b},And,0]);
Line2D[a_,b_,c_] :=
   (Line2D @@ ({a,b,c}/Sqrt[a^2+b^2])) /;
(FreeQ[{a,b,c},_Pattern] && IsZero2D[{a,b},And]);
```

**IsValid2D**[*line*] ■ Verifies that a line is syntactically valid.

IsValid2D[Line2D[a\_?IsScalar2D,b\_?IsScalar2D,c\_?IsScalar2D]] := True;

### Simplify and FullSimplify

Simplify [*line*] and FullSimplify [*line*]  $\blacksquare$  Extends the Mathematica commands Simplify and FullSimplify to simplify the coefficients of a line by factoring out common factors. Executed when the package is loaded.

```
protected=Unprotect[Simplify];
Simplify[expr_?(!FreeQ[#,Line2D[a_,b_,c_]]&),opts__] :=
Simplify[expr /. Line2D[a_,b_,c_] :>
(Line$2D @@ SimplifyCoefficients2D[{a,b,c}]),
opts] /. Line$2D->Line2D;
Protect[Evaluate[protected]];
protected=Unprotect[FullSimplify];
FullSimplify[expr_?(!FreeQ[#,Line2D[a_,b_,c_]]&),
opts__] :=
FullSimplify[expr /. Line2D[a_,b_,c_] :>
(Line$2D @@ SimplifyCoefficients2D[{a,b,c}]),
opts] /. Line$2D->Line2D;
Protect[Evaluate[protected]];
```

# Scalars

### Angle of a Line

Angle2D[*line*]  $\blacksquare$  Computes the angle measured counter-clockwise from the +x-axis to a line. The result is returned in radians.

```
Angle2D[Line2D[a_,b_,c_]] :=
PrimaryAngle2D[If[IsZero2D[b],Pi/2,ArcTan[-a/b]],Pi];
```

### Angle between Two Lines

Angle2D[*line*, *line*] ■ Computes the angle measured counter-clockwise from the first line to the second line. The result is returned in radians.

```
Angle2D[L1:Line2D[a1_,b1_,c1_],L2:Line2D[a2_,b2_,c2_]] :=
PrimaryAngle2D[(Angle2D[L2]-Angle2D[L1]),Pi];
```

### Distance from a Point to a Line

**Distance2D**[*point*, *line*] ■ Computes the distance between a point and a line.

```
Distance2D[Point2D[{x1_,y1_}],Line2D[a2_,b2_,c2_]] :=
Sqrt[(a2*x1+b2*y1+c2)^2/(a2^2+b2^2)];
```

## Slope of a Line

 $Slope2D[line] \square$  Computes the slope of a line.

Slope2D[Line2D[a\_,b\_,c\_]] := If[IsZero2D[b],Infinity,-a/b];

## Transformations

## Reflect

**Reflect2D**[*line*, *line*] ■ Reflects the first line in the second line.

### Rotate

Rotate2D[*line*,  $\theta$ , *coords*]  $\blacksquare$  Rotates a line by an angle  $\theta$  about a position given by coordinates. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

## Scale

Scale2D[*line*, *s*, *coords*] ■ Scales a line from a position given by coordinates. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

```
Scale2D[Line2D[a_,b_,c_],s_?IsScalar2D, {h_?IsScalar2D,k_?IsScalar2D}] :=
Line2D[a,b,a*(s-1)*h+b*(s-1)*k+c*s] /;
Not[IsZeroOrNegative2D[s]];
```

### Translate

Translate2D[*line*,  $\{u, v\}$ ]  $\blacksquare$  Translates a line delta distance.

## Line Construction

### Normalize a Line

Line2D[*line*] Constructs a line with normalized coefficients.

```
Line2D[Line2D[a_,b_,c_]] :=
If[IsZeroOrNegative2D[c],
    Apply[Line2D, {a,b,c}/Sqrt[a^2+b^2]],
    Apply[Line2D,-{a,b,c}/Sqrt[a^2+b^2]] ];
```

### Line Through a Point with a Given Slope

Line2D[*point*, m]  $\blacksquare$  Constructs a line through a point with a given slope. If the slope m is the symbol Infinity then a vertical line is returned.

```
Line2D[Point2D[{x0_,y0_}],Infinity] := Line2D[1,0,-x0];
Line2D[Point2D[{x0_,y0_}],m_?IsScalar2D] := Line2D[m,-1,-m*x0+y0];
```

## **Offset Line**

Line2D[*line*, d]  $\blacksquare$  Constructs a line offset a given distance from a given line. The offset distance may be positive or negative to produce the two possible offset lines.

```
Line2D[Line2D[a_,b_,c_],d_?IsScalar2D] :=
Line2D[a,b,c-d*Sqrt[a^2+b^2]];
```

### Line Through Two Coordinates

Line2D[coords, coords] Constructs a line through two points given as coordinates. Also, using Line2D[{a, 0}, {0, b}] provides a construction of the intercept form of a line, x/a + y/b = 1.

```
Line2D::sameCoords=
  "The coordinates '1' and '2' are coincident; no valid line can be
constructed.";
Line2D[{x1_?IsScalar2D,y1_?IsScalar2D}, {x2_?IsScalar2D,y2_?IsScalar2D}] :=
   If[IsCoincident2D[{x1,y1}, {x2,y2}],
      Message[Line2D::sameCoords, {x1,y1}, {x2,y2}];$Failed,
      Line2D[-(y2-y1), (x2-x1), (x1*y2-x2*y1)] ];
```

## Line Through Two Points

Line2D[*point*, *point*] Constructs a line through two points.

```
Line2D[Point2D[{x1_,y1_}],Point2D[{x2_,y2_}]] := Line2D[{x1,y1},{x2,y2}];
```

#### Line Equidistant from Two Points

Line2D[*point*, *point*, Perpendicular2D] ■ Constructs a line equidistant from two points (the perpendicular bisector of the line segment joining the two points).

```
Line2D[Point2D[{x1_,y1_}],Point2D[{x2_,y2_}],Perpendicular2D] :=
If[IsCoincident2D[{x1,y1},{x2,y2}],
Message[Line2D::sameCoords,{x1,y1},{x2,y2}];$Failed,
Line2D[x1-x2,y1-y2,-((x1^2+y1^2)-(x2^2+y2^2))/2]];
```

### Line Perpendicular to a Line Through a Point

Line2D[*point*, *line*, Perpendicular2D] Constructs a line perpendicular to a given line through a given point. The keyword Perpendicular2D is optional and may be omitted.

```
Line2D[P1:Point2D[{x1_,y1_}],L2:Line2D[a2_,b2_,c2_]] :=
Line2D[P1,L2,Perpendicular2D];
Line2D[Point2D[{x1_,y1_}],Line2D[a2_,b2_,c2_],Perpendicular2D] :=
Line2D[b2,-a2,-x1*b2+y1*a2];
```

### Line Parallel to a Line Through a Point

Line2D[*point*, *line*, Parallel2D] Constructs a line parallel to a given line through a given point.

```
Line2D[Point2D[{x1_,y1_}],Line2D[a2_,b2_,c2_],Parallel2D] :=
Line2D[-a2,-b2,x1*a2+y1*b2];
```

## Polar Line of a Quadratic

Line2D[*point*, *quad*] Constructs a polar (line) of a quadratic with respect to a pole (point). If the point is on the quadratic then the line is the tangent at the point.

```
Line2D::noPolar=
   "Since `1` is at the center of the conic, no polar line exists.";
Line2D[P1:Point2D[{x1_,y1_}],Q2:Quadratic2D[a_,b_,c_,d_,e_,f_]] :=
   Module[{p,q,r},
        p=2*a*x1+b*y1+d; q=b*x1+2*c*y1+e; r=d*x1+e*y1+2*f;
        If[IsZero2D[{p,q},And],
            Message[Line2D::noPolar,P1,Q2];$Failed,
        Line2D[p,q,r]] ];
```

```
End[ ]; (* end of "'Private" *)
EndPackage[ ]; (* end of "D2DLine2D'" *)
```

# D2DLoci2D

The package D2DLoci2D provides functions for constructing loci (points, lines and conics) from equations.

# Initialization

```
BeginPackage["D2DLoci2D`", {"D2DCircle2D`", "D2DEllipse2D`",
"D2DExpressions2D`", "D2DGeometry2D`", "D2DHyperbola2D`", "D2DLine2D`",
"D2DParabola2D`", "D2DPoint2D`", "D2DQuadratic2D`", "D2DTransform2D`"}];
D2DLoci2D::usage=
    "D2DLoci2D is a package that provides functions constructing loci
(points, lines and conics) from equations.";
Loci2D::usage=
    "Loci2D[quad] constructs a list of loci (point, lines or conics)
represented by a quadratic; Loci2D[point, line, eccentricity] constructs a
list of loci (point, lines or conics) given the focus point, directrix line
and eccentricity.";
Begin["'Private`"];
```

# Utilities

## Not Zero Query

The private function NotZero\$2D returns the logical negation of IsZero2D.

```
NotZero$2D[r_] := Not[IsZero2D[r]];
```

# **Conic Construction**

### **Conic from Quadratic**

Loci2D[quad] Constructs a list of conics (proper or degenerate) represented by a quadratic.

```
Loci2D::central=
   "The quadratic is a central conic, but its type cannot be determined.";
Loci2D::noLocus=
   "The quadratic has no real locus.";
```

LINEAR POLYNOMIAL, Dx + Ey + F = 0: Constructs a list containing one line represented by a quadratic whose second-degree coefficients are all zero.

```
Loci2D[Quadratic2D[a_?IsZero2D,b_?IsZero2D,c_?IsZero2D,d_,e_,f_]] :=
    {Line2D[d,e,f]} /;
(NotZero$2D[d] || NotZero$2D[e]);
```

PARALLEL LINES,  $Ax^2 + Dx + F = 0$ : Constructs a list of two vertical parallel lines.

PARALLEL LINES,  $Cx^2 + Ex + F = 0$ : Constructs a list of two horizontal parallel lines.

INTERSECTING LINES,  $Ax^2 + Cy^2 = 0$ : Constructs a list of two intersecting lines or a list containing a single point.

CIRCLE,  $Ax^2 + Cy^2 + F = 0, A = C$ : Constructs a list of one circle.

PARABOLA,  $Cy^2 + Dx + Ey + F = 0$ : Constructs a list of one parabola in standard position or rotated  $\pi$  radians.

PARABOLA,  $Ay^2 + Dx + Ey + F = 0$ : Constructs a list of one parabola rotated  $\frac{\pi}{2}$  or  $\frac{3\pi}{2}$  radians.

CENTRAL CONIC,  $Ax^2 + Cy^2 + F = 0$ : Constructs a list of one central conic (ellipse or hyperbola).

```
Loci2D[Quadratic2D[a_?NotZero$2D,b_?IsZero2D,c_?NotZero$2D,
                   d_?IsZero2D,e_?IsZero2D,f_?NotZero$2D]] :=
   Which[
      IsNegative2D[-f/a] && IsNegative2D[-f/c],
         Message[Loci2D::noLocus];{},
      IsNegative2D[-f/a] && IsNegative2D[f/c],
         {Hyperbola2D[\{0,0\},Sqrt[-f/c],Sqrt[f/a],Pi/2]},
      IsNegative2D[f/a] && IsNegative2D[-f/c],
         \{ Hyperbola2D[\{0,0\}, Sqrt[-f/a], Sqrt[f/c], 0] \}, 
      IsNegative2D[f/a] && IsNegative2D[f/c],
         If [IsNegative2D[(-f/a)-(-f/c)],
            {Ellipse2D[{0,0},Sqrt[-f/c],Sqrt[-f/a],Pi/2]},
            {Ellipse2D[{0,0},Sqrt[-f/a],Sqrt[-f/c],0]}],
      True,
         Message[Loci2D::central];{} ] /;
NotZero$2D[a-c];
```

REMOVE FIRST-DEGREE TERMS,  $Ax^2 + Cy^2 + Dx + Ey + F = 0$ : Removes the x- and yterms from a quadratic by applying a change of variables to the equation. The Translate2D function performs the inverse translation, thus returning the geometry to its original position.

```
Loci2D[Quadratic2D[a_?NotZero$2D,b_?IsZero2D,c_?NotZero$2D,d_,e_,f_]] :=
Translate2D[Loci2D[Quadratic2D[4*a^2*c,0,4*a*c^2,
0,0,-c*d^2-a*e^2+4*a*c*f]],
{-d/(2*a),-e/(2*c)}] /;
(NotZero$2D[d] || NotZero$2D[e]);
```

ELIMINATE CROSS-TERM,  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$ : Eliminates the cross-term of a quadratic by making the substitution x = kx + y and y = ky - x which is a scaling and rotation. The subsequent scaling and rotation accomplishes the inverse transformation, thus returning the geometry to its original position. If k is sufficiently close to zero then the substitution is not needed.

```
Loci2D[Quadratic2D[a_,b_?NotZero$2D,c_,d_,e_,f_]] :=
Module[{k=Sqrt[((c-a)/b)^2+1]+(c-a)/b,Q1},
If[IsZero2D[k],
Loci2D[Quadratic2D[a,0,c,d,e,f]],
Q1=Quadratic2D[a*k^2-b*k+c,0,c*k^2+b*k+a,d*k-e,e*k+d,f];
Rotate2D[Scale2D[Loci2D[Q1],Sqrt[1+k^2]],
-ArcTan[1/k]]] ];
```

#### Conic from Focus, Directrix and Eccentricity

Loci2D[*point*, *line*, *e*] Constructs a list of conics (proper or degenerate) from a focus point, directrix line and eccentricity.

```
Loci2D::eccentricity=
   "The eccentricity, `1`, is invalid; the eccentricity must be positive.";
Loci2D[P1:Point2D[{x1_,y1_}],L2:Line2D[a2_,b2_,c2_],e_?IsScalar2D] :=
   If[IsZeroOrNegative2D[e],
      Message[Loci2D::eccentricity,e];$Failed,
      Loci2D[Quadratic2D[P1,L2,e]] ];
```

### **Conic Vertex Equation**

Loci2D[*point*, *fcLen*, *e*,  $\theta$ ]  $\blacksquare$  Constructs a conic (circle, ellipse, hyperbola or parabola) from the vertex point, focal chord length, eccentricity and rotation angle. If the rotation angle is omitted, it defaults to zero.

```
Loci2D[P1:Point2D[{x1_,y1_}],fcLen_?IsScalar2D,e_?IsScalar2D] :=
Loci2D[P1,fcLen,e,0];
Loci2D[P1:Point2D[{x1_,y1_}],fcLen_?IsScalar2D,e_?IsScalar2D,
theta_?IsScalar2D] :=
Module[{Q},
Q=Quadratic2D[P1,fcLen,e,theta];
If[Q===$Failed,Q,Loci2D[Q]] ];
```

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DLoci2D`" *)
```

# D2DMaster2D

The package D2DMaster2D is the master package for *Descarta2D*. It establishes the names owned by all the other *Descarta2D* packages (so they will load automatically when referenced), and it provides the basic environment of queries supporting the *Descarta2D* objects.

# **Descarta2D** Initialization

Load the Descarta<sub>2D</sub> package stubs.

```
If[Names["D2DMaster2D"]=={"D2DMaster2D"},
   D2DMaster$2D::loaded=
      "The package 'D2DMaster2D' has already been loaded.";
  Message[D2DMaster$2D::loaded]];
D2DMaster$2D::noPath=
   "The path to 'D2DMaster2D.m' cannot be found; unable to initialize
Descarta2D.";
D2DMaster$2D::tooManyPaths=
   "More than one path to 'D2DMaster2D.m' was found; using `1`.";
D2D$paths=Map[(#<>"\\Descarta2D")&,$Path];
D2D$dir=Select[D2D$paths,
               (Length[FileNames["D2DMaster2D.m", {#}]]>0)&];
If[Length[D2D$dir]===0,
  Message[D2DMaster$2D::noPath],
  If[Length[D2D$dir]>1,
      Message[D2DMaster$2D::tooManyPaths,First[D2D$dir]]];
   If[!MemberQ[$Path,First[D2D$dir]],AppendTo[$Path,First[D2D$dir]]]];
Remove[D2D$paths,D2D$dir];
DeclarePackage["D2DArc2D`", {"D2DArc2D", "Arc2D", "Bulge2D",
"Complement2D"}];
DeclarePackage["D2DArcLength2D`", {"D2DArcLength2D", "ArcLength2D",
"Circumference2D", "Perimeter2D", "Span2D"}];
DeclarePackage["D2DArea2D`", {"D2DArea2D", "Area2D", "SectorArea2D",
"SegmentArea2D" }];
```

```
DeclarePackage["D2DCircle2D'", {"D2DCircle2D", "Circle2D", "Radius2D"}];
DeclarePackage["D2DConic2D'", {"D2DConic2D", "Asymptotes2D",
"Directrices2D", "Eccentricity2D", "FocalChords2D", "Foci2D",
"Vertices2D" }];
DeclarePackage["D2DConicArc2D'", {"D2DConicArc2D", "Apex2D", "ConicArc2D",
"Rho2D" }];
DeclarePackage["D2DEllipse2D'", {"D2DEllipse2D", "Ellipse2D",
"SemiMajorAxis2D", "SemiMinorAxis2D"}];
DeclarePackage["D2DEquations2D`", {"D2DEquations2D", "Equation2D",
"Polynomial2D", "SimplifyCoefficients2D" }];
DeclarePackage["D2DExpressions2D`", {"D2DExpressions2D", "IsApproximate2D",
"IsComplex2D", "IsNegative2D", "IsNumeric2D", "IsReal2D", "IsScalarPair2D",
"IsScalar2D", "IsTinyImaginary2D", "IsZero2D", "IsZeroOrNegative2D"}];
DeclarePackage["D2DGeometry2D`", {"D2DGeometry2D", "IsCoincident2D",
"IsCollinear2D", "IsConcentric2D", "IsConcurrent2D", "IsOn2D",
"IsParallel2D", "IsTripleParallel2D", "IsPerpendicular2D", "IsTangent2D"}];
DeclarePackage["D2DHyperbola2D`", {"D2DHyperbola2D", "Conjugate2D",
"Hyperbola2D", "SemiTransverseAxis2D", "SemiConjugateAxis2D"}];
DeclarePackage["D2DIntersect2D`", {"D2DIntersect2D", "Parameters2D",
"Points2D" }];
DeclarePackage["D2DLine2D`", {"D2DLine2D", "Angle2D", "Line2D",
"Parallel2D", "Perpendicular2D", "Slope2D"}];
DeclarePackage["D2DLoci2D`", {"D2DLoci2D", "Loci2D"}];
DeclarePackage["D2DMaster2D`", {"Is2D", "IsValid2D", "ObjectNames2D"}];
DeclarePackage["D2DMedial2D`", {"D2DMedial2D", "MedialEquations2D",
"MedialLoci2D"}];
DeclarePackage["D2DNumbers2D`", {"D2DNumbers2D", "ChopImaginary2D",
"PrimaryAngle2D", "PrimaryAngleRange2D"}];
DeclarePackage["D2DParabola2D'", {"D2DParabola2D", "FocalLength2D",
"Parabola2D"}];
DeclarePackage["D2DPencil2D'", {"D2DPencil2D", "Pencil2D"}];
DeclarePackage["D2DPoint2D'", {"D2DPoint2D", "Coordinates2D", "Distance2D",
"Point2D", "XCoordinate2D", "YCoordinate2D"}];
DeclarePackage["D2DQuadratic2D`", {"D2DQuadratic2D", "Quadratic2D"}];
DeclarePackage["D2DSegment2D'", {"D2DSegment2D", "Length2D", "Segment2D"}];
```

```
DeclarePackage["D2DSketch2D`", {"D2DSketch2D", "AskCurveLength2D",
"CurveLength2D", "CurveLimits2D", "IsDisplay2D", "MakePrimitives2D",
"SetDisplay2D", "Sketch2D"}];
DeclarePackage["D2DSolve2D`", {"D2DSolve2D", "MaxSeconds2D", "Solve2D"}];
DeclarePackage["D2DTangentCircles2D`", {"D2DTangentCircles2D",
"TangentCircles2D"}];
DeclarePackage["D2DTangentConics2D`", {"D2DTangentConics2D",
"TangentConics2D", "TangentQuadratics2D"}];
DeclarePackage["D2DTangentLines2D`", {"D2DTangentLines2D",
"TangentEquation2D", "TangentLines2D", "TangentSegments2D"}];
DeclarePackage["D2DTangentPoints2D`", {"D2DTangentPoints2D",
"TangentEquation2D", "TangentLines2D", "TangentSegments2D"}];
DeclarePackage["D2DTangentPoints2D`", {"D2DTangentPoints2D",
"TangentPoints2D"}];
DeclarePackage["D2DTangentPoints2D`", {"D2DTangentPoints2D",
"TangentPoints2D"}];
DeclarePackage["D2DTransform2D`", {"D2DTransform2D", "Reflect2D",
"ReflectAngle2D", "Rotate2D", "SolveTriangle2D", "Centroid2D",
"Circumscribed2D", "Inscribed2D", "SolveTriangle2D", "Triangle2D"}];
```

# Package Initialization

```
BeginPackage["D2DMaster2D`"];
D2DMaster2D::usage=
   "D2DMaster2D is the master package for Descarta2D.";
D2DMaster2D::loaded=
   "The package 'D2DMaster2D' has already been loaded.";
Is2D::usage=
   "Is2D[object,headList] returns 'True' if the object is valid and its
head is included in the list of object heads.";
IsValid2D::usage=
   "IsValid2D[object] returns 'True' if the object is syntactically valid.";
ObjectNames2D::usage=
   "ObjectNames2D[] returns a list of strings that are the names of all
Descarta2D objects.";
Begin["'Private'"];
```

# **Objects**

## **Object Names**

**ObjectNames2D[]** ■ Returns a list of strings that are the symbolic names for all *Descarta2D* objects.

```
ObjectNames2D[]:={"Arc2D", "Circle2D", "ConicArc2D", "Ellipse2D",
    "Hyperbola2D", "Line2D", "Parabola2D", "Point2D", "Quadratic2D",
    "Segment2D", "Triangle2D"};
```

# **Default Queries**

## Is Query

**Is2D**[*object*, *objHeadList*] ■ Returns **True** if an object is valid and has its head included in the objHeadList; otherwise, returns **False**.

```
Is2D[obj_,objHead_List] :=
  (IsValid2D[obj] && (Or @@ Map[(Head[obj]===#)&,objHead]));
```

### Valid Query

**IsValid2D**[*object*] ■ Returns **True** if the object is geometric and is syntactically valid; otherwise, returns **False**. Only the default case is implemented here.

```
IsValid2D[___] := False;
```

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DMaster2D`" *)
```

# D2DMedial2D

The package D2DMedial2D constructs curves that are equidistant from two points, lines or circles.

# Initialization

```
BeginPackage["D2DMedial2D`", {"D2DCircle2D`", "D2DExpressions2D`",
"D2DGeometry2D`", "D2DMaster2D`", "D2DLine2D`", "D2DLoci2D`",
"D2DPoint2D`", "D2DQuadratic2D`"}];
D2DMedial2D::usage=
    "D2DMedial2D is a package that constructs medial equations and loci.";
MedialEquations2D::usage=
    "MedialEquations2D[{obj,obj}] constructs a list of lines or quadratics
equidistant from two objects (points, lines or circles).";
MedialLoci2D::usage=
    "MedialLoci2D[{obj,obj}] constructs a list of curves equidistant from
two objects (points, lines or circles).";
Begin["`Private`"];
MedialEquations2D::coincident=
    "The objects {`1`, `2`} are coincident; no finite number of medial
equations exist.";
```

# **Medial Equations**

## Medial Linear or Quadratic

MedialEquations2D[ $\{obj_1, obj_2\}$ ]  $\blacksquare$  Constructs a list of lines or quadratics equidistant from two objects (points, lines or circles).

```
MedialEquations2D[{obj1_,obj2_}] :=
    If[TrueQ[IsCoincident2D[obj1,obj2]],
        Message[MedialEquations2D::coincident,obj1,obj2];{},
        Medial$2D[Reverse[Sort[{obj1,obj2}]]]] /;
    Is2D[obj1,{Point2D,Line2D,Circle2D}] &&
    Is2D[obj2,{Point2D,Line2D,Circle2D}];
```

## **Medial Loci**

### Medial Loci

MedialLoci2D[ $\{obj_1, obj_2\}$ ]  $\blacksquare$  Constructs a list of curves equidistant from two objects (points, lines or circles).

```
MedialLoci2D[{obj1_,obj2_}] :=
    Union[
        Flatten[
            Map[If[Head[#]===Line2D,#,Loci2D[#]]&,
                 MedialEquations2D[{obj1,obj2}]]]] /;
Is2D[obj1,{Point2D,Line2D,Circle2D}] &&
Is2D[obj2,{Point2D,Line2D,Circle2D}];
```

## **Point-Point**

The private function Medial\$2D constructs a list of one line equidistant from two points.

```
Medial$2D[{Point2D[{x1_,y1_}],Point2D[{x2_,y2_}]}] :=
{Line2D[2*(x2-x1),2*(y2-y1),(x1^2+y1^2)-(x2^2+y2^2)]};
```

## **Point-Line**

The private function Medial\$2D constructs a list of one quadratic representing the curve equidistant from a point and a line.

```
Medial$2D[{Point2D[{x1_,y1_}],Line2D[a2_,b2_,c2_]}] :=
Module[{a,b,c,d,e,f,p,q,r},
{p,q,r}={a2,b2,c2}/Sqrt[a2^2+b2^2];
a=q^2;
b=-2*p*q;
c=p^2;
d=-2*(x1+p*r);
e=-2*(y1+q*r);
f=x1^2+y1^2-r^2;
{Quadratic2D[a,b,c,d,e,f]} ];
```

## Point-Circle

The private function Medial\$2D constructs a list of one quadratic representing the curve equidistant from a point and a circle.

```
Medial$2D[{Point2D[{x1_,y1_}],Circle2D[{h2_,k2_},r2_]}] :=
Module[{R,a,b,c,d,e,f},
R=(h2^2+k2^2)-(x1^2+y1^2)-r2^2;
a=4((x1-h2)^2-r2^2);
b=8*(x1-h2)*(y1-k2);
c=4((y1-k2)^2-r2^2);
d=4*(R*(x1-h2)+2*r2^2*x1);
e=4*(R*(y1-k2)+2*r2^2*y1);
f=R^2-4*r2^2*(x1^2+y1^2);
{Quadratic2D[a,b,c,d,e,f]} ];
```

### Line-Line

The private function Medial\$2D constructs a list of two lines equidistant from two lines (the angle bisectors). If the lines are parallel, only one line is returned in the list.

```
Medial$2D[{L1:Line2D[a1_,b1_,c1_],L2:Line2D[a2_,b2_,c2_]}] :=
Module[{a,b,c,f1,f2,s},
f1=Sqrt[a1^2+b1^2];
f2=Sqrt[a2^2+b2^2];
a=a1*f2+s*a2*f1;
b=b1*f2+s*b2*f1;
c=c1*f2+s*c2*f1;
If[IsParallel2D[L1,L2],
If[IsParallel2D[L1,L2],
[Line2D[a,b,c]] /. s->1],
{Line2D[a,b,c]} /. s->1],
Map[(Line2D[a,b,c] /. s->#)&,{-1,1}]];
```

### Line-Circle

The private function Medial\$2D constructs a list of two quadratics representing curves equidistant from a line and a circle.

```
Medial$2D[{Line2D[a1_,b1_,c1_],Circle2D[{h2_,k2_},r2_]}] :=
Module[{a,b,c,d,e,f,p,q,r,s},
{p,q,r}={a1,b1,c1}/Sqrt[a1^2+b1^2];
a=q^2;
b=-2*p*q;
c=p^2;
d=-2*(h2+p*(r+s*r2));
e=-2*(k2+q*(r+s*r2));
f=(h2^2+k2^2)-r2^2-r*(r+2*s*r2);
Map[(Quadratic2D[a,b,c,d,e,f] /. s->#)&, {-1,1}]];
```

### Circle–Circle

The private function Medial\$2D constructs a list of two quadratics representing the curves equidistant from two circles.

```
Medial$2D[{Circle2D[{h1_,k1_},r1_],Circle2D[{h2_,k2_},r2_]}] :=
Module[{s,R,D1,D2,a,b,c,d,e,f},
R=(r1-s*r2)^2;
D1=h1^2+k1^2;
D2=h2^2+k2^2;
a=4*((h1-h2)^2-R);
b=8*(h1-h2)*(k1-k2);
c=4*((k1-k2)^2-R);
d=4*(h1*(-D1+D2+R)+h2*(D1-D2+R));
e=4*(k1*(-D1+D2+R)+k2*(D1-D2+R));
f=(D1-D2)^2-2*(D1+D2)*R+R^2;
Map[(Quadratic2D[a,b,c,d,e,f] /. s->#)&, {-1,1}]];
```

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DMedial2D`" *)
```

# D2DNumbers2D

The package D2DNumbers2D provides functions for special manipulations of real numbers.

# Initialization

```
BeginPackage["D2DNumbers2D`", {"D2DExpressions2D`", "D2DMaster2D`"}];
D2DNumbers2D::usage=
  "D2DNumbers2D is a package that provides utilities for manipulating
special numbers.";
ChopImaginary2D::usage=
  "ChopImaginary2D[expr,(tol)] removes tiny imaginary parts from complex
numbers; the default tolerance, if omitted, is 10^(-10).";
PrimaryAngle2D::usage=
  "PrimaryAngle2D[theta,period] normalizes an angle to a period; the
period must be Pi or 2Pi radians; if the period is omitted, it defaults to
2Pi.";
PrimaryAngleRange2D::usage=
  "PrimaryAngleRange2D[{t1,t2}] normalizes a range of angles to primary
angles.";
Begin["'Private'"];
```

## **Chop Imaginary Part**

**ChopImaginary2D**[*expr*, (tol)]  $\blacksquare$  Removes the tiny imaginary parts of complex numbers in the expression that are less than a given tolerance. The default tolerance, if omitted, is  $10^{-10}$ .

```
ChopImaginary2D[expr_,tol_:(10^(-10))] :=
MapAll[If[IsTinyImaginary2D[#],Re[#],#]&,expr] /;
TrueQ[N[tol]>=0];
```

### **Primary Angle**

**PrimaryAngle2D**[ $\theta$ , 2**Pi** | **Pi**] ■ Adjusts an angle to a primary angle in the range  $0 \le \theta < p$ . The period, p, may be **Pi** or 2**Pi** radians. The default period, if omitted, is 2**Pi** radians.

```
PrimaryAngle2D[theta_?IsScalar2D,period_:2Pi] :=
Module[{thetal=theta},
While[IsNegative2D[theta1],
thetal+=period];
While[IsZeroOrNegative2D[period-theta1],
thetal-=period];
If[Head[theta]===Real,N[theta1],theta1] ] /;
(period==Pi || period==2Pi);
```

### **Primary Angle Range**

**PrimaryAngleRange2D**[ $\{\theta_1, \theta_2\}$ ]  $\blacksquare$  Normalizes a list of two angles so that the first is in the range  $0 \le \phi_1 < 2\pi$ , and the second is in the range  $\phi_1 < \phi_2 < (\phi_1 + 2\pi)$ .

```
PrimaryAngleRange2D[{t1_?ISScalar2D,t2_?ISScalar2D}] :=
Module[{T1,T2,twoPi},
T1=PrimaryAngle2D[t1];
T2=PrimaryAngle2D[t2];
twoPi=If[Head[T2]==Real,N[2Pi],2Pi];
If[ISZeroOrNegative2D[T2-T1],
{T1,T2+twoPi},
{T1,T2}] ];
```

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DNumbers2D`" *)
```

# D2DParabola2D

The package D2DParabola2D implements the Parabola2D object.

# Initialization

```
BeginPackage["D2DParabola2D`", {"D2DExpressions2D`", "D2DGeometry2D`",
"D2DLine2D`", "D2DMaster2D`", "D2DNumbers2D`", "D2DPoint2D`",
"D2DQuadratic2D`", "D2DSegment2D`", "D2DSketch2D`", "D2DTransform2D`"}];
D2DParabola2D::usage=
    "D2DParabola2D is a package that implements the Parabola2D object.";
FocalLength2D::usage=
    "FocalLength2D[parabola] returns the focal length of a parabola.";
Parabola2D::usage=
    "Parabola2D[{h,k},f,theta] is the standard form of a parabola that opens
to the right (when theta=0); {x,y} is the vertex point of the parabola; 'f'
is the distance from the vertex point to the focus; 'theta' is the
counter-clockwise rotation (in radians) of the parabola about the vertex
point.";
```

```
Begin["'Private'"];
```

# Description

### Representation

**Parabola2D**[ $\{h, k\}, f, \theta$ ]  $\blacksquare$  Standard representation of a parabola in *Descarta2D*. The first argument is a list of coordinates representing the position of the vertex of the parabola. The second argument is a (positive) scalar representing the distance from the vertex to the focus (the focal length). The third argument is the counter-clockwise rotation (in radians) of the parabola about the vertex point.

### Equation

Quadratic2D[parabola] Constructs the quadratic representing the equation of a parabola.

```
Quadratic2D[Parabola2D[{h_,k_},f_,theta_]] :=
Rotate2D[Quadratic2D[0,0,1,-4*f,-2*k,k^2+4*h*f],theta,{h,k}];
```

## **Evaluation**

**Parabola2D**[ $\{h, k\}, f, \theta$ ][t]  $\blacksquare$  Evaluates a parabola at a parameter  $t \ (-\infty < t < +\infty)$ . The end points of the latus rectum are at t = -1 and t = 1.

```
Parabola2D[{h_,k_},f_,theta_][t_?IsScalar2D] :=
Rotate2D[{h+f*t^2,k+2*f*t},theta,{h,k}];
```

## Graphics

Provides graphics for a parabola by extending the *Mathematica* Display command. Executed when the package is loaded.

```
SetDisplay2D[
Parabola2D[{h_,k_},f_,t_][{t1_?IsScalar2D,t2_?IsScalar2D}],
MakePrimitives2D[Parabola2D[{h,k},f,t],{t1,t2}] ];
SetDisplay2D[
Parabola2D[{h_,k_},f__,t_],
MakePrimitives2D[Parabola2D[{h,k},f,t],
CurveLimits2D[{0,0},
Parabola2D[{0,0},f,0]]] ];
```

### Validation

**Parabola2D**[ $\{h, k\}, f, \theta$ ]  $\blacksquare$  Detects a parabola with imaginary arguments and returns the **\$Failed** symbol. If the imaginary parts are insignificant, they are removed.

```
Parabola2D::imaginary=
    "An invalid parabola of the form 'Parabola2D['1', '2', '3']' has been
detected; the arguments cannot be imaginary.";
Parabola2D[{h_,k_},f_,theta_] :=
    (Parabola2D @@ ChopImaginary2D[Parabola$2D[{h,k},f,theta]]) /;
(FreeQ[{h,k,f,theta},_Pattern] && IsTinyImaginary2D[{h,k,f,theta}]);
Parabola2D[{h_,k_},f_,theta_] :=
    (Message[Parabola2D::imaginary,{h,k},f,theta];$Failed) /;
(FreeQ[{h,k,f,theta},_Pattern] && IsComplex2D[{h,k,f,theta},0]);
```

**Parabola2D**[ $\{h, k\}, f, \theta$ ]  $\blacksquare$  Detects a parabola with an invalid focal length. If the focal length is negative, the parabola is rotated  $\pi$  radians to make it positive; if the focal length is zero, the **\$Failed** symbol is returned.

```
Parabola2D::invalid=
    "An invalid parabola of the form 'Parabola2D['1', '2', '3']' has been
detected; the focal length cannot be zero.";
Parabola2D[{h_,k_},f_,theta_] :=
    (Message[Parabola2D::invalid,{h,k},f,theta];$Failed) /;
(FreeQ[{h,k,f,theta},_Pattern] && IsZero2D[f,0]);
Parabola2D[{h_,k_},f_,theta_] :=
    Parabola2D[{h,k},-f,theta+Pi] /;
(FreeQ[{h,k,f,theta},_Pattern] && IsNegative2D[f,0]);
```

**Parabola2D**[{h, k},  $f, \theta$ ] Adjusts the rotation angle on a parabola to the range  $0 \le \theta < 2\pi$ .

```
Parabola2D[{h_,k_},f_,theta_] :=
    Parabola2D[{h,k},f,PrimaryAngle2D[theta]] /;
  (FreeQ[{h,k,f,theta},_Pattern] && (theta=!=PrimaryAngle2D[theta]));
```

**IsValid2D**[*parabola*] ■ Verifies that a parabola is syntactically valid.

# Scalars

## **Angle of Rotation**

**Angle2D**[*parabola*] ■ Returns the rotation angle of a parabola.

 $\label{eq:angle2D[Parabola2D[{h_,k_},f_,theta_]] := theta;$ 

## **Focal Length**

**FocalLength2D**[*parabola*] ■ Returns the focal length of a parabola.

```
\label{eq:focalLength2D[Parabola2D[{h_,k_},f_,theta_]] := f;
```

# Transformations

### Reflect

**Reflect2D**[*parabola*, *line*] ■ Reflects a parabola in a line.

```
Reflect2D[Parabola2D[{h_,k_},f_,theta_],L2:Line2D[a2_,b2_,c2_]] :=
Parabola2D[Reflect2D[{h,k},L2],f,ReflectAngle2D[theta,L2]];
```

#### Rotate

Rotate2D[*parabola*,  $\theta$ , *coords*]  $\blacksquare$  Rotates a parabola by an angle  $\theta$  about a position specified by a coordinate list. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

### Scale

Scale2D[*parabola*, *s*, *coords*] ■ Scales a parabola from a position given by coordinates. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

### Translate

Translate2D[parabola, {u, v}] ■ Translates a parabola delta distance.

```
\label{eq:lisscalar2D[Parabola2D[ {h_,k_},f_,theta_], {u_?IsScalar2D,v_?IsScalar2D}] := Parabola2D[ {h+u,k+v},f,theta];
```

# **Point Construction**

### **Vertex Point**

**Point2D**[*parabola*] ■ Returns the vertex point of a parabola.

```
Point2D[Parabola2D[{h_,k_},f_,theta_]] := Point2D[{h,k}];
```

## **Pole Point**

 $Point2D[line, parabola] \square$  Constructs the pole (point) of a polar (line) with respect to a parabola. If the line is tangent to the parabola then the point is the point of tangency.

```
Point2D[L1:Line2D[a1_,b1_,c1_],P2:Parabola2D[{h_,k_},f_,theta_]] :=
Point2D[L1,Quadratic2D[P2]];
```

## Line Construction

## **Axis Line**

Line2D[*parabola*] Constructs a line that contains the axis of a parabola.

```
Line2D[Parabola2D[{h_,k_},f_,theta_]] :=
Rotate2D[Line2D[0,1,-k],theta,{h,k}];
```

## **Polar Line**

Line2D[*point*, *parabola*] Constructs the polar (line) of a pole (point) with respect to a parabola. If the point is on the parabola then the line is tangent to the parabola at the point.

```
Line2D[P1:Point2D[{x1_,y1_}],P2:Parabola2D[{h_,k_},f_,theta_]] := Line2D[P1,Quadratic2D[P2]];
```

# Parabola Construction

### Parabola from Focus/Directrix

Parabola2D[point, line] Constructs a parabola from a focus point and a directrix line.

```
Parabola2D::invptln=
    "The focus `1` is on the directrix `2`; no valid parabola can be
constructed.";
Parabola2D[P1:Point2D[{x1_,y1_}],L2:Line2D[a2_,b2_,c2_]] :=
    Module[{pt},
    If[IsOn2D[P1,L2],
    Message[Parabola2D::invptln,P1,L2];$Failed,
    pt=Point2D[P1,L2];
    Parabola2D[Coordinates2D[Point2D[P1,pt]],
        Distance2D[P1,pt]/2,
        ArcTan[x1-XCoordinate2D[pt]]] ];
```

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DParabola2D`" *)
```

# D2DPencil2D

The package D2DPencil2D implements functions for computing families of *Descarta2D* curves (lines, circles and quadratics).

# Initialization

```
BeginPackage["D2DPencil2D`", {"D2DCircle2D`", "D2DExpressions2D`",
"D2DGeometry2D`", "D2DLine2D`", "D2DQuadratic2D`", "D2DPoint2D`"}];
D2DPencil2D::usage=
    "D2DPencil2D is a package that construction pencil curves.";
Pencil2D::usage=
    "Pencil2D is a keyword used to specify the formation of a pencil of
objects.";
Begin["`Private`"];
```

# **Line Pencils**

## Pencil of Lines Through a Point

Line2D[*point*,  $\theta$ , Pencil2D]  $\blacksquare$  Constructs a line parameterized by the variable  $\theta$  representing the pencil of lines through a point. The variable  $\theta$  is the angle of rotation of the line.

```
Line2D[Point2D[{x_,y_}],t_?IsScalar2D,Pencil2D] :=
Line2D[-Sin[t],Cos[t],x*Sin[t]-y*Cos[t]];
```

### Pencil of Lines Through Intersection Point

Line2D[*line*, *line*, *k*, Pencil2D]  $\blacksquare$  Constructs a line parameterized by the variable *k* representing the pencil of lines through the intersection of two lines.

```
Line2D[Line2D[a1_,b1_,c1_],Line2D[a2_,b2_,c2_],k_?IsScalar2D,Pencil2D] := Line2D[(1-k)*al+k*a2,(1-k)*bl+k*b2,(1-k)*c1+k*c2];
```

## **Circle Pencils**

### Pencil of Circles from Two Circles

Circle2D[*circle*, *circle*, *k*, Pencil2D]  $\blacksquare$  Constructs a circle parameterized by the variable *k* representing the pencil of circles having common intersection points with two given circles.

# **Quadratic Pencils**

### Pencil of Quadratics from Two Quadratics

Quadratic2D[quad, quad, k, Pencil2D]  $\blacksquare$  Constructs a quadratic parameterized by the variable k representing the pencil of quadratics through the intersection points of two given quadratics.

### Pencil of Quadratics from Four Lines

Quadratic2D[{line, line}, {line, line}, k, Pencil2D]  $\blacksquare$  Constructs a quadratic parameterized by the variable k representing the pencil of quadratics passing through the four intersection points of four lines taken in predetermined pairs (1-2 with 3-4, and 1-3 with 2-4).
### Pencil of Quadratics from Four Points

Quadratic2D[point, point, point, point, k, Pencil2D]  $\blacksquare$  Constructs a quadratic parameterized by the variable k representing the pencil of quadratics passing through four points.

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DPencil2D`" *)
```

# D2DPoint2D

The package D2DPoint2D implements the Point2D object.

# Initialization

```
BeginPackage["D2DPoint2D'", {"D2DExpressions2D'", "D2DGeometry2D'",
"D2DLine2D`", "D2DMaster2D`", "D2DNumbers2D`", "D2DQuadratic2D`",
"D2DSketch2D`", "D2DTransform2D`" }];
D2DPoint2D::usage=
   "D2DPoint2D is a package that implements the Point2D object.";
Coordinates2D::usage=
   "Coordinates2D[a1,a2,...] gives the \{x,y\} coordinates of the point
returned by Point2D[a1,a2,...].";
Distance2D::usage=
   "Distance2D[coords,coords] gives the distance between two positions
given by coordinates; Distance2D[point,pt | ln | cir] gives the distance
between a point and a point, line or circle.";
Point2D::usage=
  "Point2D[\{x,y\}] is the standard form of a point at coordinates \{x,y\}.";
XCoordinate2D::usage=
   "XCoordinate2D[point] gives the x-coordinate of a point;
XCoordinate2D[coords] gives the x-coordinate of a coordinate location.";
YCoordinate2D::usage=
   "YCoordinate2D[point] gives the y-coordinate of a point;
YCoordinate[coords] gives the y-coordinate of a coordinate location.";
Begin["'Private'"];
```

# Description

### Representation

 $\{x, y\} \equiv$  Standard representation of a location specified by (x, y) in Descarta2D.

 $Point2D[{x, y}]$  or  $Point2D[coords] \blacksquare$  Standard representation of a point in Descarta2D. The coordinates define the (x, y) position of the point.

### Graphics

Provides graphics for a point by extending the *Mathematica* Display command. Executed when the package is loaded.

```
SetDisplay2D[
Point2D[{x_,y_}],
{AbsolutePointSize[4],Point[{x,y}]} ];
```

### Validation

 $Point2D[{x, y}] \blacksquare$  Detects that a point has imaginary coordinates and returns the **\$Failed** symbol. If the imaginary parts are insignificant, they are removed.

```
Point2D::imaginary=
   "An invalid point of the form 'Point2D[{`1`,`2`}]' has been detected;
the coordinates of a point cannot be imaginary.";
Point2D[{x_,y_}] :=
   (Point2D @@ ChopImaginary2D[Point$2D[{x,y}]]) /;
(FreeQ[{x,y},_Pattern] && IsTinyImaginary2D[{x,y}]);
Point2D[{x_,y_}] :=
   (Message[Point2D::imaginary,x,y];$Failed) /;
(FreeQ[{x,y},_Pattern] && IsComplex2D[{x,y},0]);
```

**IsValid2D**[*point*] ■ Verifies that a point is syntactically valid.

```
IsValid2D[Point2D[{x_?IsScalar2D,y_?IsScalar2D}]] := True;
```

# Scalars

### **Coordinates Function**

Coordinates2D[args..]  $\blacksquare$  Returns the {x, y} coordinates of Point2D[args..].

```
Coordinates2D[a___]:=
Module[{pt},
If[pt===$Failed,pt,{XCoordinate2D[pt],YCoordinate2D[pt]}] /;
(pt=Point2D[a] /. Point2D[Point2D[x__]]->Point2D[x];
(Is2D[pt,{Point2D}] || pt===$Failed))];
```

#### **Distance between Coordinates**

**Distance2D**[*coords*, *coords*] ■ Computes the distance between two locations defined by coordinates.

#### **Distance between Two Points**

**Distance2D**[*point*, *point*] ■ Computes the distance between two points.

```
Distance2D[Point2D[{x1_,y1_}],Point2D[{x2_,y2_}]] :=
Sqrt[(x1-x2)^2+(y1-y2)^2];
```

### X-Coordinate (Abscissa)

XCoordinate2D[coords]  $\blacksquare$  Returns the x-coordinate.

```
XCoordinate2D[{x_?IsScalar2D,y_?IsScalar2D}] := x;
```

#### X-Coordinate of a Point (Abscissa)

**XCoordinate2D**[*point*] ■ Returns the *x*-coordinate of a point.

```
\label{eq:coordinate2D[Point2D[{x_,y_}]] := x;} \\
```

### Y-Coordinate (Ordinate)

YCoordinate2D[coords]  $\blacksquare$  Returns the *y*-coordinate.

```
YCoordinate2D[{x_?IsScalar2D,y_?IsScalar2D}] := y;
```

#### Y-Coordinate of a Point (Ordinate)

YCoordinate2D[point] Returns the *y*-coordinate of a point.

 $\texttt{YCoordinate2D[Point2D[{x_,y_}]] := y;}$ 

# Equations

### Quadratic

**Quadratic2D**[*point*] ■ Constructs the quadratic representing the equation of a point (a point circle).

```
Quadratic2D[Point2D[{x_,y_}]] := Quadratic2D[1,0,1,-2*x,-2*y,x^2+y^2];
```

# Transformations

#### Reflect

**Reflect2D**[*point*, *line*] ■ Reflects a point in a line.

```
Reflect2D[Point2D[{x,y}],L:Line2D[a_,b_,c_]] :=
Point2D[ Reflect2D[{x,y},L] ];
```

#### Rotate

Rotate2D[*point*,  $\theta$ , *coords*]  $\blacksquare$  Rotates a point by an angle  $\theta$  about a position specified by coordinates. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

### Scale

Scale2D[*point*, *s*, *coords*] ■ Scales a point from a position given by coordinates. If the argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

### Translate

Translate2D[point,  $\{u, v\}$ ]  $\blacksquare$  Translates a point delta distance.

```
\label{eq:lisscalar2D[Point2D[{x_,y_}], {u_!IsScalar2D,v_!IsScalar2D}] := Point2D[{x+u,y+v}];
```

# **Point Construction**

#### **Point from Coordinates**

 $Point2D[coords] \blacksquare$  Constructs a point from the x- and y-coordinates.

```
Point2D[x_?IsScalar2D,y_?IsScalar2D] := Point2D[{x,y}];
```

#### **Midpoint of Two Points**

**Point2D**[*point*, *point*] ■ Constructs the midpoint of two given points.

```
Point2D[Point2D[{x1_,y1_}],Point2D[{x2_,y2_}]] := Point2D[{x1+x2,y1+y2}/2];
```

### Offset Point from Two Points

Point2D[*point*, *point*, *d*]  $\blacksquare$  Constructs a point at a given distance from the first point on the line joining the two given points.

```
Point2D::noDir=
   "Points {`1`, `2`} are coincident and do not define a valid direction.";
Point2D[P1:Point2D[{x1_,y1_}],P2:Point2D[{x2_,y2_}],d_?IsScalar2D] :=
   Module[{d12=Sqrt[(x2-x1)^2+(y2-y1)^2]},
    If[IsZero2D[d12],
        Message[Point2D::noDir,P1,P2];$Failed,
        Point2D[{x1,y1}+{d*(x2-x1),d*(y2-y1)}/d12]] ];
```

### Point of Division

Point2D[point, point,  $r_1$ ,  $r_2$ ]  $\blacksquare$  Constructs a point which divides a line segment determined by two points into the ratio  $r_1/r_2$ .

```
Point2D::noRatio=
    "The sum of the ratio numbers {`1`, `2`} cannot be zero.";
Point2D[Point2D[{x1_,y1_}],Point2D[{x2_,y2_}],
    r1_?IsScalar2D,r2_?IsScalar2D] :=
    If[IsZero2D[r1+r2],
        Message[Point2D::noRatio,r1,r2];$Failed,
        Point2D[{x1*r2+x2*r1,y1*r2+y2*r1}/(r1+r2)];
```

## Point Offset Along a Line

Point2D[*point*, *line*, d]  $\blacksquare$  Constructs a point at a distance d from a given point in the direction of a given line. If the given point is on the line then the offset point will also be on the line. The distance may be positive or negative producing one of the two possible points.

```
Point2D[Point2D[{x1_,y1_}],Line2D[A2_,B2_,C2_],d_?IsScalar2D] :=
Point2D[{x1,y1}+{-d*B2,d*A2}/Sqrt[A2^2+B2^2]];
```

#### Point Projected Onto a Line

**Point2D**[*point*, *line*] ■ Constructs a point on a line by projecting a given point onto the line.

### **General Offset Point**

**Point2D**[*point*, *line*,  $\{u, v\}$ ]  $\blacksquare$  Constructs a point offset from the origin of an inferred coordinate system. The given point is on the +x-axis of the inferred coordinate system and the given line is the *y*-axis. The point constructed has coordinates (u, v) in the inferred coordinate system.

### **Intersection Point of Two Lines**

**Point2D**[*line*, *line*] ■ Constructs the intersection point of two lines.

```
Point2D::coincident=
  "No unique intersection point exists; lines '1' and '2' are coincident.";
Point2D::parallel=
  "No intersection point exists; lines '1' and '2' are parallel.";
Point2D[L1:Line2D[A1_,B1_,C1_],L2:Line2D[A2_,B2_,C2_]] :=
  Which[
    IsCoincident2D[L1,L2],
    Message[Point2D::coincident,L1,L2];$Failed,
    IsParallel2D[L1,L2],
    Message[Point2D::parallel,L1,L2];$Failed,
    True,
        Point2D[{B1*C2-B2*C1,A2*C1-A1*C2}/(A1*B2-A2*B1)] ];
```

## Center Point of a Quadratic

**Point2D**[*quad*] ■ Constructs the center point of a central quadratic.

```
Point2D::notCentral=
    "'1' is not a central conic; it has no center point.";
Point2D[Q1:Quadratic2D[a_,b_,c_,d_,e_,f_]] :=
    Module[{disc=b^2-4*a*c},
    If[IsZero2D[disc],
        Message[Point2D::notCentral,Q1];$Failed,
        Point2D[{2*c*d-b*e,2*a*e-b*d}/(b^2-4*a*c)]] ];
```

### Pole Point of a Quadratic

**Point2D**[*line*, *quad*] ■ Constructs a pole (point) of a quadratic with respect to a polar (line).

```
Point2D::noPole=
   "Since `1` passes through the center of the conic, no pole point
exists.";
Point2D[L1:Line2D[A1_,B1_,C1_],Q2:Quadratic2D[a_,b_,c_,d_,e_,f_]] :=
   Module[{q12,q1,q2},
      q12=A1*(b*e-2*c*d)+B1*(b*d-2*a*e)+C1*(4*a*c-b*2);
      If[IsZero2D[q12],
           Message[Point2D::noPole,L1,Q2];$Failed,
       q1=A1*(4*c*f-e*2)+B1*(d*e-2*b*f)+C1*(b*e-2*c*d);
       q2=A1*(d*e-2*b*f)+B1*(4*a*f-d*2)+C1*(b*d-2*a*e);
       Point2D[{q1,q2}/q12]] ];
```

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DPoint2D`" *)
```

# D2DQuadratic2D

The package D2DQuadratic2D implements the Quadratic2D object.

# Initialization

```
BeginPackage["D2DQuadratic2D`", {"D2DEquations2D`", "D2DExpressions2D`",
"D2DLine2D`", "D2DLoci2D`", "D2DMaster2D`", "D2DNumbers2D`", "D2DPoint2D`",
"D2DSketch2D`", "D2DTransform2D`"}];
D2DQuadratic2D::usage=
   "D2DQuadratic2D is a package providing support for the quadratic
object.";
Quadratic2D[a,b,c,d,e,f] represents the polynomial
a*x^2+b*x*y+c*y^2+d*x+e*y+f.";
Begin["`Private`"];
```

# Description

### Representation

Quadratic2D[a, b, c, d, e, f]  $\blacksquare$  A quadratic is used to represent a quadratic polynomial in two unknowns. Quadratic2D[a, b, c, d, e, f] represents  $ax^2 + bxy + cy^2 + dx + ey + f$ .

## Graphics

Provides graphics primitives for a quadratic by extending the *Mathematica* Display command. Executed when the package is loaded.

```
SetDisplay2D[
   Quadratic2D[a_,b_,c_,d_,e_,f_],
   Loci2D[Quadratic2D[a,b,c,d,e,f]]];
```

#### Validation

Quadratic2D[a, b, c, d, e, f]  $\blacksquare$  Detects a quadratic with imaginary coefficients and returns the **\$Failed** symbol. If the imaginary parts are insignificant, they are removed.

```
Quadratic2D::imaginary=
    "An invalid quadratic of the form 'Quadratic2D['1', '2', '3', '4', '5',
'6']' has been detected; the arguments cannot be imaginary.";
Quadratic2D[a_,b_,c_,d_,e_,f_] :=
    (Quadratic2D @@ ChopImaginary2D[Quadratic$2D[a,b,c,d,e,f]]) /;
(FreeQ[{a,b,c,d,e,f},_Pattern] && IsTinyImaginary2D[{a,b,c,d,e,f}]);
Quadratic2D[a_,b_,c_,d_,e_,f_] :=
    (Message[Quadratic2D::imaginary,a,b,c,d,e,f];$Failed) /;
(FreeQ[{a,b,c,d,e,f},_Pattern] && IsComplex2D[{a,b,c,d,e,f},0]);
```

Quadratic2D[a, b, c, d, e, f]  $\blacksquare$  Returns the **\$Failed** symbol when an invalid quadratic is detected (the first five coefficients are zero). Also, normalizes quadratics with tiny coefficients to improve numerical stability.

```
Quadratic2D::invalid=
    "An invalid quadratic of the form 'Quadratic2D['1', '2', '3', '4', '5',
    '6']' was encountered; at least one of the first five coefficients must be
non-zero.";
Quadratic2D[a_,b_,c_,d_,e_,f_] :=
    (Message[Quadratic2D::invalid,a,b,c,d,e,f];$Failed) /;
(FreeQ[{a,b,c,d,e,f},_Pattern] && IsZero2D[{a,b,c,d,e},And,0]);
Quadratic2D[a_,b_,c_,d_,e_,f_] :=
    (Quadratic2D @@ ({a,b,c,d,e,f}/Sqrt[a^2+b^2+c^2+d^2+e^2])) /;
(FreeQ[{a,b,c,d,e,f},_Pattern] && IsZero2D[{a,b,c,d,e},And]);
```

**IsValid2D**[*quad*] ■ Verifies that a quadratic is valid.

# Transformations

### Reflect

**Reflect2D**[*quad*, *line*] ■ Reflects a quadratic in a line.

```
Reflect2D[Q:Quadratic2D[a_,b_,c_,d_,e_,f_],L:Line2D[p_,q_,r_]] :=
Module[{eq1,eq2,x,y},
    eq1=Equation2D[Q,{x,y}];
    eq2=Reflect2D[eq1,{x,y},L];
    Quadratic2D[eq2,{x,y}] ];
```

#### Rotate

Rotate2D[quad,  $\theta$ , coords]  $\blacksquare$  Rotates a quadratic by an angle  $\theta$  about a position specified by a coordinate list. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

### Scale

Scale2D[quad, s, coords] ■ Scales a quadratic from a position given by coordinates. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

#### Translate

Translate2D[quad,  $\{u, v\}$ ]  $\blacksquare$  Translates a quadratic delta distance.

# **Quadratic Construction**

### Simplify and FullSimplify

Simplify[quad] and FullSimplify[quad] ■ Extends the Mathematica Simplify and FullSimplify commands to simplify the coefficients of a quadratic by factoring out common factors. Executed when the package is loaded.

### Normalize

Quadratic2D[quad]  $\blacksquare$  Normalizes the coefficients of a quadratic so that the sum of the squares of the first five coefficients equals one.

```
Quadratic2D[Quadratic2D[a_,b_,c_,d_,e_,f_]] :=
 (Quadratic2D @@ ({a,b,c,d,e,f}/Sqrt[a^2+b^2+c^2+d^2+e^2]));
```

### Quadratic from Equation/Polynomial

Quadratic2D[*expr*,  $\{x, y\}$ ] Forms a quadratic from a polynomial or equation in two unknowns. For example, the expression  $ax^2 + bxy + cy^2 + dx + ey + f == 0$  will return Quadratic2D[*a*, *b*, *c*, *d*, *e*, *f*]; the polynomial  $ax^2 + bxy + cy^2 + dx + ey + f$  will also return Quadratic2D[*a*, *b*, *c*, *d*, *e*, *f*]. The *x* and *y* arguments are assumed to be the names of the variables.

```
Quadratic2D::noPoly=
   "The expression '1' cannot be recognized as a quadratic polynomial or
equation in variables '2' and '3'.";
Quadratic2D[expr_,\{x_,y_\}] :=
   Module[\{eqn,a,b,c,d,e,f\},
      eqn=If[Head[expr]===Equal,
             expr[[1]]-expr[[2]],
             expr] //Expand;
      a=Coefficient[eqn,x^2];
      b=Coefficient[eqn,x*y];
      c=Coefficient[eqn,y^2];
      d=Coefficient[Expand[eqn /. {x*y->0}],x];
      e=Coefficient[Expand[eqn /. {x*y->0}],y];
      f=(eqn /. {x->0,y->0}) //Expand;
      \label{eq:liszero2D[a*x^2+b*x*y+c*y^2+d*x+e*y+f-eqn],} \\
         Quadratic2D[a,b,c,d,e,f],
         Message[Quadratic2D::noPoly,expr,x,y];$Failed] ];
```

### **Quadratic from Coordinates**

Quadratic2D[*coords*] ■ Forms a (degenerate) quadratic from a point given by a coordinate list (a point circle).

```
Quadratic2D[{x_?IsScalar2D,y_?IsScalar2D}] :=
Quadratic2D[1,0,1,-2*x,-2*y,x^2+y^2];
```

#### **Quadratic Through Three Points**

Quadratic2D[*point*, *point*] Constructs a quadratic (circle) that passes through three points.

```
Quadratic2D[Point2D[{x1_,y1_}],Point2D[{x2_,y2_}],Point2D[{x3_,y3_}]] :=
Module[{eqn,x,y},
    eqn=Det[{{ x^2+ y^2, x, y,1},
        {x1^2+y1^2,x1,y1,1},
        {x2^2+y2^2,x2,y2,1},
        {x3^2+y3^2,x3,y3,1}];
Quadratic2D[eqn,{x,y}] ;;
```

### **Quadratic Through Five Points**

Quadratic2D[*point*, *point*, *point*, *point*] ■ Constructs a quadratic that passes through five points.

```
Quadratic2D[Point2D[{x1_,y1_}],Point2D[{x2_,y2_}],Point2D[{x3_,y3_}],
           Point2D[{x4_,y4_}], Point2D[{x5_,y5_}]] :=
  Module[{full,a,b,c,d,e,f},
     full={{x1^2, x2^2, x3^2, x4^2, x5^2},
            {x1*y1,x2*y2,x3*y3,x4*y4,x5*y5},
            {y1^2, y2^2, y3^2, y4^2, y5^2},
            \{x1, x2, x3, x4, x5\},\
                y2,
                            у4,
                       γЗ,
                                    y5},
            {y1,
                                    1}};
                  1,
                        1,
                             1,
            {1,
      {a,b,c,d,e,f}=Map[Det[Transpose[Drop[full,{#}]]]&,{1,2,3,4,5,6}];
     Quadratic2D[a,-b,c,-d,e,-f] ];
```

### **Quadratic Tangent to Five Lines**

Quadratic2D[line, line, line, line, line]  $\blacksquare$  Constructs a quadratic tangent to five lines. The private function Reciprocal\$2D constructs the reciprocal of a conic with respect to the unit circle  $x^2 + y^2 = 1$ . If any of the lines pass through the origin, the entire configuration is transformed to avoid the infinities involved.

### **Quadratic from Two Lines**

Quadratic2D[*line*, *line*] ■ Constructs a quadratic representing two lines multiplied together.

```
Quadratic2D[Line2D[a1_,b1_,c1_],Line2D[a2_,b2_,c2_]] :=
Quadratic2D[a1*a2,a1*b2+a2*b1,b1*b2,a1*c2+a2*c1,b1*c2+b2*c1,c1*c2];
```

### Quadratic from Focus/Directrix/Eccentricity

Quadratic2D[*point*, *line*, *e*] Constructs a quadratic from a focus point, directrix line and eccentricity.

### **Quadratic Vertex Equation**

Quadratic2D[*point*, *fcLen*, *e*,  $\theta$ ]  $\blacksquare$  Constructs a quadratic from the vertex point, focal chord length, eccentricity and rotation angle. If the rotation angle is omitted, it defaults to zero.

```
Quadratic2D::invLen=
    "A non-positive focal chord length, `1`, is invalid; no valid quadratic
can be constructed.";
Quadratic2D::invEcc=
    "A negative eccentricity, `1`, is invalid; no valid quadratic can be
constructed.";
Quadratic2D[P1:Point2D[{x1_,y1_}],fcLen_?IsScalar2D,e_?IsScalar2D] :=
    Quadratic2D[P1,fcLen,e,0];
```

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DQuadratic2D`" *)
```

# D2DSegment2D

The package D2DSegment2D implements the Segment2D object.

# Initialization

```
BeginPackage["D2DSegment2D`", {"D2DCircle2D`", "D2DExpressions2D`",
"D2DGeometry2D`", "D2DLine2D`", "D2DMaster2D`", "D2DNumbers2D`",
"D2DPoint2D`", "D2DSketch2D`", "D2DTransform2D`"}];
D2DSegment2D::usage=
   "D2DSegment2D is a package providing support for line segments.";
Length2D::usage=
   "Length2D[lnseg] computes the length of a line segment.";
Segment2D::usage=
   "Segment2D[{x0,y0},{x1,y1}] is the standard form of a line segment with
end points {x0,y0} and {x1,y1}.";
Begin["`Private`"];
```

# Description

### Representation

Segment2D[ $\{x_0, y_0\}$ ,  $\{x_1, y_1\}$ ]  $\blacksquare$  Standard representation of a line segment in Descarta2D. The coordinates  $\{x_0, y_0\}$  and  $\{x_1, y_1\}$  are start and end points, respectively, of the line segment.

### Evaluation

Segment2D[{ $x_0, y_0$ }, { $x_1, y_1$ }][t]  $\blacksquare$  Evaluates a parameter,  $-\infty < t < \infty$ , on a line segment. The parameter values 0 and 1 are the start and end points, respectively. Returns a coordinate list {x, y}.

```
Segment2D[ {x0_,y0_}, {x1_,y1_}][t_?IsScalar2D] :=
    {x0+t*(x1-x0), y0+t*(y1-y0)};
```

#### Graphics

Provides graphics primitives for a line segment by extending the *Mathematica* Display command. Executed when the package is loaded.

```
SetDisplay2D[
Segment2D[{x0_,y0_},{x1_,y1_}][{t1_?IsScalar2D,t2_?IsScalar2D}],
Line[{Segment2D[{x0,y0},{x1,y1}][t1],
Segment2D[{x0,y0},{x1,y1}][t2]}] ];
SetDisplay2D[
Segment2D[{x0_,y0_},{x1_,y1_}],
Line[{{x0,y0},{x1,y1}}] ];
```

#### Validation

Segment2D[ $\{x_0, y_0\}, \{x_1, y_1\}$ ]  $\blacksquare$  Detects line segments with imaginary arguments and returns the **\$Failed** symbol. If the imaginary parts are insignificant, they are removed.

```
Segment2D::imaginary=
    "An invalid line segment of the form Segment2D['1','2'] has been
detected; the arguments cannot be imaginary.";
Segment2D[{x0_,y0_},{x1_,y1_}] :=
    (Segment2D @@ ChopImaginary2D[Segment$2D[{x0,y0},{x1,y1}]]) /;
(FreeQ[{x0,y0,x1,y1},_Pattern] && IsTinyImaginary2D[{x0,y0,x1,y1}]);
Segment2D[{x0_,y0_},{x1_,y1_}] :=
    (Message[Segment2D::imaginary,{x0,y0},{x1,y1}];$Failed) /;
(FreeQ[{x0,y0,x1,y1},_Pattern] && IsComplex2D[{x0,y0,x1,y1},0]);
```

Segment2D[ $\{x_0, y_0\}$ ,  $\{x_1, y_1\}$ ]  $\blacksquare$  Returns the **\$Failed** symbol for line segments with coincident start and end points.

```
Segment2D::invalid=
    "An invalid line segment of the form Segment2D['1','2'] has been
detected; the defining coordinates cannot be coincident.";
Segment2D[{x0_,y0_},{x1_,y1_}] :=
    (Message[Segment2D::invalid,{x0,y0},{x1,y1}];$Failed) /;
(FreeQ[{x0,y0,x1,y1},_Pattern] && IsCoincident2D[{x0,y0},{x1,y1}]);
```

**IsValid2D**[*lnseg*] ■ Verifies that a line segment is syntactically valid.

```
IsValid2D[
Segment2D[{x0_?IsScalar2D,y0_?IsScalar2D},
{x1_?IsScalar2D,y1_?IsScalar2D}]] := True;
```

# Scalars

### Length

Length2D[Segment2D[{x0\_,y0\_},{x1\_,y1\_}]] := Sqrt[(x0-x1)^2+(y0-y1)^2];

## Slope

 $Slope2D[lnseg] \blacksquare$  Computes the slope of a line segment.

```
Slope2D[Segment2D[{x0_,y0_}, {x1_,y1_}]] :=
If[IsZero2D[x1-x0], Infinity, (y1-y0)/(x1-x0)];
```

# Transformations

### Reflect

**Reflect2D**[*lnseg*, *line*] ■ Reflects a line segment in a line.

```
Reflect2D[Segment2D[{x0_,y0_}, {x1_,y1_}],L2:Line2D[a_,b_,c_]] :=
Segment2D[Reflect2D[{x0,y0},L2],Reflect2D[{x1,y1},L2]];
```

### Rotate

Rotate2D[*lnseg*,  $\theta$ , *coords*]  $\blacksquare$  Rotates a line segment by an angle  $\theta$  about a position specified by a coordinate list. If the third argument is omitted it defaults to the origin (see D2DTransform2D.nb).

#### Scale

Scale2D[*lnseg*, *s*, *coords*]  $\blacksquare$  Scales a line segment from a position given by coordinates. If the third argument is omitted it defaults to the origin (see D2DTransform2D.nb).

### Translate

Translate2D[lnseg, {u, v}]  $\blacksquare$  Translates a line segment delta distance.

```
Translate2D[Segment2D[{x0_,y0_}, {x1_,y1_}], {u_?IsScalar2D, v_?IsScalar2D}]
:=
    Segment2D[{x0+u,y0+v}, {x1+u,y1+v}];
```

# **Point Construction**

### Midpoint

Point2D[lnseg] Constructs the midpoint of a line segment.

```
Point2D[Segment2D[{x0_,y0_},{x1_,y1_}]] := Point2D[{x0+x1,y0+y1}/2];
```

# Line Segment Construction

### Line Segment from Two Points

Segment2D[point, point] ■ Constructs a line segment from two points.

```
Segment2D[Point2D[{x0,y0}],Point2D[{x1,y1}]] :=
Segment2D[{x0,y0},{x1,y1}];
```

# Line Construction

#### Line from Line Segment

Line2D[*lnseg*] Constructs a line containing a line segment.

```
Line2D[Segment2D[{x0_,y0_}, {x1_,y1_}]] :=
Line2D[-(y1-y0), (x1-x0), (x0*y1-x1*y0)];
```

#### Line Bisecting a Line Segment

Line2D[*lnseg*, Perpendicular2D] Constructs a line that is the perpendicular bisector of a line segment.

```
Line2D[Segment2D[{x0_,y0_}, {x1_,y1_}], Perpendicular2D] :=
Line2D[2*(x1-x0), 2*(y1-y0), x0^2-x1^2+y0^2-y1^2];
```

# **Circle Construction**

## **Circle from Diameter Chord**

 $Circle2D[lnseg] \blacksquare$  Constructs a circle from a line segment that is one of the circle's diameter chords.

```
Circle2D[Segment2D[{x0_,y0_}, {x1_,y1_}]] :=
Circle2D[{(x0+x1)/2,(y0+y1)/2},Sqrt[(x0-x1)^2+(y0-y1)^2]/2];
```

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DSegment2D`" *)
```

# D2DSketch2D

The package D2DSketch2D provides the Sketch2D command which is the basic command for plotting *Descarta2D* objects.

# Initialization

```
BeginPackage["D2DSketch2D`", {"D2DExpressions2D`", "D2DMaster2D`"}];
D2DSketch2D::usage=
   "D2DSketch2D is a package providing sketching functions.";
AskCurveLength2D::usage=
   "AskCurveLength2D[ ] returns the value of the option CurveLength2D for
the Sketch2D function (the approximate sketched length of an unbounded
curve).";
CurveLength2D::usage=
   "CurveLength2D->n is an option for the Sketch2D function to specify the
approximate sketched length of an unbounded curve.";
CurveLimits2D::usage=
   "CurveLimits2D[{x,y},curve] returns a list of two parameter values,
\{-t,t\}, so that the distance the point on the unbounded curve at the
parameter 't' is a distance CurveLength2D/2 from the given coordinates.";
IsDisplay2D::usage=
   "IsDisplay2D[object] returns 'True' if the object can be displayed.";
MakePrimitives2D::usage=
  "MakePrimitives2D[object, {t1,t2}] returns a list of graphics primitives
between a pair of parameters for a parametrically defined curve.";
SetDisplay2D::usage=
   "SetDisplay2D[objPatt,objPrim] modifies the Display command to allow
plotting of a new object.";
Sketch2D::usage=
   "Sketch2D[objList,opts] sketches a list of geometric objects.";
Begin["'Private'"];
```

# Utilities

### **Filter Options**

The private function FilterOptions\$2D filters a list of options and provides a Sequence of valid options for the specified command.

```
FilterOptions$2D[command_Symbol,opts___] :=
   Module[{keywords = First /@ Options[command]},
        Sequence @@ Select[{opts},MemberQ[keywords,First[#]]&] ];
```

# Plotting

### Set Display

SetDisplay2D[*objPatt*, *objPrim*] ■ Modifies the *Mathematica* Display command to enable plotting of a new object. The argument objPatt is a pattern which matches the standard form of the object used in *Descarta2D*; the argument objPrim provides the commands that generate the primitives required to plot the object.

```
SetAttributes[SetDisplay2D,HoldAll];
SetDisplay2D[objPatt_,objPrim_] :=
Module[{protected},
protected=Unprotect[Display];
Display[ch_,prim_?(!FreeQ[#,objPatt]&),format__] :=
Display[ch,prim /. {objPatt :> objPrim},format];
Protect[Evaluate[protected]];
IsDisplay2D[obj:objPatt] :=
IsValid2D[obj /. h_[a__][t__]->h[a]] &&
IsNumeric2D[obj /. h_[a__][t__]->h[a]] &&
IsNumeric2D[obj /. h_[a__][t__]->{t}];
Null];
```

## **Display Query**

 $IsDisplay2D[object] \blacksquare$  Returns True if the object can be displayed and has parameters that can be evaluated to real numbers; otherwise, returns False. The function SetDisplay2D, above, provides the implementation of IsDisplay2D for each object after its display graphics are defined.

IsDisplay2D[\_\_\_] := False;

#### **Curve Length**

CurveLength2D-> $n \blacksquare$  The option CurveLength2D of the Sketch2D command specifies the plotted length of an infinite curve and is measured from the midpoint to one of the plotted end points of the curve.

```
Sketch2D::invalidLength=
   "Option CurveLength2D->`1` is invalid; 'CurveLength2D' must be positive;
the current value of CurveLength2D->`2` will be retained.";
protected=Unprotect[SetOptions];
SetOptions[Sketch2D,opts1___,CurveLength2D->n_,opts2___] :=
   Message[Sketch2D::invalidLength,n,AskCurveLength2D[ ]] /;
(IsZeroOrNegative2D[n] || !IsReal2D[n]);
Protect[Evaluate[protected]];
```

AskCurveLength2D[] ■ Returns the value of the Sketch2D command CurveLength2D option.

```
AskCurveLength2D[ ] := Options[Sketch2D,CurveLength2D][[1,2]];
```

#### **Curve Parameter Limits**

CurveLimits2D[ $\{x, y\}$ , curve] Returns a list of two parameter values  $\{-t, t\}$  on an unbounded curve such that the points at the parameter values on the curve are at a distance CurveLength2D/2 from the given base-point coordinates. This is a numerical function used to support plotting, and, therefore, requires numerical arguments.

```
CurveLimits2D[p0:{x0_,y0_},crv_?IsValid2D] :=
Module[{xt,yt,t,eqn,root},
        {xt,yt}=crv[t];
        eqn=Sqrt[(xt-x0)^2+(yt-y0)^2]==AskCurveLength2D[]/2;
        root=FindRoot[Evaluate[eqn],{t,1}];
        {-t,t} /. root[[1]] ] /;
IsNumeric2D[{p0,crv},CurveLimits2D];
```

### **Make Graphics Primitives**

MakePrimitives2D[curve,  $\{t_1, t_2\}$ ] Provides graphics primitives for a parametrically defined curve between two parameters.

```
MakePrimitives2D[crv_?IsValid2D, {t1_?IsScalar2D,t2_?IsScalar2D}] :=
Module[{saveMsg,t,parPlot},
saveMsg=Head[ParametricPlot::ppcom];Off[ParametricPlot::ppcom];
parPlot=ParametricPlot[crv[t] //Evaluate,{t,t1,t2},
DisplayFunction->Identity];
If[saveMsg===String,On[ParametricPlot::ppcom]];
parPlot[[1,1,1]] ];
```

#### Sketch

Sketch2D[*objList*, *opts*] ■ Plots a list of *Descarta2D* objects. The options may be any options supported by the *Mathematica* Graphics command. The list is flattened before it is plotted.

Sketch2D::noObj="No valid objects to sketch.";

```
Sketch2D::notReal=
   "<`1`> object(s) cannot be sketched.";
Options[Sketch2D] =
   {Axes->True,
   Frame->True,
   AspectRatio->Automatic,
   PlotRange->Automatic,
   CurveLength2D->10};
Sketch2D[obj_List,opts___?OptionQ] :=
  Module[{sketchOptsList,inputOptsList,allOptsList,
          grOptsSequence,realObj,n,grafix},
      sketchOptsList=Options[Sketch2D];
      inputOptsList=Flatten[{opts}];
     SetOptions[Sketch2D,
                FilterOptions$2D[Sketch2D,
                                  Sequence @@ inputOptsList]];
     allOptsList=Flatten[Join[inputOptsList,sketchOptsList]];
     grOptsSequence=FilterOptions$2D[Graphics, Sequence @@ allOptsList];
     realObj=Select[Flatten[obj],IsDisplay2D] //N;
      If[(n=Length[Flatten[obj]]-Length[realObj])>0,
         Message[Sketch2D::notReal,n]];
      grafix=If[Length[realObj]>0,
                Show[Graphics[realObj,grOptsSequence]],
                Message[Sketch2D::noObj];Null];
      SetOptions[Sketch2D,Sequence @@ sketchOptsList];
     grafix ];
```

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DSketch2D`" *)
```

# D2DSolve2D

The package D2DSolve2D provides the Solve2D function which is a specialized version of the *Mathematica* Solve and NSolve commands.

# Initialization

```
BeginPackage["D2DSolve2D`", {"D2DExpressions2D`"}];
D2DSolve2D::usage=
   "D2DSolve2D is a package for solving equations.";
MaxSeconds2D::usage=
   "MaxSeconds2D is an option of the Solve2D function that time constrains
the solution of the equations.";
Solve2D::usage=
   "Solve2D[eqns,vars] solves a list of equations for variables in given in
a list.";
Options[Solve2D]=
   {MaxSeconds2D->30};
Begin["`Private`"];
```

# **Symbol Queries**

### Single Symbol Query

The private function IsSymbol\$2D[*expr*, *symbol*] returns True if the expression contains a given symbol; otherwise, returns False.

```
IsSymbol$2D[expr_,sym_Symbol] := MemberQ[Level[expr, {-1}],sym];
```

### Symbol List Query

The private function IsSymbol\$2D[*expr*, *symbolList*] returns True if the expression contains any of the symbols in a list; otherwise, returns False.

```
IsSymbol$2D[expr_,sym_List] := Or @@ Map[IsSymbol$2D[expr,#]&,sym];
```

## Solve

#### **Maximum Seconds Option**

 $MaxSeconds2D \rightarrow n \square$  The option MaxSeconds2D specifies the maximum number of seconds allowed to solve equations using Solve2D. The private function AskMaxSeconds2D returns the current setting for MaxSeconds2D.

```
Solve2D::invalidTime=
   "Option MaxSeconds2D->`1` is invalid; 'MaxSeconds2D' must be positive;
the current value of MaxSeconds2D->`2` will be retained.";
protected=Unprotect[SetOptions];
SetOptions[Solve2D,opts1___,MaxSeconds2D->n_,opts2___] :=
   Message[Solve2D::invalidTime,n,AskMaxSeconds$2D[ ]] /;
(IsZeroOrNegative2D[n] || !IsReal2D[n]);
Protect[Evaluate[protected]];
AskMaxSeconds$2D[ ] := Options[Solve2D,MaxSeconds2D][[1,2]];
```

#### Solve

 $Solve2D[eqnList, varsList, opts] \blacksquare$  Solves a list of equations for a list of variables. Uses the Mathematica function NSolve if any real numbers are involved, or if the Mathematica function N is in the evaluation stack; otherwise, uses the Mathematica function Solve. An empty list is returned (and a warning message output) if the equations cannot be solved in the number of seconds specified by the option MaxSeconds2D->n.

```
Solve2D::infinite=
   "An infinite number of solutions exist; only independent solutions will
be returned. ";
Solve2D::time=
   "The equations could not be solved in MaxSeconds2D->'1', an empty list
of solutions will be returned; using approximate numbers may produce a more
complete list of solutions.";
SimplifyEquation$2D[eqn_Equal] :=
   (eqn[[1]]//ExpandAll)==(eqn[[2]]//ExpandAll);
Solve2D[eqns:{HoldPattern[Equal[_,_]..]},
        vars:{_Symbol..},
        MaxSeconds2D->secs ] :=
  Module[{save,result},
     save=AskMaxSeconds$2D[ ];
      SetOptions[Solve2D,MaxSeconds2D->secs];
      result=Solve2D[eqns,vars];
      SetOptions[Solve2D,MaxSeconds2D->save];
      result ];
```

```
Solve2D[eqns:{HoldPattern[Equal[_,_]..]},vars:{_Symbol..}] :=
Module[{save,ans},
save=Head[Solve::svars];Off[Solve::svars];
ans=TimeConstrained[
If[IsApproximate2D[eqns],
NSolve[Map[SimplifyEquation$2D,eqns]//N,vars,
WorkingPrecision->$MachinePrecision],
Solve[Map[SimplifyEquation$2D,eqns],vars]],
AskMaxSecond$2D[],
Message[Solve2D::time,AskMaxSecond$2D[]];{}];
If[save===String,On[Solve::svars]];
If[IsSymbol$2D[Map[(vars /. #)&,ans],vars],
Message[Solve2D::infinite];
Select[ans,Not[IsSymbol$2D[(vars /. #),vars]]&],
ans]];
```

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DSolve2D`" *)
```

# D2DTangentCircles2D

The package D2DTangentCircles2D provides a variety of functions for constructing tangent circles that satisfy three conditions. The conditions include passing through a given point, center at a given point or on a given curve, tangent to a given line and tangent to a given circle.

# Initialization

```
BeginPackage["D2DTangentCircles2D`", {"D2DCircle2D`", "D2DExpressions2D`",
"D2DGeometry2D`", "D2DLine2D`", "D2DMaster2D`", "D2DPoint2D`",
"D2DSolve2D`"}];
D2DTangentCircles2D::usage=
    "D2DTangentCircles2D is a package for constructing tangent circles.";
TangentCircles2D::usage=
    "TangentCircles2D[objList,(center),(radius)] constructs a list of
circles tangent to one, two or three objects (points, lines or circles);
optionally, the center may be constrained to a point, line or circle;
optionally, the radius may be specified; the total number of constraints
must be three (constraining the center to a point is two constraints).";
Begin["'Private`"];
```

# Queries

### Point, Line or Circle Query

The private function IsSimple\$2D returns True if an object is a point, line or circle; otherwise, returns False.

IsSimple\$2D[obj\_] := Is2D[obj,{Point2D,Line2D,Circle2D}];

### Line or Circle Query

The private function IsSimpleCurve\$2D returns True if an object is a line or a circle; otherwise, returns False.

```
IsSimpleCurve$2D[obj_] := Is2D[obj,{Line2D,Circle2D}];
```

# Tangent/On Equations

### Point Tangent to a Circle (On Circle)

The private function TangentEquation\$2D returns an equation constraining a point to be on a circle.

```
TangentEquation$2D[Point2D[{x1_,y1_}],Circle2D[{h2_,k2_},r2_]] :=
(x1-h2)^2+(y1-k2)^2==r2^2;
```

#### Line Tangent to a Circle

The private function TangentEquation\$2D returns an equation constraining a line to be tangent to a circle.

```
TangentEquation$2D[Line2D[a1_,b1_,c1_],Circle2D[{h2_,k2_},r2_]] :=
(a1^2+b1^2)*r2^2==(a1*h2+b1*k2+c1)^2;
```

#### Circle Tangent to a Circle

The private function TangentEquation\$2D returns an equation constraining two circles to be tangent.

```
TangentEquation$2D[Circle2D[{h1_,k1_},r1_],Circle2D[{h2_,k2_},r2_]] :=
  ((h1-h2)^2+(k1-k2)^2-(r1-r2)^2)*((h1-h2)^2+(k1-k2)^2-(r1+r2)^2) == 0;
```

### Point on a Point

The private function **OnEquation\$2D** returns a pair of equations constraining two points to be coincident.

```
\label{eq:definition} \end{tabular} \\ \texttt{OnEquation$2D[\{x1_,y1_\},\texttt{Point2D[\{x2_,y2_\}\}]]} := \{x1 = x2, \ y1 = y2\}; \\ \end{aligned}
```

#### Point on a Line

The private function OnEquation\$2D returns equation constraining a point to be on a line.

```
\label{eq:definition} \\ \texttt{OnEquation$2D[{x1_,y1_}, \texttt{Line2D[a2_,b2_,c2_]]} := \{\texttt{a2*x1+b2*y1+c2==0}\}; \\
```

#### Point on a Circle

The private function  ${\tt OnEquation\$2D}$  returns an equation constraining a point to be on a circle.

```
OnEquation$2D[\{x1_, y1_\},Circle2D[\{h2_, k2_\},r2_]] := { (x1-h2)^2+(y1-k2)^2==r2^2};
```

# **General Circle Tangency**

### **Tangent Circles**

The private function TangentCircles\$2D is a general function that constructs a list of circles tangent to a list of one, two or three objects, optionally with center on a given object, optionally with a given radius.

```
TangentCircles$2D[obj_List,cenObj_,radius_] :=
Module[{h,k,r,c1,eq1,eq2,eq3,ans,circles},
c1=Circle2D[{h,k},r];
eq1=Map[TangentEquation$2D[#,c1]&,obj];
eq2=If[cenObj===Nul1,{},OnEquation$2D[{h,k},cenObj]];
eq3=If[radius==Nul1,{},{r==radius}];
ans=Solve2D[Join[eq1,eq2,eq3],{h,k,r}];
ans=Select[ans,Not[IsComplex2D[{h,k,r} / . #]]&];
ans=Select[ans,Not[IsZeroOrNegative2D[r /. #]]&];
circles=Map[(c1 /. #)&, ans];
Complement[Union[circles],obj] ];
```

# **Tangent Circle Construction**

#### Tangent Object, Center Point

**TangentCircles2D**[ $\{pt | ln | cir\}$ , *point*]  $\blacksquare$  Constructs a list of circles tangent to a point, line or circle and passing through a point.

```
TangentCircles2D[{obj?IsSimple$2D}, P:Point2D[{x_,y_}]] :=
TangentCircles$2D[{obj}, P,Null];
```

## Tangent Object, Center on Object, Radius

TangentCircles2D[ $\{pt | ln | cir\}$ , ln | cir, r]  $\blacksquare$  Constructs a list of circles tangent to a point, line or circle, with center point on a line or circle and with a given radius.

## Two Tangent Objects, Center On Object

TangentCircles2D[ $\{pt | ln | cir, pt | ln | cir \}$ , ln | cir ] Constructs a list of circles tangent to two objects (points, lines or circles) centered on a line or circle.

#### Two Tangent Objects, Radius

TangentCircles2D[ $\{pt | ln | cir, point | ln | cir \}$ , r]  $\blacksquare$  Constructs a list of circles tangent to two objects (points, lines or circles) with a given radius.

# **Three Tangent Objects**

TangentCircles2D[ $\{pt | ln | cir, pt | ln | cir, pt | ln | cir \}$ ]  $\blacksquare$  Constructs a list of circles tangent to three objects (points, lines or circles).

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DTangentCircles2D`" *)
```
# D2DTangentConics2D

The package D2DTangentConics2D provides functions for constructing conics and quadratics that satisfy five conditions. Each condition may be either passing through a given point or tangent to a given line.

## Initialization

```
BeginPackage["D2DTangentConics2D`", {"D2DCircle2D`", "D2DEllipse2D`",
"D2DExpressions2D'", "D2DGeometry2D'", "D2DHyperbola2D'", "D2DLine2D'",
"D2DLoci2D'", "D2DMaster2D'", "D2DParabola2D'", "D2DPencil2D'",
"D2DPoint2D'", "D2DQuadratic2D'", "D2DSolve2D'", "D2DTransform2D'"}];
D2DTangentConics2D::usage=
   "D2DTangentConics2D is a package for constructing tangent conics and
quadratics.";
TangentConics2D::usage=
   "TangentConics2D[{obj1,obj2,obj3,obj4,obj5}] constructs list of conic
curves given five objects; the objects may be any combination of points and
lines; the conics will pass through the given points and be tangent to the
given lines.";
TangentOuadratics2D::usage=
   "TangentQuadratics2D[{obj1,obj2,obj3,obj4,obj5}] constructs list of
quadratics given five objects; the objects may be any combination of points
and lines; the quadratics will pass through the given points and be tangent
to the given lines.";
```

```
Begin["'Private'"];
```

## **Error Messages**

#### **General Error Messages**

```
TangentConics2D::coincident=
    "Two or more of the defining points or lines are coincident; no proper
conic can be constructed.";
```

```
TangentConics2D::collinear=
    "Three or more of the defining points are collinear; no proper conic can
be constructed.";
TangentConics2D::concurrent=
    "Three or more of the tangent lines are concurrent; no proper conic can
be constructed.";
TangentConics2D::linesThru=
    "One of the points is on more than one of the tangent lines; no proper
conic can be constructed.";
TangentConics2D::parallel=
    "Three or more of the defining lines are parallel; no proper conic can
be constructed.";
TangentConics2D::pointsOn=
    "Two or more of the points are on a tangent line; no proper conic can be
constructed.";
```

## Utilities

#### **Numeric Computations**

The private function N\$2D numerically normalizes lines and quadratics (or lists of such objects) if approximate numerical computations are underway; otherwise, no action is taken.

```
N$2D[expr_List] := Map[N$2D,expr];
N$2D[L:Line2D[a_,b_,c_]] :=
If[IsApproximate2D[L],Line2D[ N[L] ],L];
N$2D[P:Point2D[{x_,Y_}]] :=
If[IsApproximate2D[P],N[P],P];
N$2D[Q:Quadratic2D[a_,b_,c_,d_,e_,f_]] :=
If[IsApproximate2D[Q],Quadratic2D[ N[Q] ],Q];
```

#### Number of Points on a Line

The private function CountPointsOn\$2D returns the number of points from a given list that are on a given line.

```
CountPointsOn$2D[pts_List,L:Line2D[a_,b_,c_]] :=
Count[Map[ISOn2D[#,L]&, pts], True];
```

The private function MaxPointsOn\$2D returns the maximum number of points from a given list that are on any of the lines in a list.

```
MaxPointsOn$2D[pts_List,lns_List] :=
  If[Length[pts]<1 || Length[lns]<1,
    0,
    Max @@ Map[CountPointsOn$2D[pts,#]&,lns] ];</pre>
```

#### Number of Lines Through a Point

The private function CountLinesThru\$2D returns the number of lines from a given list that pass through a given point.

```
CountLinesThru$2D[lns_List,P:Point2D[{x_,y_}]] :=
Count[Map[IsOn2D[P,#]&, lns], True];
```

The private function MaxLinesThru\$2D returns the maximum number of lines from a given list that pass through any of the points in a list.

```
MaxLinesThru$2D[lns_List,pts_List] :=
    If[Length[lns]<1 || Length[pts]<1,
        0,
        Max @@ Map[CountLinesThru$2D[lns,#]&,pts] ];</pre>
```

#### Validity Queries

The private function ValidObjects\$2D verifies that the object list contains valid objects. The function private ValidConfigurationQ\$2D verifies that the configuration of the objects is valid.

```
ValidObjectsQ$2D[obj_List,funcName_] :=
   ((Count[Map[IsValid2D,obj],True]==
    Count[Map[Is2D[#, {Point2D,Line2D}]&,obj],True]==
    Length[obj]==5) &&
    IsNumeric2D[obj,funcName]);
ValidConfigurationQ$2D[obj_List] :=
   Module[{pts,lns},
     pts=Select[N$2D[obj], Is2D[#, {Point2D}]&];
      lns=Select[N$2D[obj],Is2D[#,{Line2D}]&];
     Which
         IsCoincident2D[pts],
           Message[TangentConics2D::coincident];False,
         IsCoincident2D[lns],
           Message[TangentConics2D::coincident];False,
         IsCollinear2D[pts],
            Message[TangentConics2D::collinear];False,
         IsConcurrent2D[lns],
            Message[TangentConics2D::concurrent];False,
         IsTripleParallel2D[lns],
           Message[TangentConics2D::parallel];False,
         MaxPointsOn$2D[pts,lns]>1,
            Message[TangentConics2D::pointsOn];False,
         MaxLinesThru$2D[lns,pts]>1,
            Message[TangentConics2D::linesThru];False,
         True,
            True] ];
```

## **Polynomials**

#### Point on Line

The private function Polynomial\$2D forms a polynomial by substituting the coordinates of a point into the equation of a line.

```
\label{eq:polynomial} \end{tabular} Polynomial \end{tabular} 2D[\end{tabular} Polynomial \end{tabular} 2D[\end{tabular} Polynomial \end{tabular} 2D[\end{tabular} A \end{tabular} A \end{tabular} A \end{tabular} A \end{tabular} \end{tabular} A \end{tabul
```

#### **Point on Quadratic**

The private function Polynomial\$2D forms a polynomial by substituting the coordinates of a point into a quadratic equation.

#### Line Tangent to Quadratic

The private function Polynomial\$2D forms a polynomial of coefficients from a line and a quadratic when the line is tangent to the quadratic.

```
Polynomial$2D[Line2D[p_,q_,r_],Quadratic2D[a_,b_,c_,d_,e_,f_]] :=
  ((4*c*f-e^2)*p^2+(4*a*f-d^2)*q^2+(4*a*c-b^2)*r^2+
  2*(b*d-2*a*e)*q*r+2*(b*e-2*c*d)*p*r+2*(d*e-2*b*f)*p*q);
```

## **Quadratic and Conic Construction**

#### **Quadratic Tangent to Five Objects**

TangentQuadratics2D[ $\{obj_1, obj_2, obj_3, obj_4, obj_5\}$ ]  $\blacksquare$  Constructs a list of conics tangent to five objects. The objects may be any combination of points or lines.

```
TangentQuadratics2D[obj_List] :=
    If[ValidConfigurationQ$2D[obj],
        TangentQuadratic$2D[obj//N$2D],
        {}] /;
ValidObjectsQ$2D[obj,TangentQuadratics2D];
```

#### **Conic Tangent to Five Objects**

TangentConics2D[ $\{obj_1, obj_2, obj_3, obj_4, obj_5\}$ ]  $\blacksquare$  Constructs a list of conics tangent to five objects. The objects may be any combination of points or lines.

```
TangentConics2D[obj_List] :=
Module[{Q,conics},
If[ValidConfigurationQ$2D[obj],
Q=TangentQuadratics2D[obj//N$2D];
conics=Flatten[Map[Loci2D,Q]];
Union[
Select[conics,
Is2D[#,{Circle2D,Ellipse2D,Hyperbola2D,Parabola2D}]&]],
{}] ] /;
ValidObjectsQ$2D[obj,TangentConics2D];
```

#### **Preprocess Arguments**

Preprocesses the arguments to private function TangentQuadratic\$2D to match the implemented functions.

```
TangentQuadratic$2D[{a_,b_,c_,d_,e_}] :=
TangentQuadratic$2D[a,b,c,d,e];
TangentQuadratic$2D[a__,L1_Line2D,b__,L2_Line2D,c__,L3_Line2D,d__] :=
TangentInverse$2D[{L1,L2,L3,a,b,c,d}];
TangentQuadratic$2D[a__,L_Line2D,b__,P_Point2D,c__] :=
TangentQuadratic$2D[a,P,b,c,L];
TangentQuadratic$2D[a__,P_Point2D,b__,L_Line2D,c__] :=
(TangentQuadratic$2D[a__,P_Point2D,b__,L_Line2D,c__] :=
(TangentQuadratic$2D[{P,L},a,b,c]) /;
IsOn2D[P,L];
```

#### **Five Points**

Private function that constructs a list containing one quadratic passing through five points.

```
TangentQuadratic$2D[P1_,P2_,P3_,P4_,P5_] :=
{Quadratic2D[P1,P2,P3,P4,P5] //N$2D};
```

#### Four Points, One Line (No Points on Line)

Private function that constructs a list containing two quadratics passing through four points and tangent to one line. None of the points can be on the tangent line.

```
TangentQuadratic$2D[P1_Point2D,P2_Point2D,P3_Point2D,P4_Point2D,
L5_Line2D] :=
Module[{Q,k,allRoots,realRoots},
Q=Quadratic2D[{Line2D[P1,P2],Line2D[P3,P4]},
{Line2D[P1,P3],Line2D[P2,P4]},k,Pencil2D] //N$2D;
allRoots=Solve2D[{P0lynomial$2D[L5,Q]==0},{k}];
realRoots=Select[allRoots,IsReal2D[k /. #]&];
N$2D[Map[(Q /. #)&, realRoots]] ];
```

#### Four Points, One Line (One Point on Line)

Private function that constructs a list containing one quadratic passing through four points and tangent to one line. One of the points must be on the tangent line.

#### Three Points, Two Lines (No Points on Lines)

Private function that constructs a list containing four quadratics given three points and two tangent lines. None of the points can be on the tangent lines.

```
\label{eq:constraint} \texttt{TangentQuadratic$2D[Point2D[\{0,0\}],Point2D[\{x2\_,y2\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_\}],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[\{x3\_,y3\_]],Point2D[[x3\_,y3\_]],Point2D[[x3\_,y3\_]],Point2D[[x3\_,y3\_]]],Point2D[
                                                                          Line2D[a1_,b1_,c1_],Line2D[a2_,b2_,c2_]] :=
          Module[{p11,p12,p13,p21,p22,p23,p31,p32,p33,a,b,ans,k,Q},
                      pll=cl; pl2=al*x2+bl*y2+cl; pl3=al*x3+bl*y3+cl;
                      p21=c2; p22=a2*x2+b2*y2+c2; p23=a2*x3+b2*y3+c2;
                                                                                                                       p33=a*x3+b*y3+1;
                      p31=1; p32=a*x2+b*y2+1;
                      ans=Solve2D[{p11*p21*p32^2==p12*p22*p31^2,
                                                                     p12*p22*p33^2==p13*p23*p32^2}, {a,b}];
                      ans=Select[ans,(IsReal2D[a /. #] && IsReal2D[b /. #])&];
                      k=c1*c2;
                      Q=(a1*x+b1*y+c1)*(a2*x+b2*y+c2)-k*(a*x+b*y+1)^2;
                      N$2D[Map[Quadratic2D[(Q /. #), {x,y}]&,ans]] ];
TangentQuadratic$2D[P1:Point2D[{x1_,y1_}], P2:Point2D[{x2_,y2_}],
                                                                          P3:Point2D[{x3_,y3_}],
                                                                         L1:Line2D[a1_,b1_,c1_],L2:Line2D[a2_,b2_,c2_]] :=
          Module[{pt2,pt3,ln1,ln2,Q},
                      {pt2,pt3,ln1,ln2}=Translate2D[{P2,P3,L1,L2},{-x1,-y1}] //N$2D;
                      Q=TangentQuadratic$2D[Point2D[{0,0}],pt2,pt3,ln1,ln2];
                      N$2D[Translate2D[Q, {x1,y1}]] ];
```

#### Three Points, Two Lines (One Point on Line)

Private function that constructs a list containing up to two quadratics through three points, tangent to two lines when one of the points is on a tangent line.

#### Three Points, Two Lines (Two Points On Lines)

Private function that constructs a list containing up to one quadratic through three points, tangent to two lines when two of the points are on the tangent lines (one point on each tangent line).

#### **Reciprocal Method**

Private function that constructs a list containing quadratics given five elements (points or tangent lines). The method of reciprocals is used. Using the reciprocal method converts a case with more than two tangent lines to its reciprocal, which has two or fewer tangent lines.

```
TangentInverse$2D[origObjs_List] :=
Module[{offset,objsTrans,invertedObjs,Q},
offset=SaveOffset$2D[origObjs];
objsTrans=Translate2D[origObjs,-offset];
invertedObjs=Map[Invert$2D,objsTrans] //N$2D;
Q=TangentQuadratic$2D[invertedObjs];
Translate2D[Map[Reciprocal$2D,Q],offset] //N$2D ];
```

Private functions that construct the pole point of a line with respect to a unit circle and the polar line of a point with respect to a unit circle.

Invert\$2D[Line2D[a\_,b\_,c\_]] := Point2D[{-a/c,-b/c}];

 $Invert\$2D[Point2D[{x_,y_}]] := Line2D[x,y,-1];$ 

Private function that constructs the reciprocal quadratic of a quadratic with respect to a unit circle.

```
Reciprocal$2D[Quadratic2D[a_,b_,c_,d_,e_,f_]] :=
Quadratic2D[4*c*f-e^2,2*d*e-4*b*f,4*a*f-d^2,
4*c*d-2*b*e,4*a*e-2*d*b,4*a*c-b^2] //N$2D;
```

Private functions that determine an offset that will safely position a list of objects insuring that no line passes through the center of inversion and no point is coincident with the center of inversion.

```
InvalidOffsetQ$2D[P1:Point2D[{x1_,y1_}],offset:{dx_,dy_}] :=
IsCoincident2D[P1,Point2D[offset]];
```

```
InvalidOffsetQ$2D[L1:Line2D[a1_,b1_,c1_],offset:{dx_,dy_}] :=
IsOn2D[Point2D[offset],L1];
SaveOffset$2D[obj_List] :=
Module{{offset={0,0}},
While[MemberQ[Map[InvalidOffsetQ$2D[#,offset]&,obj],
True],
offset={Random[Integer,{-4,4}],
Random[Integer,{-4,4}]};
offset];
```

## Epilogue

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DTangentConics2D`" *)
```

# D2DTangentLines2D

The package D2DTangentLines2D provides functions for computing lines that are tangent to curves using a variety of defining conditions.

## Initialization

```
BeginPackage["D2DTangentLines2D`", {"D2DCircle2D`", "D2DConic2D`",
"D2DEllipse2D'", "D2DEquations2D'", "D2DExpressions2D'", "D2DGeometry2D'",
"D2DHyperbola2D'", "D2DLine2D'", "D2DMaster2D'", "D2DParabola2D'",
"D2DPoint2D'", "D2DQuadratic2D'", "D2DSegment2D'", "D2DSolve2D'",
"D2DTangentPoints2D`", "D2DTransform2D`"}];
D2DTangentLines2D::usage=
  "D2DTangentLines2D is a package for constructing tangent lines and line
segments.";
TangentEquation2D::usage=
   "TangentEquation2D[line, quad] constructs an equation involving the
coefficients of a line and a quadratic constraining the line to be tangent
to the quadratic.";
TangentLines2D::usage=
   "TangentLines2D[point, curve] constructs a list of lines through a point
and tangent to a second-degree curve; TangentLines2D[line, curve,
Parallel2D] constructs a list of tangent lines parallel to a given line;
TangentLines2D[line, curve, Perpendicular2D] constructs a list of tangent
lines perpendicular to a given line; TangentLines2D[curve, curve]
constructs a list of lines tangent to two curves.";
TangentSegments2D::usage=
   "TangentSegments2D[curve,curve] constructs a list of line segments
tangent to two curves.";
Begin["'Private'"];
```

## **Tangent Equation**

#### Line Tangent to a Quadratic

TangentEquation2D[*line*, *quad*] ■ Forms an equation between the coefficients of a line and a quadratic constraining the line to be tangent to the quadratic.

## Line Construction

#### Lines Through a Point Tangent to a Circle

TangentLines2D[*point*, *circle*] ■ Constructs a list containing up to two lines through a point and tangent to a circle.

```
TangentLines2D[Point2D[{x1_y1_}],Circle2D[{h2_,k2_},r2_]] :=
Union[TangentLine$2D[{x1,y1},0,{h2,k2},r2,1]];
```

#### Lines Through a Point Tangent to a Curve

TangentLines2D[*point*, *curve*] ■ Constructs a list containing up to two lines through a point and tangent to a curve or quadratic.

```
TangentLines2D[P1:Point2D[{x1_,y1_}],crv2_] :=
Module[{pts},
    pts=TangentPoints2D[P1,crv2];
    Which[
        pts=={}, {},
        Length[pts]==1, {Line2D[P1,crv2]},
        True, Map[Line2D[P1,#]&,pts]] ] /;
Is2D[crv2,{Ellipse2D,Hyperbola2D,Parabola2D,Quadratic2D}];
```

#### Parallel or Perpendicular Tangent Lines

TangentLines2D[*line*, *curve*, Parallel2D|Perpendicular2D]  $\blacksquare$  Constructs a list containing up to two lines which are parallel or perpendicular to a given line and tangent to a conic curve quadratic. If the Parallel2D|Perpendicular2D keyword is omitted, the default is Parallel2D.

#### D2DTangentLines2D - Line Construction

```
TangentLines2D[L:Line2D[p_,q_,r_],Q:Quadratic2D[a_,b_,c_,d_,e_,f_],
               Parallel2D] :=
  Module[{z,eq1,R,ans},
     z=R*(-b^2+4*a*c)+q*(b*d-2*a*e)+p*(-2*c*d+b*e);
      If[IsZero2D[z],{},
         eq1=TangentEquation2D[Line2D[p,q,R],Q];
         ans=Select[Solve2D[{eq1}, {R}],
                   Not[IsComplex2D[R /. #]]&];
         Map[Line2D[p,q,(R /. #)]&,ans]] ];
TangentLines2D[L:Line2D[a_,b_,c_],crv_,Parallel2D] :=
   TangentLines2D[L,Quadratic2D[crv],Parallel2D] /;
Is2D[crv,{Circle2D,Ellipse2D,Hyperbola2D,Parabola2D}];
TangentLines2D[L:Line2D[a_,b_,c_],crv_] :=
   TangentLines2D[L,crv,Parallel2D] /;
Is2D[crv,{Circle2D,Ellipse2D,Hyperbola2D,Parabola2D,Quadratic2D}];
TangentLines2D[Line2D[a_,b_,c_],crv_,Perpendicular2D] :=
  TangentLines2D[Line2D[-b,a,c],crv,Parallel2D] /;
Is2D[crv,{Circle2D,Ellipse2D,Hyperbola2D,Parabola2D,Quadratic2D}];
```

#### Lines Tangent to Two Circles

**TangentLines2D**[*circle*, *circle*]  $\blacksquare$  Constructs a list containing up to four lines which are tangent to two circles. The first two lines in the list (if present) are the external tangents; the last two lines (if present) are the internal tangents. The private function TangentLine\$2D implements the mathematics.

```
TangentLines2D[Circle2D[{h1_,k1_},r1_],Circle2D[{h2_,k2_},r2_]] :=
Flatten[{Map[Union[TangentLine$2D[{h1,k1},r1,{h2,k2},r2,#]]&,{-1,1}]}];
TangentLine$2D[{h1_,k1_},r1_,{h2_,k2_},r2_,s_] :=
Module[{H=h1-h2,K=k1-k2,R=r1+s*r2,L,A2,B2,C2,lns,sv1,sv2},
L=H^2+K^2;
A2=A1*H-B1*K; B2=B1*H+A1*K; C2 =L*r1-h1*A2-k1*B2;
sv1=Head[Line2D::imaginary];Off[Line2D::imaginary];
sv2=Head[Line2D::invalid];Off[Line2D::invalid];
lns=Map[(Line2D[A2, B2, C2] /. #)&,
{{A1->R, B1-> Sqrt[L-R^2]},
{A1->R, B1->-Sqrt[L-R^2]}];
If[sv1===String,On[Line2D::invalid];
Select[lns,IsValid2D]];
```

#### Lines Tangent to Two Quadratics

TangentLines2D[quad, quad]  $\blacksquare$  Constructs a list containing up to four lines tangent to two quadratics. The private function TanLn\$2D computes the candidate tangent lines, and the private function DeleteCoincident\$2D removes coincident solutions.

```
TanLn$2D[Q1:Quadratic2D[a1_,b1_,c1_,d1_,e1_,f1_],
          Q2:Quadratic2D[a2_,b2_,c2_,d2_,e2_,f2_]] :=
   Module[{L,p,q,r,ans,lns,svMsg1,svMsg2},
      L=Line2D[p,q,r];
      ans=Solve2D[{TangentEquation2D[L,Q1],
                   TangentEquation2D[L,Q2],
                   p^2+q^2==1}, {p,q,r}];
      svMsg1=Head[Line2D::imaginary];Off[Line2D::imaginary];
      svMsg2=Head[Line2D::invalid];Off[Line2D::invalid];
      lns=Map[(L /. #)&,ans];
      If[svMsg1===String,On[Line2D::imaginary]];
      If[svMsg2===String,On[Line2D::invalid]];
      Select[lns,IsValid2D] ];
DeleteCoincident$2D[{s1_
     L1:Line2D[a1_,b1_,c1_],s2_
      L2:Line2D[a2_,b2_,c2_],s3___}]:=
   DeleteCoincident$2D[{s1,L1,s2,s3}] /;
IsCoincident2D[L1,L2];
DeleteCoincident$2D[lns_List]:=lns;
TangentLines2D[Q1:Quadratic2D[a1_,b1_,c1_,d1_,e1_,f1_],
               Q2:Quadratic2D[a2_,b2_,c2_,d2_,e2_,f2_]] :=
   If[IsCoincident2D[Q1,Q2],{},
      DeleteCoincident$2D[TanLn$2D[Q1,Q2]]];
```

#### Lines Tangent to Two Conics

**TangentLines2D**[*curve*, *curve*] ■ Constructs a list containing up to four lines tangent to two conic curves (circles, ellipses, hyperbolas, parabolas or quadratics).

```
TangentLines2D[crv1_,crv2_] :=
Module[{Q1,Q2},
Q1=If[Is2D[crv1,{Quadratic2D}],crv1,Quadratic2D[crv1]];
Q2=If[Is2D[crv2,{Quadratic2D}],crv2,Quadratic2D[crv2]];
TangentLines2D[Q1,Q2]] /;
Is2D[crv1,{Circle2D,Ellipse2D,Hyperbola2D,Parabola2D,Quadratic2D}] &&
Is2D[crv2,{Circle2D,Ellipse2D,Hyperbola2D,Parabola2D,Quadratic2D}];
```

#### Line Segments Tangent to Two Curves

TangentSegments2D[*curve*, *curve*] ■ Constructs a list containing up to four line segments tangent to two curves (circles, ellipses, hyperbolas, parabolas or quadratics).

## **Epilogue**

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DTangentLines2D`" *)
```

# D2DTangentPoints2D

The package D2DTangentPoints2D provides functions for constructing the point of contact between a curve and a tangent line.

## Initialization

```
BeginPackage["D2DTangentPoints2D`", {"D2DCircle2D`", "D2DEllipse2D`",
"D2DExpressions2D`", "D2DGeometry2D`", "D2DHyperbola2D`",
"D2DIntersect2D`", "D2DLine2D`", "D2DMaster2D`", "D2DParabola2D`",
"D2DPoint2D`", "D2DQuadratic2D`"}];
D2DTangentPoints2D::usage=
    "D2DTangentPoints2D is a package for constructing tangent points.";
TangentPoints2D::usage=
    "TangentPoints2D[point,curve] constructs a list containing points that
are the tangency points of the lines from a point to a curve.";
Begin["`Private`"];
```

## **Point Construction**

#### **Circle Contact Points**

TangentPoints2D[*point*, *circle*]  $\blacksquare$  Constructs a list containing up to two points that are the contact points of the lines tangent to a circle from a point. This is a simplified formula for circles; the general second-degree form also produces the correct points (see below).

```
TangentPoints2D[Point2D[{x1_,y1_}],Circle2D[{h_,k_},r_]] :=
Module[{d,R,c,s,S},
d=(x1-h)^2+(y1-k)^2;
If[IsZero2D[d],{},
R=If[IsZero2D[d-r^2],0,d-r^2];
c=(r*(x1-h)-S*Sqrt[R]*(y1-k))/d;
s=(r*(y1-k)+S*Sqrt[R]*(x1-h))/d;
Map[(Point2D[{h+r*c,k+r*s}] /. S->#)&,
Which[IsZero2D[R],{1},
IsNegative2D[R],{},
True, {-1,1}]]];
```

#### **Conic Contact Points**

**TangentPoints2D**[*point*, *curve*]  $\blacksquare$  Constructs a list containing up to two points that are the contact points of the lines tangent to a second-degree curve (ellipse, parabola, hyperbola or quadratic) from a point. The circle is handled as a special case (see above).

```
TangentPoints2D[P1:Point2D[{x1_,y1_}],crv2_] :=
Module[{Q,a,b,c,d,e,f,p,q,r,pts},
Q=If[Head[crv2]===Quadratic2D,crv2,Quadratic2D[crv2]];
{a,b,c,d,e,f}=List @@ Q;
p=2*a*x1+b*y1+d; q=b*x1+2*c*y1+e; r=d*x1+e*y1+2*f;
Which[
IsZero2D[{p,q},And], {},
IsOn2D[P1,Q], {Point2D[Line2D[p,q,r],crv2]},
True, Points2D[Line2D[p,q,r],crv2]] ] /;
Is2D[crv2,{Ellipse2D,Hyperbola2D,Parabola2D,Quadratic2D];
```

## **Epilogue**

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DTangentPoints2D`" *)
```

# D2DTransform2D

The package D2DTransform2D provides the basic support for the transformations provided by *Descarta2D* (reflect, rotate, scale and translate). Each *Descarta2D* object provides specific support for the transformations using these basic capabilities.

## Initialization

```
BeginPackage["D2DTransform2D`", {"D2DExpressions2D`", "D2DLine2D`",
"D2DMaster2D`"}];
D2DTransform2D::usage=
   "D2DTransform2D is a package providing transformations.";
Reflect2D::usage=
   "Reflect2D[obj,line] reflects an object in a line;
Reflect2D[objList,line] reflects a list of objects in a line;
Reflect[eqn, \{x, y\}, line] reflects an equation in variables 'x' and 'y' in a
line.";
ReflectAngle2D::usage=
   "ReflectAngle2D[theta,line] reflects an angle in a line.";
Rotate2D::usage=
   "Rotate2D[obj,theta,\{h,k\}] rotates an object an angle 'theta' about a
point with coordinates {h,k}; Rotate2D[objList,theta,{h,k}] rotates a list
of objects; Rotate2D[eqn,\{x,y\}, \{h,k\}] rotates an equation in variables 'x'
and 'Y'; if the {h,k} coordinates are omitted, the default is \{0,0\}.";
Scale2D::usage=
   "Scale2D[obj,s,{h,k}] scales an object about coordinates {h,k} by scale
factor 's'; Scale2D[objList,s,{h,k}] scales a list of objects;
\label{eq:scale2D[eqn, \{x,y\}, \{h,k\}] scales an equation in variables `x' and `y'; if
the center of scaling \{h,k\} is omitted, the default is \{0,0\}.";
Translate2D::usage=
   "Translate2D[obj,{u,v}] translates an object delta distance \{u,v\};
Translate2D[objList,{u,v}] translates a list of objects;
\label{eq:linear} Translate2D[eqn, \{x,y\}, \{u,v\}] \mbox{ translates an equation in variables } 'x' \mbox{ and }
'y'.";
Begin["'Private'"];
```

## Queries

#### **Transformable Query**

The private function IsTransformable\$2D[] returns True if an object or all the objects in a list are transformable; otherwise, returns False.

```
IsTransformable$2D[obj_List] :=
   (And @@ Map[IsTransformableSingleLevel$2D[#]&,obj]) /;
Not[IsScalarPair2D[obj]];
IsTransformable$2D[obj_] := IsTransformableSingleLevel$2D[obj];
IsTransformableSingleLevel$2D[obj_] :=
   (IsValid2D[obj] || IsScalarPair2D[obj]);
```

## Reflect

#### **Reflect Angle**

**ReflectAngle2D**[ $\theta$ , *line*]  $\blacksquare$  Reflects an angle in a line. This function is useful for reflecting objects that are defined by rotation angles.

```
ReflectAngle2D[theta_?IsScalar2D,Line2D[a_,b_,c_]] :=
2*ArcTan[b,-a]-theta;
```

#### **Reflect Coordinates**

Reflect2D[{x, y}, *line*]  $\blacksquare$  Reflects a list of coordinates {x, y} in a line.

#### **Reflect Equation**

Reflect2D[eqn,  $\{x, y\}$ , line]  $\blacksquare$  Reflects an equation in the variables x and y in a line.

```
Reflect2D[eqn_Equal,{x_?IsScalar2D,y_?IsScalar2D},Line2D[a_,b_,c_]] :=
eqn /. {x->((b^2-a^2)*x-2*a*b*y-2*a*c)/(a^2+b^2),
y->((a^2-b^2)*y-2*a*b*x-2*b*c)/(a^2+b^2)};
```

#### **Reflect List**

Reflect2D[*objList*, *line*]  $\blacksquare$  Reflects a list of  $\{x, y\}$  coordinates or objects.

```
Reflect2D[obj_List?IsTransformable$2D,L:Line2D[a_,b_,c_]] :=
Map[Reflect2D[#,L]&, obj] /;
Not[IsScalarPair2D[obj]];
```

#### Rotate

#### **Rotate About Origin**

Rotate2D[*object*,  $\theta$ ]  $\blacksquare$  Rotates an object by an angle  $\theta$  about the origin.

```
Rotate2D[obj_?IsTransformable$2D,theta_?IsScalar2D] :=
Rotate2D[obj,theta,{0,0}];
```

#### **Rotate Coordinates**

Rotate2D[coords,  $\theta$ , coords]  $\blacksquare$  Rotates a coordinate list  $\{x, y\}$  by an angle  $\theta$  about a position specified by a coordinate list. If the third argument is omitted, it defaults to the origin.

#### **Rotate Equation**

Rotate2D[eqn,  $\{x, y\}$ ,  $\theta$ , coords]  $\blacksquare$  Rotates an equation in the variables x and y by an angle  $\theta$  about a position given by coordinates.

#### **Rotate List**

Rotate2D[*objList*,  $\theta$ , *coords*]  $\blacksquare$  Rotates a list of  $\{x, y\}$  coordinates or objects by an angle  $\theta$  about a position specified by a coordinate list. If the third argument is omitted, it defaults to the origin.

## Scale

#### Scale from Origin

Scale2D[*object*, s]  $\blacksquare$  Scales an object about the origin.

```
Scale2D[obj_?IsTransformable$2D,s_?IsScalar2D] :=
Scale2D[obj,s,{0,0}] /;
Not[IsZeroOrNegative2D[s]];
```

#### **Scales Coordinates**

Scale2D[*coords*, *s*, *coords*]  $\blacksquare$  Scales a coordinate list from a position given by coordinates. If the position is omitted, it defaults to the origin.

#### **Scale Equation**

Scale2D[eqn,  $\{x, y\}$ , s, coords]  $\blacksquare$  Scales an equation in the variables x and y from a position given by coordinates.

## Scale List

Scale2D[*objList*, *s*, *coords*]  $\blacksquare$  Scales a list of  $\{x, y\}$  coordinates or objects from a position given by coordinates. If the position is omitted, it defaults to the origin.

```
Scale2D[obj_List,s_?IsScalar2D, {h_?IsScalar2D,k_?IsScalar2D}] :=
Map[Scale2D[#,s,{h,k}]&, obj] /;
(Not[IsScalarPair2D[obj]] && Not[IsZeroOrNegative2D[s]]);
```

#### **Invalid Scale**

Returns the **\$Failed** symbol when **Scale2D** is called with a non-positive scale, s.

```
Scale2D::invalidScale=
    "The scale factor `1` is invalid; the scale factor must be positive.";
Scale2D[obj_?IsTransformable$2D,s_?IsScalar2D,___] :=
    (Message[Scale2D::invalidScale,s];$Failed) /;
IsZeroOrNegative2D[s];
```

## Translate

#### **Translate Coordinates**

Translate2D[coords,  $\{u, v\}$ ]  $\blacksquare$  Translates a coordinate list delta distance.

```
\label{eq:constraints} Translate2D[ \{x_?IsScalar2D, y_?IsScalar2D\}, \\ \{u_?IsScalar2D, v_?IsScalar2D\} \} := \{x+u, y+v\}; \end{cases}
```

#### **Translate Equation**

**Translate2D**[*eqn*, {x, y}, {u, v}]  $\blacksquare$  Translates an equation in the variables x and y by delta distance.

#### **Translate List**

Translate2D[objList, {u, v}]  $\blacksquare$  Translates a list of {x, y} coordinates or objects delta distance.

## **Epilogue**

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DTransform2D`" *)
```

# D2DTriangle2D

The package D2DTriangle2D implements the Triangle2D object.

## Initialization

| <pre>BeginPackage["D2DTriangle2D`", {"D2DCircle2D`", "D2DExpressions2D`",<br/>"D2DGeometry2D`", "D2DLine2D`", "D2DMaster2D`", "D2DNumbers2D`",<br/>"D2DPoint2D`", "D2DSegment2D`", "D2DSketch2D`", "D2DTransform2D`"}];</pre>  |
|--|
| D2DTriangle2D::usage=<br>"D2DTriangle2D is a package that implements the Triangle2D object.";  |
| Centroid2D::usage=<br>"Centroid2D is the keyword required in Point2D[triangle, Centroid2D].";  |
| <pre>Circumscribed2D::usage=     "Circumscribed2D is the keyword required in Circle2D[triangle, Circumscribed2D]; it is also required in Point2D[triangle, Circumscribed2D].";</pre>   |
| <pre>Inscribed2D::usage=     "Inscribed2D is the keyword required in Circle2D[triangle, Inscribed2D]; it is also required in Point2D[triangle, Inscribed2D].";</pre>   |
| <pre>SolveTriangle2D::usage=    "SolveTriangle2D[{{s1,s2,s3},{a1,a2,a3}}] computes a complete triangle    configuration of the form {{s1,s2,s3},{a1,a2,a3}} given three of the six    sides and/or angles; unspecified sides and/or angles should be specified as    Null; SolveTriangle2D[{{s1,s2,s3},{a1,a2,a3}}, True] computes an alternate    configuration, if one exists.";</pre> |
| <pre>Triangle2D::usage=     "Triangle2D[{x1,y1},{x2,y2},{x3,y3}] is the standard form of a triangle, the coordinates being the vertices of the triangle.";</pre>   |
| <pre>Begin["'Private'"];</pre>   |

## Description

#### Representation

**Triangle2D**[ $\{x_1, y_1\}, \{x_2, y_2\}, \{x_3, y_3\}$ ]  $\blacksquare$  Standard representation of a triangle object in *Descarta2D*. The three arguments are the vertex coordinates of the triangle.

#### Graphics

Provides graphics for a triangle by extending the *Mathematica* Display command. Executed when the package is loaded.

```
SetDisplay2D[
Triangle2D[{x1_,y1_},{x2_,y2_},{x3_,y3_}],
Line[{x1,y1},{x2,y2},{x3,y3},{x1,y1}]];
```

#### Validation

Triangle2D[ $\{x_1, y_1\}, \{x_2, y_2\}, \{x_3, y_3\}$ ]  $\blacksquare$  Detects that the arguments of a triangle are imaginary and returns the **\$Failed** symbol. If the imaginary parts are insignificant, they are removed.

```
Triangle2D::imaginary=
    "An invalid triangle of the form 'Triangle2D['1', '2', '3']' has been
detected; the arguments cannot be imaginary.";
Triangle2D[p1:{x1_,y1_},p2:{x2_,y2_},p3:{x3_,y3_}] :=
    (Triangle2D @@
        ChopImaginary2D[Triangle$2D[p1,p2,p3]]) /;
(FreeQ[{p1,p2,p3},_Pattern] && IsTinyImaginary2D[{p1,p2,p3}]);
Triangle2D[p1:{x1_,y1_},p2:{x2_,y2_},p3:{x3_,y3_}] :=
    (Message[Triangle2D::imaginary,p1,p2,p3];$Failed) /;
(FreeQ[{p1,p2,p3},_Pattern] && IsComplex2D[{p1,p2,p3},0]);
```

Triangle2D[ $\{x_1, y_1\}$ ,  $\{x_2, y_2\}$ ,  $\{x_3, y_3\}$ ]  $\blacksquare$  Detects that the vertex points of a triangle are collinear and returns the **\$Failed** symbol.

```
Triangle2D::invalid=
    "An invalid triangle of the form 'Triangle2D['1', '2', '3']' has
detected; the vertex points cannot be collinear.";
Triangle2D[p1:{x1_,y1_},p2:{x2_,y2_},p3:{x3_,y3_}] :=
    (Message[Triangle2D::invalid,{x1,y1},{x2,y2},{x3,y3}];$Failed) /;
(FreeQ[{p1,p2,p3},_Pattern] &&
    IsCollinear2D[Point2D[p1],Point2D[p2],Point2D[p3]]);
```

**IsValid2D**[*triangle*] ■ Verifies that a triangle is valid.

## Queries

#### **Configuration Query and Check**

The private function IsValidConfiguration\$2D[{ $\{s_1, s_2, s_3\}$ , { $a_1, a_2, a_3$ }}] checks the validity of a complete triangle configuration and returns True if it is valid; otherwise, returns False.

```
IsValidConfiguration$2D[{
    S:{s1_?IsScalar2D,s2_?IsScalar2D,a3_?IsScalar2D},
    A:{a1_?IsScalar2D,a2_?IsScalar2D,a3_?IsScalar2D}]:=True /;
Not[IsZeroOrNegative2D[{s1,s2,s3,a1,a2,a3}]] &&
(IsZero2D[s1*Sin[a2]-s2*Sin[a1]] ||
    Not[IsReal2D[s1*Sin[a2]-s2*Sin[a1]]]) &&
(IsZero2D[s2*Sin[a3]-s3*Sin[a2]] ||
    Not[IsReal2D[s2*Sin[a3]-s3*Sin[a2]]]) &&
(IsZero2D[s1*Sin[a3]-s3*Sin[a1]] ||
    Not[IsReal2D[s1*Sin[a3]-s3*Sin[a1]]]) &&
(IsZero2D[p1-(a1+a2+a3)] ||
    Not[IsReal2D[p1-(a1+a2+a3)]]);
IsValidConfiguration$2D[__]:=False;
```

The private function CheckConfiguration\$2D[{ $\{s_1, s_2, s_3\}$ , { $a_1, a_2, a_3$ }}] checks the validity of a complete triangle configuration and returns the configuration unchanged if it is valid; otherwise, reports an error and returns \$Failed.

```
SolveTriangle2D::invConfig=
    "The configuration of sides and/or angles specified is invalid; no
triangle can be constructed.";
CheckConfiguration$2D[{
    S:{s1_?IsScalar2D,s2_?IsScalar2D,s3_?IsScalar2D},
    A:{a1_?IsScalar2D,a2_?IsScalar2D,a3_?IsScalar2D}]:={S,A} /;
IsValidConfiguration$2D[{S,A}];
CheckConfiguration$2D[__]:=
    (Message[SolveTriangle2D::invConfig];$Failed);
```

#### **Vertex Query**

The private function IsVertex2D[n] returns True if n is a valid triangle vertex number (1, 2 or 3); otherwise, returns False.

```
IsVertex$2D[n_] := (n==1 || n==2 || n==3);
```

## Scalars

#### Angle

Angle2D[*triangle*, n]  $\blacksquare$  Computes the angle at a vertex of a triangle.

```
Angle2D[Triangle2D[p1:{x1_,y1_},p2:{x2_,y2_},p3:{x3_,y3_}],n_] :=
Angle2D[Triangle2D[p2,p3,p1],n-1] /;
(n==2 || n==3);
Angle2D[Triangle2D[p1:{x1_,y1_},p2:{x2_,y2_},p3:{x3_,y3_}],n_] :=
Module[{a,b,c},
a=Distance2D[p1,p2];
b=Distance2D[p1,p3];
c=Distance2D[p2,p3];
ArcCos[(a^2+b^2-c^2)/(2*a*b)] ] /;
(n==1);
```

#### Solve Triangle

SolveTriangle2D[{ $\{s_1, s_2, s_3\}$ , { $a_1, a_2, a_3\}$ }, True |False]  $\blacksquare$  Completely solves a triangle given a partial configuration of side lengths and angles and returns the complete configuration of sides and angles. Three of the six configuration parameters are expected, and the others should be set to Null. If more than three configuration parameters are specified, then they must be consistent. The second argument, when set to True, returns an alternate configuration if two solutions exist; if omitted, it defaults to False. The global variable is used at lower levels to resolve ambiguous cases. The private function SolveTriangle\$2D must be called three times to compute up to three missing configuration parameters.

D2D\$SolveTriangle2D\$AlternateSolution=False;

```
SolveTriangle2D[{
   S:{s1_?IsScalar2D | Null,s2_?IsScalar2D | Null,s3_?IsScalar2D | Null},
   A:{a1_?IsScalar2D | Null,a2_?IsScalar2D | Null,a3_?IsScalar2D | Null}},
   alternateSolution_:False]:=
   (D2D$SolveTriangle2D$AlternateSolution=alternateSolution;
   CheckConfiguration$2D[Nest[SolveTriangle$2D,{S,A},3]]) /;
MemberQ[{True,False},alternateSolution];
```

Checks for under-constrained configurations, and, if detected, displays an error message.

```
SolveTriangle2D::constrain=
    "The triangle configuration is under-constrained; three constraints are
expected.";
SolveTriangle2D[{
    S:{s1_?IsScalar2D | Null,s2_?IsScalar2D | Null,s3_?IsScalar2D | Null},
    A:{a1_?IsScalar2D | Null,a2_?IsScalar2D | Null,a3_?IsScalar2D | Null},
    alternateSolution_:False]:=
    (Message[SolveTriangle2D::constrain];$Failed) /;
(Count[A,Null]>1 && Count[Join[S,A],Null]>3) &&
MemberQ[{True,False},alternateSolution];
```

Two angles are known, compute the third using  $\theta_1 + \theta_2 + \theta_3 = \pi$ .

Three sides are known (SSS), but not all the angles. Compute the missing angles directly.

Three angles are known (AAA) but no sides. Compute all the sides directly. Since this case is under-constrained, we use the added constraint that the perimeter is set equal to 1 and issue a warning message.

Three angles are known and at least one side. Compute the missing side(s) using the Law of Sines.

Two sides and the included angle are known (SAS). Compute the third side using the Law of Cosines.

```
SolveTriangle$2D[
    {S:{s1_?IsScalar2D,Null,s3_?IsScalar2D},A:{a1_,a2_?IsScalar2D,a3_}} |
    {S:{Null,s3_?IsScalar2D,s1_?IsScalar2D},A:{a2_?IsScalar2D,a3_,a1_}} |
    {S:{s3_?IsScalar2D,s1_?IsScalar2D,Null},A:{a3_,a1_,a2_?IsScalar2D}} ]:=
    {S /. Null->Sqrt[s1^2+s3^2-2*s1*s3*Cos[a2]],A};
```

Two sides and one angle (not the included angle) are known (SSA-CCW).

```
SolveTriangle$2D[
    {S:{s1_?IsScalar2D,s2_?IsScalar2D,Null},A:{a1_?IsScalar2D,Null,Null}}
    {S:{s2_?IsScalar2D,Null,s1_?IsScalar2D},A:{Null,Null,a1_?IsScalar2D}}
    {S:{Null,s1_?IsScalar2D,s2_?IsScalar2D},A:{Null,a1_?IsScalar2D,Null}}
    ]:=
    {S /. Null->SolveTriangle$SSA$2D[{s1,s2,a1}],A};
```

One angle (not the included angle) and two sides are known (SSA-CW).

```
SolveTriangle$2D[
{S:{s1_?IsScalar2D,Null,s3_?IsScalar2D},A:{a1_?IsScalar2D,Null,Null}}
{S:{Null,s3_?IsScalar2D,s1_?IsScalar2D},A:{Null,Null,a1_?IsScalar2D}}
{S:{s3_?IsScalar2D,s1_?IsScalar2D,Null},A:{Null,a1_?IsScalar2D,Null}}
]:=
{S /. Null->SolveTriangle$SSA$2D[{s1,s3,a1}],A};
```

Special function for solving SSA cases. Returns the length of the third side of the configuration, or Null if the configuration is invalid.

```
SolveTriangle2D::ambiguous=
   "Two valid solutions exist for this configuration; set the alternate
solution option to ''1'' to compute the other configuration.";
SolveTriangle$SSA$2D[{s1_,s2_,a1_}]:=
Module[{a2,a2alt,a3,a3alt,s3,s3alt,normValid,altValid},
   a2=ArcSin[s2*Sin[a1]/s1]; a2alt=Pi-a2;
   a3=Pi-(a1+a2);
                              a3alt=Pi-(a1+a2alt);
   s3=Sqrt[s1^2+s2^2-2*s1*s2*Cos[a3]];
   s3alt=Sqrt[s1^2+s2^2-2*s1*s2*Cos[a3alt]];
   normValid=IsValidConfiguration$2D[{{s1,s2,s3}, {a1,a2,a3}}];
   altValid=IsValidConfiguration$2D[{{s1,s2,s3alt},{a1,a2alt,a3alt}}];
   If[normValid && altValid && Not[IsZero2D[s3-s3alt]],
      Message[SolveTriangle2D::ambiguous,
             Not[D2D$SolveTriangle2D$AlternateSolution]]];
   Switch[{normValid,altValid,D2D$SolveTriangle2D$AlternateSolution},
      {True ,True ,True }, s3alt,
      {True ,True ,False}, s3,
      {True ,False,True }, s3,
      {True ,False,False}, s3,
      {False,True, True }, s3alt,
      {False,True, False}, s3alt,
      {False,False,True }, Null,
      {False,False,False}, Null]];
```

No other cases match, just return the configuration.

SolveTriangle\$2D[{S:{s1\_,s2\_,s3\_},A:{a1\_,a2\_,a3\_}}]:={S,A};

## Transformations

#### Reflect

**Reflect2D**[*triangle*, *line*] ■ Reflects a triangle in a line.

```
Reflect2D[Triangle2D[{x1_,y1_}, {x2_,y2_}, {x3_,y3_}],
L2:Line2D[A2_,B2_,C2_]] :=
Triangle2D[Reflect2D[{x1,y1},L2],
Reflect2D[{x2,y2},L2],
Reflect2D[{x3,y3},L2]];
```

#### Rotate

Rotate2D[*triangle*,  $\theta$ , *coords*]  $\blacksquare$  Rotates a triangle by an angle  $\theta$  about a position specified by a coordinate list. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

#### Scale

 $Scale2D[triangle, s, coords] \blacksquare$  Scales a triangle from a position given by coordinates. If the third argument is omitted, it defaults to the origin (see D2DTransform2D.nb).

#### Translate

Translate2D[*triangle*,  $\{u, v\}$ ]  $\blacksquare$  Translates a triangle delta distance.

## **Point Construction**

#### Centroid

**Point2D**[*triangle*, **Centroid2D**]  $\blacksquare$  Constructs the centroid point of a triangle. The centroid is the intersection of the medians of the triangle (the lines connecting the vertices to the midpoints of the sides).

```
Point2D[Triangle2D[{x1_,y1_}, {x2_,y2_}, {x3_,y3_}],Centroid2D] :=
Point2D[{x1+x2+x3,y1+y2+y3}/3];
```

#### **Center of Circumscribed Circle**

Point2D[*triangle*, Circumscribed2D]  $\blacksquare$  Constructs the center of the circle that circumscribes a triangle. The center of the circumscribed circle is the intersection of the perpendicular bisectors of the triangle sides.

```
Point2D[Triangle2D[{x1_,y1_}, {x2_,y2_}, {x3_,y3_}], Circumscribed2D] :=
Point2D[Circle2D[Point2D[{x1,y1}], Point2D[{x2,y2}], Point2D[{x3,y3}]]];
```

#### **Center of Inscribed Circle**

**Point2D**[*triangle*, **Inscribed2D**]  $\blacksquare$  Constructs the center of the circle that inscribes a triangle. The center of the inscribed circle is the intersection of the angle bisectors of the triangle sides.

```
Point2D[T1:Triangle2D[{x1_,y1_}, {x2_,y2_}, {x3_,y3_}], Inscribed2D] :=
Point2D[Circle2D[T1,Inscribed2D]];
```

#### **Vertex Point**

Point2D[*triangle*, n]  $\blacksquare$  Constructs a vertex point of a triangle. The vertex points are numbered from 1 to 3.

```
Point2D[T1:Triangle2D[{x1_,y1_}, {x2_,y2_}, {x3_,y3_}],n_?IsVertex$2D] :=
Point2D[T1[[n]]];
```

## Line Construction

#### Side of a Triangle

Line2D[*triangle*,  $n_1$ ,  $n_2$ ]  $\blacksquare$  Constructs the line associated with vertices  $n_1$  and  $n_2$  of a triangle.

## Line Segment Construction

#### Side of a Triangle

Segment2D[*triangle*,  $n_1$ ,  $n_2$ ]  $\blacksquare$  Constructs the line segment associated with vertices  $n_1$  and  $n_2$  of a triangle.

## **Circle Construction**

#### **Circumscribed Circle**

Circle2D[*triangle*, Circumscribed2D] Constructs a circle that circumscribes a triangle.

```
Circle2D[Triangle2D[{x1_,y1_}, {x2_,y2_}, {x3_,y3_}], Circumscribed2D] :=
Circle2D[Point2D[{x1,y1}], Point2D[{x2,y2}], Point2D[{x3,y3}]];
```

#### **Inscribed Circle**

Circle2D[*triangle*, Inscribed2D] Constructs a circle inscribed in a triangle.

```
Circle2D[Triangle2D[p1:{x1_,y1_},p2:{x2_,y2_},p3:{x3_,y3_}],Inscribed2D] :=
Module[{s1,s2,s3,s,r,h,k},
s1=Distance2D[p2,p3];
s2=Distance2D[p1,p3];
s3=Distance2D[p1,p2];
s=(s1+s2+s3)/2;
r=Sqrt[(s-s1)*(s-s2)*(s-s3)/s];
{h,k}=(s1*{x1,y1}+s2*{x2,y2}+s3*{x3,y3})/(2*s);
Circle2D[{h,k},r] ];
```

## **Triangle Construction**

#### **Triangle from Three Points**

**Triangle2D**[*point*, *point*] ■ Constructs a triangle from three vertex points.

```
Triangle2D[Point2D[{x1_,y1_}],Point2D[{x2_,y2_}],Point2D[{x3_,y3_}]] :=
Triangle2D[{x1,y1},{x2,y2},{x3,y3}];
```

#### **Triangle from Three Lines**

**Triangle2D**[*line*, *line*, *line*] ■ Constructs a triangle from three lines that define the sides of the triangle.

```
Triangle2D::noTriangle=
    "Two of the lines `1` are parallel, or the three are concurrent; no
triangle exists.";
```

```
Triangle2D[L1:Line2D[a1_,b1_,c1_],
            L2:Line2D[a2_,b2_,c2_],
            L3:Line2D[a3_,b3_,c3_]] :=
        If[IsParallel2D[{L1,L2,L3}] || IsConcurrent2D[L1,L2,L3],
        Message[Triangle2D::noTriangle, {L1,L2,L3}];$Failed,
        Triangle2D[Point2D[L1,L2],Point2D[L1,L3],Point2D[L2,L3]] ];
```

#### Triangle from Sides/Angles

**Triangle2D**[ $\{s_1, s_2, s_3\}$ ]  $\blacksquare$  Constructs a triangle in standard position from a configuration of three side lengths. The first vertex will be at the origin and the second on the +*x*-axis.

```
Triangle2D[{s1_?IsScalar2D,s2_?IsScalar2D,s3_?IsScalar2D}]:=
Triangle2D[{{s1,s2,s3}, {Null,Null,Null}}];
```

Triangle2D[{{ $s_1, s_2, s_3$ }, { $a_1, a_2, a_3$ }}, True | False]  $\blacksquare$  Constructs a triangle in standard position from a configuration of side lengths and angles. The first vertex will be at the origin and the second on the +x-axis. Three of the six configuration parameters are expected, and the others should be set to Null. If more than three configuration parameters are specified, then they must be consistent. The second argument, when set to True, returns an alternate solution if two solutions exist; if omitted, it defaults to False.

```
Triangle2D[{
    S:{s1_?IsScalar2D | Null,s2_?IsScalar2D | Null,s3_?IsScalar2D | Null},
    A:{a1_?IsScalar2D | Null,a2_?IsScalar2D | Null,a3_?IsScalar2D | Null},
    alternateSolution_:False]:=
Module[{SA,S1,S2,S3,A1,A2,A3,f1,f2,a,b,d},
    SA=SolveTriangle2D[{S,A},alternateSolution];
    If[SA===$Failed,$Failed,
        {{S1,S2,S3},{A1,A2,A3}}=SA;
        f1=-S1^2+S2^2+S3^2;
        f2=-(S1-S2-S3)(S1+S2-S3)(S1+S2+S3)(S1+S2+S3);
        Triangle2D[{0,0},{d,0},{a,b}] /.
        {a->f1/(2*S3),b->Sqrt[f2/S3^2]/2,d->S3}] ] /;
MemberQ[{True,False},alternateSolution];
```

## **Epilogue**

```
End[ ]; (* end of "`Private" *)
EndPackage[ ]; (* end of "D2DTriangle2D`" *)
```

Part VIII Explorations

# apollon.nb

## **Circle of Apollonius**

## **Exploration**

Show that the locus of a point P(x, y) that moves so that the ratio of its distance from two fixed points  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$  is a circle with radius

$$r = \frac{dk}{\sqrt{\left(k^2 - 1\right)^2}}$$

and center

$$\left(\frac{-x_1+k^2x_2}{k^2-1}, \frac{-y_1+k^2y_2}{k^2-1}\right)$$

where  $d = |P_1P_2|$ . The locus is called the Circle of Apollonius for the points  $P_1$  and  $P_2$  and the ratio k.

## Approach

Form the equation of the locus directly from the conditions. Show that the locus is the circle described.

## Solution

Construct the points.

```
In[1]: Clear[x1, y1, x2, y2, x, y];
P1 = Point2D[{x1, y1}];
P2 = Point2D[{x2, y2}];
P = Point2D[{x, y}];
```

Compute the distances.

Form the equation representing the relationship.

```
In [3]: Clear[k] = q1 = k^2 * d2^2 - d1^2 // Expand
Out [3] -x^2 + k^2 x^2 + 2x x1 - x1^2 - 2k^2 x x2 + k^2 x2^2 - y^2 + k^2 y^2 + 2y y1 - y1^2 - 2k^2 y y2 + k^2 y2^2
```

Construct the circle from its equation. The numerator under the radical is dk.

```
In[4]: Cl = Circle2D[Quadratic2D[eq1, {x, y}]] // FullSimplify
out[4] Circle2D[\{\frac{-x1 + k^2 x2}{-1 + k^2}, \frac{-y1 + k^2 y2}{-1 + k^2}\}, \sqrt{\frac{k^2 ((x1 - x2)^2 + (y1 - y2)^2)}{(-1 + k^2)^2}}]
In[5]: Clear[d, E1, E2, E3];
C2 = C1 //. \{
k^2 * ((x1 - x2)^2 + (y1 - y2)^2) \rightarrow d^2 * k^2,
sqrt[E1_^2 * E2_^2 * E3_] \rightarrow E1 * E2 / sqrt[1 / E3]\}
out[5] Circle2D[\{\frac{-x1 + k^2 x2}{-1 + k^2}, \frac{-y1 + k^2 y2}{-1 + k^2}\}, \frac{d k}{\sqrt{(-1 + k^2)^2}}]
```

## Discussion

This is a plot of a numerical example with  $P_1(1,1)$ ,  $P_2(-1,-2)$  and k = 1.5.

-6 -4

-2 0

```
In[6]: d = Distance2D[P1, P2];
Sketch2D[{P1, P2, C2} //. {
    x1 -> 1, y1 -> 1, x2 -> -1, y2 -> -2, k -> 1.5}];

    0
    -2
    -4
    -6
    -8
```
## arccent.nb

## Centroid of Semicircular Arc

### **Exploration**

Show that the centroid of the area bounded by a semicircular arc of radius r and its chord is on the axis of symmetry at a distance

$$H = \frac{4r}{3\pi}$$

from the chord of the arc.

## Approach

Construct representative geometry for the semicircular arc. Using symbolic computations, compute the width of a horizontal rectangle spanning the arc having infinitesimal height. Use integration to find moments of inertia on each side of the centroid. Equate the moments of inertia on each side of the centroid.

## Solution

Construct a semicircular arc of radius r (the portion of the circle above the x-axis).

```
In[1]: Clear[r];
C1 = Circle2D[{0, 0}, r];
```

Construct a horizontal line at height y.

```
In[2]: Clear[y];
L1 = Line2D[Point2D[0, y], 0];
```

Compute the intersection points of the horizontal line with the arc.

```
In[3]: pts = Points2D[L1, C1]Out[3] \{Point2D[\{\sqrt{r^2 - y^2}, y\}], Point2D[\{-\sqrt{r^2 - y^2}, y\}]\}
```

The width of the arc is the difference between the abscissas of the intersection points.

```
In[4]: L = XCoordinate2D[First[pts]] - XCoordinate2D[Last[pts]]
Out[4] 2 \sqrt{r^2 - y^2}
```

By integrating Ld find the moment of inertia of an area, where d is the distance from the centroid line  $(y = \overline{y})$ .  $I_1$  is the expression for the moment of inertia of the upper area with respect to  $\overline{y}$ .

```
In[5]: Clear[yB];
int1 = Integrate[L* (yB - y), y] // Simplify
Out[5] \frac{1}{3} \sqrt{r^2 - y^2} (2r^2 + y(-2y + 3yB)) + r^2 yB ArcTan \left[\frac{y}{\sqrt{r^2 - y^2}}\right]
```

The next few steps show the output computed by *Mathematica* Version 3.0.1. Version 4.0 produces slightly different results that are algebraically equivalent. The final step is the same in both versions.

```
In[6]: \mathbf{I1} = (\mathbf{int1} / . \mathbf{y} \rightarrow \mathbf{yB}) - (\mathbf{int1} / . \mathbf{y} \rightarrow \mathbf{0}) / . (\mathbf{r}^2)^{(3/2)} \rightarrow \mathbf{r}^3Out[6] - \frac{2r^3}{3} + \frac{1}{3}\sqrt{r^2 - \mathbf{yB}^2} (2r^2 + \mathbf{yB}^2) + r^2\mathbf{yB}\operatorname{ArcTan}\left[\frac{\mathbf{yB}}{\sqrt{r^2 - \mathbf{yB}^2}}\right]
```

 $I_2$  is the expression for the moment of inertia of the upper area with respect to  $\overline{y}$ .

```
In[7]: int2 = Integrate[L * (y - yB), y] // Simplify
Out[7] - \frac{1}{3} \sqrt{r^2 - y^2} (2r^2 + y(-2y + 3yB)) - r^2 yB ArcTan[\frac{y}{\sqrt{r^2 - y^2}}]
In[8]: I2 = -Limit[int2, y -> r] - (int2 /. y -> yB) // Simplify
Out[8] - \frac{1}{2} \pi r^2 yB + \frac{1}{3} \sqrt{r^2 - yB^2} (2r^2 + yB^2) + r^2 yB ArcTan[\frac{yB}{\sqrt{r^2 - yB^2}}]
```

The moments of inertia must be the same on each side of the centroid line.

$$In[9]: eql = I2 - I1 == 0$$
  
Out[9]  $\frac{2 r^3}{3} - \frac{1}{2} \pi r^2 yB == 0$ 

Solve for  $\overline{y}$ .

```
In[10]: Solve[eq1, yB]
Out[10] \left\{ \left\{ yB \rightarrow \frac{4 r}{3 \pi} \right\} \right\}
```

## arcentry.nb

## Arc from Bounding Points and Entry Direction

#### **Exploration**

Let  $P_0$  and  $P_1$  be the start and end points of an arc, respectively, and P be a third point on the vector tangent to the arc at  $P_0$ . Show that

$$s = |(P - P_0) \times (P_1 - P_0)|$$
$$c = (P - P_0) \cdot (P_1 - P_0)$$

represent values of s and c useful for computing the bulge factor of the arc.

## Approach

Use the trigonometric definition of a cross-product to justify the value for s. Use the trigonometric definition of a dot product to justify the value for c.

### Solution

The cross-product definition in two dimensions is  $A \times B = |A||B|\sin(\alpha)$  where  $\alpha$  is the angle between vectors A and B. Therefore,  $|(P - P_0) \times (P_1 - P_0)|$  is equal to  $|PP_0||PP_1|\sin(\alpha)$ which is a scalar multiple of  $\sin(\alpha)$ . The dot product trigonometric definition in two dimensions is  $A \cdot B = |A||B|\cos(\alpha)$  where  $\alpha$  is the angle between vectors A and B. Therefore,  $(P - P_0) \cdot (P_1 - P_0)$  is equal to  $|PP_0||PP_1|\cos(\alpha)$  which is the same scalar multiple of  $\cos(\alpha)$ . Therefore, s and c are multiples of the sine and cosine of the angle between the chord and the entry angle as required.

### Discussion

Example: Construct and sketch the arc with start point (3,0) and end point (0,0) with an entry angle vector through the point (4,1). First define functions for the two-dimensional cross-product and magnitude.

 $In[1]: Cross2D[{u1_, v1_}, {u2_, v2_}] := Cross[{u1, v1, 0}, {u2, v2, 0}];$ 

In[2]: Magnitude2D[{u1\_, v1\_, w1\_: 0}] := Sqrt[u1^2 + v1^2 + w1^2];

Compute the bulge factor using s and c. The bulge factor is given by

$$B = \frac{s}{c + \sqrt{c^2 + s^2}}.$$
P0 = Point2D[p0 = {3, 0}];  
P1 = Point2D[p1 = {0, 0}];  
P = Point2D[p = {4, 1}];  
s = Magnitude2D[Cross2D[p - p0, p1 - p0]];  
c = Dot[p - p0, p1 - p0];  
B = s / (c + Sqrt[c^2 + s^2])

Plot the geometry.

 $Out[3] \quad \frac{3}{-3+3\sqrt{2}}$ 

In[3]:

In[4]: Sketch2D[{P0, P1, P, Arc2D[p0, p1, B], Segment2D[P, P0]}];



## arcexit.nb

#### Arc from Bounding Points and Exit Direction

## **Exploration**

Let  $P_0$  and  $P_1$  be the start and end points, respectively, of an arc and P be a third point on the vector tangent to the arc at  $P_1$ . Show that

$$s = |(P_1 - P_0) \times (P - P_1)|$$
$$c = (P_1 - P_0) \cdot (P - P_1)$$

represent values of s and c useful for computing the bulge factor of the arc.

### Approach

Use the trigonometric definition of a cross-product to justify the value for s. Use the trigonometric definition of a dot product to justify the value for c.

#### Solution

Let Q be the point of intersection of the tangents at end points  $P_0$  and  $P_1$ . The entry angle  $QP_0P_1 = \alpha$  is equal to the angle  $QP_1P_0$  because triangle  $\triangle QP_0P_1$  is an isosceles triangle. The cross-product definition in two dimensions is given by  $A \times B = |A||B|\sin(\alpha)$  where  $\alpha$  is the angle between vectors A and B. Therefore, the expression  $|(P_1 - P_0) \times (P - P_1)|$  is  $|P_0P_1||PP_1|\sin(\alpha)$  which is a scalar multiple of  $\sin(\alpha)$ . The dot product trigonometric definition in two dimensions is given by  $A \cdot B = |A||B|\cos(\alpha)$  where  $\alpha$  is the angle between vectors A and B.  $(P_1 - P_0) \cdot (P - P_1)$  is  $|P_0P_1||PP_1|\cos(\alpha)$  and, therefore, is the same scalar multiple of  $\cos(\alpha)$ . Therefore, s and c are multiples of the sine and cosine of the angle between the chord and the entry angle as required.

#### Discussion

Example: Construct and sketch the arc with end points (3,0) and (0,0) with an exit angle vector through the point (1,-1). First define functions for the two-dimensional cross-product and magnitude.

 $In[1]: Cross2D[{u1_, v1_}, {u2_, v2_}] := Cross[{u1, v1, 0}, {u2, v2, 0}];$ 

In[2]: Magnitude2D[{u1\_, v1\_, w1\_: 0}] := Sqrt[u1^2 + v1^2 + w1^2];

Compute the bulge factor using s and c. The bulge factor is given by

$$B = \frac{s}{c + \sqrt{c^2 + s^2}}.$$
 In [3]: P0 = Point2D[p0 = {3, 0}];  
P1 = Point2D[p1 = {0, 0}];

```
P = Point2D[p = {1, -1}];
s = Magnitude2D[Cross2D[p1 - p0, p - p1]];
c = Dot[p1 - p0, p - p1];
B = s / (c + Sqrt[c^2 + s^2])
Out[3] 3/2
-3 + 3 \sqrt{2}
```

Plot the geometry.

In[4]: Sketch2D[{P0, P1, P, Arc2D[p0, p1, B], Segment2D[P, P1]}];



## archimed.nb

Archimedes' Circles

## **Exploration**



Draw the vertical tangent line at the intersection point of the two smaller tangent circles,  $c_1$  and  $c_2$ , in an arbelos (shoemaker's knife, see figure). Prove that the two circles C' and C'' tangent to this line, the large semicircle,  $c_3$ , and  $c_1$  and  $c_2$ , are congruent (have equal radii). These circles are known as Archimedes' Circles.

## Approach

Position the arbelos from the origin using circles whose radii are  $r_1$ ,  $r_2$  and  $r_3 = 2(r_1 + r_2)$  (see definitions, below). Compute the tangent circles as described in the exploration statement. Compare the radii of these circles to show they are equal.

## Solution

Construct the arbelos circles and the tangent line.

```
In[1]: Clear[r1, r2];
c1 = Circle2D[{r1, 0}, r1];
c2 = Circle2D[{2*r1+r2, 0}, r2];
c3 = Circle2D[{(r1+r2), 0}, r1+r2];
l12 = Line2D[Point2D[2*r1, 0], Infinity]
Out[1] Line2D[1, 0, -2r1]
```

Construct the tangent circles.

```
In[2]: Off[Solve2D::infinite];
    t1 = TangentCircles2D[{c1, c3, l12}];
    t2 = TangentCircles2D[{c2, c3, l12}];
    On[Solve2D::infinite];
```

Compare the radii. Since negative radii are invalid, the radius of the Archimedes' Circle is given by  $R = r_1 r_2 / (r_1 + r_2)$ . One pair is above the x-axis, the other pair is below.

```
 \begin{array}{l} In[3]: \mbox{ [Radius2D, t1], } \\ \mbox{ Map[Radius2D, t2] } // \mbox{ Simplify } \\ Out[3] \mbox{ } \left\{ \left\{ -r1, -r2, r2, -\frac{r1r2}{r1+r2}, \frac{r1r2}{r1+r2}, -\frac{r1r2}{r1+r2}, \frac{r1r2}{r1+r2} \right\}, \\ \mbox{ } \left\{ -r1, r1, -r2, -\frac{r1r2}{r1+r2}, \frac{r1r2}{r1+r2}, -\frac{r1r2}{r1+r2}, \frac{r1r2}{r1+r2} \right\} \end{array}
```

# arcmidpt.nb

Midpoint of an Arc

## **Exploration**



Show that the midpoint, P, of a bulge factor arc between points  $P_0$  and  $P_1$  whose bulge factor is B has coordinates

$$P\left(\frac{(x_0+x_1)-B(y_0-y_1)}{2}, \frac{(y_0+y_1)+B(x_0-x_1)}{2}\right).$$

## Approach

Construct the perpendicular bisector of the arc's chord. Offset the midpoint of the chord an appropriate direction and distance to find the arc's midpoint.

## Solution

Create the arc end points.

```
In[1]: Clear[x0, y0, x1, y1];
P0 = Point2D[{x0, y0}];
P1 = Point2D[{x1, y1}];
```

Construct the midpoint of the arc's chord.

In[2]: PM = Point2D[P0, P1];

Rotate  $P_0$  about  $P_M \pi/2$  radians to find Q, which is on the vector from  $P_M$  to P.

```
In[3]: Q = Rotate2D[P0, Pi/2, Coordinates2D[PM]]
```

 $\textit{Out[3] Point2D}\Big[ \Big\{ \frac{x0+x1}{2} - y0 + \frac{y0+y1}{2} \,,\, x0 + \frac{1}{2} \,\, (-x0-x1) \,+\, \frac{y0+y1}{2} \Big\} \Big]$ 

Offset  $P_M$  in the direction of Q by distance h = Bd/2, where d is the distance between  $P_0$  and  $P_1$ .

```
In[4]: Clear[B, d];
P = Point2D[PM, Q, B*d/2] /. d -> Sqrt[(x0 - x1)^2 + (y0 - y1)^2] // Simplify
Out[4] Point2D[{ 1/2 (x0 + x1 + B (-y0 + y1)), 1/2 (B (x0 - x1) + y0 + y1)}]
```

The coordinates of the point at the parameter t = 1/2 produce the same result.

```
In[5]: Arc2D[{x0, y0}, {x1, y1}, B][1/2] // FullSimplify
```

```
\textit{Out[5]} \hspace{0.2cm} \left\{ \hspace{0.2cm} \frac{1}{2} \hspace{0.2cm} ( x0 + x1 + B \hspace{0.2cm} ( -y0 + y1) \hspace{0.2cm} ) \hspace{0.2cm} , \hspace{0.2cm} \frac{1}{2} \hspace{0.2cm} ( B \hspace{0.2cm} ( x0 - x1) \hspace{0.2cm} + y0 + y1) \hspace{0.2cm} \right\}
```

### Discussion

Example: Construct the midpoint of the bulge factor arc with end points (4, 0) and (0, 4) and bulge factor B = 2. First, define a function for computing the midpoint.

Construct the midpoint and plot the geometry.

```
In[7]: P0 = Point2D[p0 = {4, 0}];
P1 = Point2D[p1 = {0, 4}];
P = ArcMidPoint2D[P0, P1, 2];
Sketch2D[{P0, P1, P, Arc2D[p0, p1, 2]}];
```



## caarclen.nb

## Arc Length of a Parabolic Conic Arc

### **Exploration**

Using exact integration in *Mathematica* show that the arc length of a parabolic conic arc with control points  $P_0(0,0)$ ,  $P_A(a,b)$  and  $P_1(1,0)$  can be expressed exactly in symbolic form in terms of elementary functions of a and b.

## Approach

Create the conic arc. Compute the arc length using the standard formula. Show that the result is a function of a and b only.

## Solution

Create the conic arc.

In[1]: Clear[a, b]; cal = ConicArc2D[{0, 0}, {a, b}, {1, 0}, 1/2];

Find the parametric equations.

In[2]: Clear[t];
 {xt, yt} = cal[t] // Simplify
Out[2] {t (-2a(-1+t)+t), -2b(-1+t)t}

Compute the derivatives.

```
In[3]: {Dx, Dy} = Map[D[#, t]&, {xt, yt}] // Simplify
Out[3] {2 (a+t-2at), b (2-4t)}
```

Integrate to find the arc length. The resulting function involves elementary functions of a and b only. The result shown here was computed by *Mathematica* Version 3.0.1. Version 4.0 produces a different result that is algebraically equivalent and involves elementary functions of a and b only.

 $In[4]: II = Integrate[Sqrt[Dx^2 + Dy^2], t];$ arclenl = (II /. t -> 1) - (II /. t -> 0) // Simplify  $Out[4] \quad \frac{\sqrt{1 - 2a + a^2 + b^2} (1 - 3a + 2a^2 + 2b^2)}{1 - 4a + 4a^2 + 4b^2} + \frac{\sqrt{a^2 + b^2} (-a + 2a^2 + 2b^2)}{1 - 4a + 4a^2 + 4b^2} - \frac{b^2 \log[4 (\sqrt{a^2 + b^2} + \frac{a - 2a^2 - 2b^2}{\sqrt{1 - 4a + 4a^2 + 4b^2}})]}{(1 - 4a + 4a^2 + 4b^2)^{3/2}} + \frac{b^2 \log[4 (\sqrt{1 - 2a + a^2 + b^2} + \frac{1 - 3a + 2a^2 + 2b^2}{\sqrt{1 - 4a + 4a^2 + 4b^2}})]}{(1 - 4a + 4a^2 + 4b^2)^{3/2}} + \frac{b^2 \log[4 (\sqrt{1 - 2a + a^2 + b^2} + \frac{1 - 3a + 2a^2 + 2b^2}{\sqrt{1 - 4a + 4a^2 + 4b^2}})]}{(1 - 4a + 4a^2 + 4b^2)^{3/2}}$ 

# caarea1.nb

## Area of a Conic Arc (General)

## **Exploration**

For the conic arc whose control points are (0,0), (a,b) and (d,0), show that the area between the conic arc and its chord is given by

$$A = \frac{bd\rho}{2r^3} \left( \rho r + (-1+\rho)^2 \log_e \left( \frac{1-\rho}{\rho+r} \right) \right)$$

where  $r = \sqrt{-1 + 2\rho} \ (\rho \neq 1/2)$ . Assume b > 0 and d > 0.

## Approach

Construct the conic arc in the given position and use integration to find the area.

## Solution

Construct the conic arc.

In[1]: Clear[a, b, d, p]; cal = ConicArc2D[{0, 0}, {a, b}, {d, 0}, p];

Determine the coordinates of a point at parameter t.

```
In[2]: Clear[t]; \\ \{XI, YI\} = ca1[t] // FullSimplifyOut[2] \left\{ \frac{t (2ap (-1+t) + d (-1+p) t)}{-1+p (1-2t)^2 - 2 (-1+t) t}, \frac{2bp (-1+t) t}{-1+p (1-2t)^2 - 2 (-1+t) t} \right\}
```

Form an implicit equation of the curve by eliminating t.

```
In [3]: Clear[x, y];
eq1 = Eliminate[{x == X1, y == Y1}, t]

Out[3] 4 a<sup>2</sup> p<sup>2</sup> y<sup>2</sup> + a p<sup>2</sup> y (4 b d - 8 b x - 4 d y) ==

4 b<sup>2</sup> d p<sup>2</sup> x - 4 b<sup>2</sup> p<sup>2</sup> x<sup>2</sup> - 4 b d p<sup>2</sup> x y - d<sup>2</sup> y<sup>2</sup> + 2 d<sup>2</sup> p y<sup>2</sup> - d<sup>2</sup> p<sup>2</sup> y<sup>2</sup>
```

Solve the implicit equation for x.

```
In [4]: ans = Solve[eql, x] // FullSimplify
Out [4] \left\{ \left\{ x \rightarrow \frac{b \, d \, p + 2 \, a \, p \, y - d \, \left( p \, y + \sqrt{(b \, p - y) \, (b \, p + y - 2 \, p \, y)} \, \right)}{2 \, b \, p} \right\}, \\ \left\{ x \rightarrow \frac{b \, d \, p + 2 \, a \, p \, y + d \, \left( -p \, y + \sqrt{(b \, p - y) \, (b \, p + y - 2 \, p \, y)} \, \right)}{2 \, b \, p} \right\} \right\}
```

The length, L, of an area element in terms of y is the difference between the two x locations on the curve.

$$In[5]: L = (x /. ans[[2, 1]]) - (x /. ans[[1, 1]]) // FullSimplify$$

$$Out[5] \frac{d\sqrt{(bp-y) (bp+y-2py)}}{bp}$$

The area between the curve and the x-axis is the integral of L evaluated between the limits on the y-axis. The curve is smooth, so we ignore the convergence warning by turning the warning message off. The result shown in this step was computed using *Mathematica* Version 3.0.1. Version 4.0 produces a slightly different result that is algebraically equivalent.

$$In [6]: Off[Integrate::gener]; \\Clear[E1, E2, r]; \\Al = Integrate[L, {y, 0, b * p}] // FullSimplify \\Out [6] \frac{1}{\sqrt{1 - 2 p} (-2 + 4 p)} \left( d \left( \frac{I (-1 + p) \sqrt{-b (-1 + p) p} \sqrt{-b^2 (-1 + p) p^2} Log[\frac{2 I b (-1 + p) p}{\sqrt{1 - 2 p}}}{\sqrt{b p}} + \sqrt{b^2 p^2} \left( \sqrt{1 - 2 p} p + I (-1 + p)^2 Log[2 b p \left( 1 - \frac{I p}{\sqrt{1 - 2 p}} \right)] \right) \right) \right)$$

This is the area formula given in the exploration statement above.

```
In[7]: On[Integrate::gener]; 
A2 = b * d * p * (p * r + (-1 + p) ^2 * Log[(1 - p) / (p + r)]) / (2 * r ^3) /. r -> Sqrt[-1 + 2 * p] 
Out[7] 
\frac{b d p (p \sqrt{-1 + 2 p} + (-1 + p)^2 Log[\frac{1 - p}{p + \sqrt{-1 + 2 p}}])}{2 (-1 + 2 p)^{3/2}}
```

The area under the curve is the same as the area given by the formula.

```
In[8]: IsZero2D[A1 - A2]
Out[8] True
```

# caarea2.nb

## Area of a Conic Arc (Parabola)

## **Exploration**

Show that the area between a conic arc whose projective discriminant is  $\rho = 1/2$  and its chord is given by

 $A = \frac{bd}{3}$ 

when the control points are (0,0), (a,b) and (d,0).

## Approach

Place the conic arc in the position given and use integration to find the area.

## Solution

Create the conic arc.

In[1]: Clear[a, b, d]; cal = ConicArc2D[{0, 0}, {a, b}, {d, 0}, 1/2];

Solve for t in terms of the y-coordinate.

$$In[2]: Clear[t];$$
  
ans = Solve[ca1[t][[2]] == y, t] // Simplify  
$$Out[2] \left\{ \left\{ t \rightarrow \frac{1}{2} - \frac{\sqrt{b-2y}}{2\sqrt{b}} \right\}, \left\{ t \rightarrow \frac{1}{2} \left( 1 + \frac{\sqrt{b-2y}}{\sqrt{b}} \right) \right\} \right\}$$

Find the *x*-coordinate of the left side of the rectangle.

In[3]: XI = cal[t][[1]] /. ans[[1, 1]] // Simplify  $Out[3] \frac{bd - \sqrt{b} d \sqrt{b - 2y} + 2ay - dy}{2b}$ 

Find the *x*-coordinate of the right side of the rectangle.

In[4]: X2 = cal[t][[1]] /. ans[[2, 1]] // Simplify  $Out[4] \frac{bd + \sqrt{b} d\sqrt{b - 2y} + 2ay - dy}{2b}$ 

Find the width of the rectangle.

In [5]: L = X2 - X1 // Simplify Out [5]  $\frac{d\sqrt{b-2y}}{\sqrt{b}}$ 

Find the area by integration ( $\rho = 1/2$ , so the limits of integration are 0 to b/2).

```
In[6]: II = Integrate[L, y] // Simplify;
A1 = (II /. y -> b/2) - (II /. y -> 0)
Out[6] bd/3
```

## cacenter.nb

## Center of a Conic Arc

## **Exploration**

Show that the center (H, K) of a conic arc whose control points are  $P_0(x_0, y_0)$ ,  $P_A(x_A, y_A)$ and  $P_1(x_1, y_1)$  and whose projective discriminant is  $\rho$  is

$$H = \frac{-\rho^2 x_A + (\rho - 1)^2 x_M}{1 - 2\rho}$$
$$K = \frac{-\rho^2 y_A + (\rho - 1)^2 y_M}{1 - 2\rho}$$

where  $P_M(x_M, y_M)$  is the midpoint of the conic arc's chord and has coordinates

$$x_M = \frac{x_0}{x_1} 2$$
 and  $y_M = \frac{y_0 + y_1}{2}$ .

## Approach

Form the quadratic equation of a conic arc and convert it to a quadratic. Find the center point of the quadratic and simplify.

## Solution

Determines the quadratic equation of a conic arc. The following steps were computed using *Mathematica* Version 3.0.1. Version 4.0 produces different results that are algebraically equivalent. Both versions produce the same final step.

```
 \begin{array}{l} In \left\{ 1 \right\}: \ \mbox{Clear[a, b, k, x, y, x0, y0, xA, yA, x1, y1, F];} \\ eql = a * b - k \left( 1 - a - b \right) ^2 /. \\ \left\{ a -> \left( \left( y - yA \right) \, \left( x1 - xA \right) - \left( x - xA \right) \, \left( y1 - yA \right) \right) / F, \\ b -> \left( \left( y - yA \right) \, \left( x0 - xA \right) - \left( x - xA \right) \, \left( y0 - yA \right) \right) / \left( -F \right) \right\} \ // \ \mbox{Simplify} \end{array} \right. \\ Out \left[ 1 \right] \ \ \frac{1}{F^2} \left( -k \, \left( F + x0 \, y - x1 \, y - x \, y0 + xA \, y0 + x \, y1 - xA \, y1 - x0 \, yA + x1 \, yA \right)^2 + \\ \left( xA \, \left( y - y1 \right) + x \, \left( y1 - yA \right) + x1 \, \left( -y + yA \right) \right) \, \left( xA \, \left( -y + y0 \right) + x0 \, \left( y - yA \right) + x \left( -y0 + yA \right) \right) \right) \end{array} \right)
```

Multiply through by  $F^2$ .

```
In[2]: eq2 = eq1 * F * 2
```

Construct the quadratic and the center points of the quadratic.

```
 \begin{split} & In \left[ 3 \right]: \, \textbf{ql} = \textbf{Quadratic2D[eql, \{x, y\}] // Simplify;} \\ & \textbf{cl} = \textbf{Point2D[ql] // Simplify} \\ \\ & \textit{Out[3] Point2D[} \\ & \left\{ \frac{2 \, F \, k \, \left( x0 + x1 - 2 \, xA \right) + \left( -1 + 4 \, k \right) \, xA \, \left( -xA \, y0 - x0 \, y1 + xA \, y1 + x1 \, \left( y0 - yA \right) + x0 \, yA \right) }{ \left( -1 + 4 \, k \right) \, \left( x1 \, y0 - xA \, y0 - x0 \, y1 + xA \, y1 + x0 \, yA - x1 \, yA \right) } , \\ & \frac{2 \, F \, k \, \left( y0 + y1 - 2 \, yA \right) + \left( -1 + 4 \, k \right) \, yA \left( -xA \, y0 - x0 \, y1 + xA \, y1 + x1 \, \left( y0 - yA \right) + x0 \, yA \right) }{ \left( -1 + 4 \, k \right) \, \left( x1 \, y0 - xA \, y0 - x0 \, y1 + xA \, y1 + x1 \, \left( y0 - yA \right) + x0 \, yA \right) }{ \left( -1 + 4 \, k \right) \, \left( x1 \, y0 - xA \, y0 - x0 \, y1 + xA \, y1 + x0 \, yA - x1 \, yA \right) } \Big\} \Big] \end{split}
```

Simplify.

```
In[4]: Clear[p]; 
c2 = c1 /. 
{F -> (y0 - yA) (x1 - xA) - (x0 - xA) (y1 - yA), 
k -> (1 - p)^2 / (4p^2), 
x0 + x1 -> 2 * xM, 
y0 + y1 -> 2 * yM} // FullSimplify 
Out[4] Point2D[{<math>\frac{p^2 xA - (-1 + p)^2 xM}{-1 + 2 p}, \frac{p^2 yA - (-1 + p)^2 yM}{-1 + 2 p}}]
```

Change the signs on the numerator and denominator to get the desired formulas.

In[5]: Map[((-1\*Numerator[#]) / (-1\*Denominator[#]))&, c2]Out[5] Point2D[{ $\frac{-p^2 xA + (-1+p)^2 xM}{1-2 p}, \frac{-p^2 yA + (-1+p)^2 yM}{1-2 p}$ }]

# cacircle.nb

## Circular Conic Arc

#### **Exploration**

Show that the conic arc with control points (0,0), (a,b) and (2a,0) will be a circular arc if

$$\rho = \frac{a(-a + \sqrt{a^2 + b^2})}{b^2}.$$

## Approach

Create the conic arc and find the quadratic associated with it. Force the quadratic's coefficients to represent a circle and solve for  $\rho$ .

## Solution

Create the conic arc.

In[1]: Clear[a, b, p]; cal = ConicArc2D[{0, 0}, {a, b}, {2a, 0}, p];

Construct the quadratic associated with the conic arc.

In[2]: Q1 = Quadratic2D[ca1];

Extract and simplify the coefficients.

```
In [3]: {a1, b1, c1, d1, e1, f1} = Map[Together, List @@ Q1]
Out [3] \left\{-\frac{1}{4 a^2}, 0, \frac{-1+2 p}{4 b^2 p^2}, \frac{1}{2 a}, -\frac{1}{2 b}, 0\right\}
```

Find  $\rho$  that makes the quadratic a circle.

```
In[4]: ans1 = Solve[a1 == c1, p] // Simplify
```

 $\textit{Out[4]} \hspace{0.1in} \left\{ \left\{ p \rightarrow - \frac{a \left(a + \sqrt{a^2 + b^2} \right)}{b^2} \right\}, \hspace{0.1in} \left\{ p \rightarrow \frac{a \left(-a + \sqrt{a^2 + b^2} \right)}{b^2} \right\} \right\}$ 

Use the positive result.

```
In[5]: ans2 = Last[ans1][[1]]Out[5] p \rightarrow \frac{a(-a + \sqrt{a^2 + b^2})}{b^2}
```

## Discussion

A numerical example with a = 6 and b = 40.



## camedian.nb

#### Shoulder Point on Median

#### **Exploration**

Let C be a conic arc with control points  $P_0(x_0, y_0)$ ,  $P_A(x_A, y_A)$  and  $P_1(x_1, y_1)$  and projective discriminant  $\rho$ . Let P be the point on the median  $P_A P_M$  associated with vertex  $P_A$  of  $\Delta P_0 P_A P_1$  such that  $|PP_M|/|P_A P_M| = \rho (P_M(x_M, y_M))$  is the midpoint of  $P_0 P_1$ ). Show that P is coincident with the shoulder point of C, having coordinates

$$(x_M + \rho(x_A - x_M), y_M + \rho(y_A - y_M)).$$

## Approach

Construct the geometry and compare the coordinates of P to the shoulder point coordinates.

#### Solution

Create the conic arc control points.

```
In[1]: Clear[x0, y0, xA, yA, x1, y1];
    p0 = Point2D[P0 = {x0, y0}];
    pA = Point2D[PA = {xA, yA}];
    p1 = Point2D[P1 = {x1, y1}];
```

Construct the midpoint of the chord.

In[2]: pM = Point2D[p0, p1] $Out[2] Point2D[\left\{\frac{x0 + x1}{2}, \frac{y0 + y1}{2}\right\}]$ 

Construct the point on the median. This result was computed using *Mathematica* Version 3.0.1. Version 4.0 computes a slightly different result that is algebraically equivalent. Both versions verify that the points are coincident in the final step.

Construct the shoulder point.

```
In[4]: Clear[xM, yM];
Q = Point2D[{
    xM + p (xA - xM) /. xM -> (x0 + x1) / 2,
    yM + p (yA - yM) /. yM -> (y0 + y1) / 2}] // Simplify
Out[4] Point2D[{ [ 1/2 (x0 - p x0 + x1 - p x1 + 2 p xA) , 1/2 (y0 - p y0 + y1 - p y1 + 2 p yA) }]
```

The point on the median is coincident with the shoulder point.

```
In[5]: IsCoincident2D[P, Q]
Out[5] True
```

## Discussion

This is a plot of a numerical example.

```
In[6]: cal = ConicArc2D[P0, PA, P1, p];
    Sketch2D[{cal, p0, pA, p1, pM, Q,
        Segment2D[pM, pA]} //. {
        x0 -> 0, y0 -> 0, xA -> 2, yA -> 6, x1 -> 6, y1 -> 0, p -> 0.65},
        PlotRange -> All];
```



## caparam.nb

## Parametric Equations of a Conic Arc

### **Exploration**

Show that the parametric equations of a unit conic arc represent the same implicit quadratic equation as the one underlying the conic as derived from the control points  $P_0(0,0)$ ,  $P_A(a,b)$  and  $P_2(1,0)$  and  $\rho$ .

## Approach

Create the unit conic arc. Eliminate t from the parametric equations and construct a quadratic from the result. Construct a quadratic directly from the conic arc. Verify that the two quadratics are identical.

## Solution

Create the unit conic arc.

```
In[1]: Clear[a, b, p];
cal = ConicArc2D[{0, 0}, {a, b}, {1, 0}, p];
```

Eliminate t from the parametric equations.

```
In[2]: Clear[xt, yt, t];
eq1 = Eliminate[{xt == First[ca1[t]], yt == Last[ca1[t]]}, {t}]
Out[2] a p<sup>2</sup> (4b - 8bxt - 4yt) yt + 4a<sup>2</sup> p<sup>2</sup> yt<sup>2</sup> ==
4b<sup>2</sup> p<sup>2</sup> xt - 4b<sup>2</sup> p<sup>2</sup> xt<sup>2</sup> - 4bp<sup>2</sup> xt yt - yt<sup>2</sup> + 2p yt<sup>2</sup> - p<sup>2</sup> yt<sup>2</sup>
```

Construct the quadratic represented by the parametric equations.

```
In[3]: ql = Quadratic2D[eql, {xt, yt}] // SimplifyOut[3] Quadratic2D[4b^2p^2, 4(1-2a)bp^2, 1-2p+(1-2a)^2p^2, -4b^2p^2, 4abp^2, 0]
```

Construct the quadratic from the conic arc.

```
In[4]: q2 = Map[Simplify, Quadratic2D[ca1]] // SimplifyOut[4] Quadratic2D[-4 b<sup>2</sup> p<sup>2</sup>, 4 (-1+2 a) b p<sup>2</sup>, -1+2 p - (1-2 a)<sup>2</sup> p<sup>2</sup>, 4 b<sup>2</sup> p<sup>2</sup>, -4 a b p<sup>2</sup>, 0]
```

Both quadratics are the same, ignoring the -1 factor.

```
In[5]: IsCoincident2D[q1, q2]
Out[5] True
```

## carlyle.nb

#### Carlyle Circle

## **Exploration**

Given a circle,  $C_1$ , passing through the three points (0, 1), (0, -p) and (s, -p), show that the x-coordinates of the intersection points  $P_1(x_1, 0)$  and  $P_2(x_2, 0)$  of  $C_1$  with the x-axis are the roots of the quadratic equation  $x^2 - sx - p = 0$ .

## Approach

Construct the circle through three points and intersect it with the x-axis. Solve the quadratic equation directly and show that the roots are equal to the x-coordinates of the intersection points.

## Solution

Construct the circle through three points.

```
In[1]: Clear[p, s];

Cl = Circle2D[

pl = Point2D[{0, 1}],

p2 = Point2D[{0, -p}],

p3 = Point2D[{s, -p}]] // FullSimplify

Out[1] Circle2D[{\frac{s}{2}, \frac{1-p}{2}}, \frac{1}{2}\sqrt{(1+p)^2 + s^2}]
```

Intersect the circle with the *x*-axis.

```
In[2]: pts = Points2D[Line2D[0, 1, 0], C1] // FullSimplifyOut[2] \left\{ Point2D\left[\left\{\frac{1}{2}\left(s - \sqrt{4 p + s^{2}}\right), 0\right\}\right], Point2D\left[\left\{\frac{1}{2}\left(s + \sqrt{4 p + s^{2}}\right), 0\right\}\right] \right\}
```

Solve the quadratic directly which produces the same roots as the x-axis intersections.

```
 \begin{array}{l} \mbox{In [3]: Clear [x];} \\ \mbox{Solve [x^2 - s * x - p == 0, x]} \end{array} \\ \label{eq:out [3]} Out [3] $ \left\{ \left\{ x \to \frac{1}{2} \left( s - \sqrt{4 \, p + s^2} \right) \right\}, $ \left\{ x \to \frac{1}{2} \left( s + \sqrt{4 \, p + s^2} \right) \right\} \right\} $ \end{array}
```

## Discussion

This is a plot of a numerical example with p = 2 and s = 4.

```
In[4]: Sketch2D[{C1, pts, p1, p2, p3} /. {p -> 2, s -> 4}];
```



The intersection points on the x-axis are the same as the roots of the equation.

$$\label{eq:Incompared} \begin{split} & \text{In}[5]: \mbox{NSolve}[\mbox{$\mathbf{x}^2-4\star x-2==0, x$}] \\ & \text{Out}[5] \ \{\{\mbox{$\mathbf{x} \to -0.44949$}\}, \ \{\mbox{$\mathbf{x} \to 4.44949$}\}\} \end{split}$$

## castill.nb

#### Castillon's Problem

#### Exploration

Let  $P_1$ ,  $P_2$  and  $P_3$  be three points inside the circle  $C_1 \equiv x^2 + y^2 = 1$ . Describe a method for inscribing a triangle inside  $C_1$  such that the sides of the triangle pass through the three given points.

## Approach

Let  $V_1$ ,  $V_2$  and  $V_3$  be the vertex points of the inscribed triangle. Using the rational parametric equations of the circle, express the coordinates of the vertex points in terms of parameters  $t_1$ ,  $t_2$  and  $t_3$ . Form three equations in three unknowns,  $t_1$ ,  $t_2$  and  $t_3$ , using the condition that each of the given points must lie on a line containing one side of the triangle. Solve the three equations for the parameter values of the vertex points.

## Solution

This is a function that returns the rational parameterization of a unit circle at the origin given a parameter value, t.

In[1]: RationalParameterization2D[t\_] :=
 {(1-t^2) / (1+t^2), 2t / (1+t^2)};

Construct three points on  $C_1$  at parameters  $t_1$ ,  $t_2$  and  $t_3$ .

```
\begin{split} In[2]: \mbox{Clear[t1, t2, t3];} \\ & \{\mbox{V1, V2, V3} = \mbox{Map[Point2D[RationalParameterization2D[#]]&,} \\ & \{\mbox{t1, t2, t3}\} ] \\ Out[2] \end{tabular} \left\{ \begin{array}{l} \mbox{Point2D}[\left\{ \frac{1-t1^2}{1+t1^2}, \ \frac{2\,t1}{1+t1^2} \right\} \right], \ \mbox{Point2D}[\left\{ \frac{1-t2^2}{1+t2^2}, \ \frac{2\,t2}{1+t2^2} \right\} \right], \\ & \ \mbox{Point2D}[\left\{ \frac{1-t3^2}{1+t3^2}, \ \frac{2\,t3}{1+t3^2} \right\} \right] \\ \end{split}
```

Construct three lines containing the three sides of the triangle.

```
In[3]: L12 = Line2D[V1, V2] // FullSimplify;
L23 = Line2D[V2, V3] // FullSimplify;
L13 = Line2D[V1, V3] // FullSimplify;
{L12, L23, L13}
Out[3] {Line2D[1-t1t2, t1+t2, -1-t1t2], Line2D[1-t2t3, t2+t3, -1-t2t3],
Line2D[1-t1t3, t1+t3, -1-t1t3]}
```

Form three equations by forcing the points  $P_1$ ,  $P_2$  and  $P_3$  to be on the lines containing the sides of the triangle.

```
In[4]: Clear[x1, y1, x2, y2, x3, y3];
P1 = Point2D[{x1, y1}];
P2 = Point2D[{x2, y2}];
P3 = Point2D[{x3, y3}];
eq1 = Equation2D[L12, {x1, y1}];
eq2 = Equation2D[L13, {x2, y2}];
eq3 = Equation2D[L13, {x3, y3}];
eqns = {eq1, eq2, eq3}
Out[4] {-1 - t1 t2 + (1 - t1 t2) x1 + (t1 + t2) y1 == 0,
-1 - t2 t3 + (1 - t2 t3) x2 + (t2 + t3) y2 == 0,
-1 - t1 t3 + (1 - t1 t3) x3 + (t1 + t3) y3 == 0}
```

Solve the equations for the parameter values. The resulting expressions are complicated and uninteresting, so we suppress them and use them in the graphical illustrations below. Notice that in general there are two solutions to the problem.

```
In[5]: ans = Solve2D[eqns, {t1, t2, t3}] // FullSimplify;
Length[ans]
Out[5] 2
```

#### Discussion

Example 1: Plot the solutions for the points  $P_1(0.25, 0.25)$ ,  $P_2(0.5, -0.5)$  and  $P_3(-0.5, -0.5)$ .

```
In[6]: Sketch2D[{Circle2D[{0, 0}, 1],
    Map[({V1, V2, V3, P1, P2, P3, Segment2D[V1, V2],
        Segment2D[V2, V3], Segment2D[V1, V3]} /. #)&,
        ans]} /.
        {x1 -> -0.25, y1 -> 0.25,
        x2 -> 0.5, y2 -> -0.5,
        x3 -> -0.5, y3 -> -0.5}];
```



Example 2: Plot the solutions for the points  $P_1(0.5, 0.5)$ ,  $P_2(-0.5, 0.5)$  and  $P_3(0, -0.5)$ .

```
In[7]: Sketch2D[{Circle2D[{0, 0}, 1],
Map[({V1, V2, V3, P1, P2, P3, Segment2D[V1, V2],
        Segment2D[V2, V3], Segment2D[V1, V3]} /. #)&,
        ans]} /.
        {x1 -> 0.5, y1 -> 0.5,
        x2 -> -0.5, y2 -> 0.5,
        x3 -> 0, y3 -> -0.5}];
```



## catnln.nb

## Tangent Line at Shoulder Point

### **Exploration**

Let P be the point at parameter value t = 1/2 on a unit conic arc, C, whose control points are  $P_0(0,0)$ ,  $P_A(a,b)$  and  $P_1(1,0)$  and whose projective discriminant is  $\rho$ . Let L be the line tangent to C at t. Show that L is parallel to the chord  $P_0P_1$  at a distance  $b\rho$  from  $P_0P_1$ . The point P is called the *shoulder point* of the conic arc.

## Approach

Create the conic arc and construct a point at t = 1/2. Construct the quadratic underlying the conic arc. Construct the polar of P with respect to the quadratic (the tangent, L). Show that L is horizontal and, therefore, parallel to the conic arc's chord.

## Solution

Create the conic arc.

```
In[1]: Clear[a, b, p];
cal = ConicArc2D[{0, 0}, {a, b}, {1, 0}, p];
```

Construct the point at t = 1/2.

```
In[2]: \mathbf{P} = \mathbf{Point2D}[\mathbf{cal[1/2]}] // SimplifyOut[2] \quad Point2D\left[\left\{\frac{1}{2} + \left(-\frac{1}{2} + \mathbf{a}\right)\mathbf{p}, \mathbf{bp}\right\}\right]
```

Construct the underlying quadratic.

```
In[3]: Q = Quadratic2D[ca1] // Simplify
Out[3] Quadratic2D[-4b^{2}p^{2}, 4(-1+2a)bp^{2}, -1+2p-(1-2a)^{2}p^{2}, 4b^{2}p^{2}, -4abp^{2}, 0]
```

The tangent line at P is horizontal and at a distance  $b\rho$  from  $P_0P_2$ .

In[4]: L = Line2D[P, Q] // Simplify
Out[4] Line2D[0, 1, -bp]

## Discussion

Plot a numerical example with a = 1, b = 2 and  $\rho = 0.45$ .

```
In[5]: Sketch2D[{ca1, P, L} /. {a -> 1, b -> 2, p -> 0.45},
CurveLength2D -> 5, PlotRange -> {{-0.25, 1.25}, {-0.25, 1.25}}];
```



## center.nb

## Center of a Quadratic

### **Exploration**

Show that applying the change in variables

$$x = x' + \frac{2cd - be}{b^2 - 4ac}$$
 and  $y = y' + \frac{2ae - bd}{b^2 - 4ac}$ 

to the quadratic equation  $ax^2 + bxy + cy^2 + dx + ey + f = 0$  causes the linear terms to vanish, implying that the center of the conic is

$$h = \frac{2cd - be}{b^2 - 4ac}, \ k = \frac{2ae - bd}{b^2 - 4ac}$$

### Approach

Directly apply the change in variables to the equation and simplify the resulting quadratic.

## Solution

Apply the specified change in variables.

$$In [1]: Clear[a, b, c, d, e, f, x, y];$$

$$eql = a * x^{2} + b * x * y + c * y^{2} + d * x + e * y + f /.$$

$$\{x -> x + (2 c * d - b * e) / (b^{2} - 4 a * c),$$

$$y -> y + (2 a * e - b * d) / (b^{2} - 4 a * c)\}$$

$$Out [1] f + d \left(\frac{2 c d - b e}{b^{2} - 4 a c} + x\right) + a \left(\frac{2 c d - b e}{b^{2} - 4 a c} + x\right)^{2} + e \left(\frac{-b d + 2 a e}{b^{2} - 4 a c} + y\right) + b \left(\frac{2 c d - b e}{b^{2} - 4 a c} + x\right) \left(\frac{-b d + 2 a e}{b^{2} - 4 a c} + y\right) + c \left(\frac{-b d + 2 a e}{b^{2} - 4 a c} + y\right)^{2}$$

Simplify the quadratic and notice that the linear terms have vanished. This result was computed using *Mathematica* Version 3.0.1. Version 4.0 produces a slightly different result for the constant term that is algebraically equivalent to the one shown here. In[2]: Q1 = Quadratic2D[eq1, {x, y}] // FullSimplify

 $\textit{Out[2] Quadratic2D} \Big[ a, b, c, 0, 0, \frac{c d^2 + e (-b d + a e)}{b^2 - 4 a c} + f \Big]$ 

## Discussion

Notice that the coefficients a, b and c are unaffected by this change in variables.

# chdlen.nb

Chord Length of Intersecting Circles

### **Exploration**



Show that the distance, d, between the intersection points of two circles is given by

$$d = \frac{\sqrt{-(D - r_1 - r_2)(D + r_1 - r_2)(D - r_1 + r_2)(D + r_1 + r_2)}}{D}$$

where D is the distance between the centers of the circles, and  $r_1$  and  $r_2$  are the radii of the two circles.

## Approach

Assume the radii of the two circles centered at  $C_1$  and  $C_2$  are  $r_1$  and  $r_2$ , respectively,  $P_1$  is one of the intersection points, and the distance between the centers is D. The length of the common chord, d, can be found by equating the area (squared) of  $\triangle C_1 C_2 P_1$  using Heron's formula and the standard area formula A = bh/2.

#### Solution

 $A_1$  is the area (squared) by Heron's formula.

```
In[1]: Clear[r1, r2, D1];
s = (r1 + r2 + D1) / 2;
A1 = s (s - r1) (s - r2) (s - D1) // Simplify
Out[1] - 1/16 (D1 - r1 - r2) (D1 + r1 - r2) (D1 - r1 + r2) (D1 + r1 + r2)
```

 $A_2$  is the area (squared) by the standard area formula A = bh/2 (d is the distance between the intersection points, i.e. the length of the chord).

Set the areas equal and solve for d. Take the positive value. This result was computed using Mathematica Version 3.0.1. Version 4.0 produces a different result involving  $\sqrt{-1}$  that is algebraically equivalent.

```
 \begin{array}{l} \mbox{In[3]: ans = Solve[A1 == A2, d] // Fullsimplify} \\ \label{eq:out[3]} & \mbox{Ulambda} \left\{ \left\{ d \rightarrow - \frac{\sqrt{-(D1 - r1 - r2) \ (D1 + r1 - r2) \ (D1 - r1 + r2) \ (D1 + r1 + r2)}}{D1} \right\}, \\ & \mbox{Ulambda} \left\{ d \rightarrow \frac{\sqrt{-(D1 - r1 - r2) \ (D1 + r1 - r2) \ (D1 - r1 + r2) \ (D1 + r1 + r2)}}{D1} \right\} \right\} \end{array}
```

#### Discussion

If the radii are equal the result can be significantly simplified. This result was computed using Mathematica Version 3.0.1. Version 4.0 produces a different result involving  $\sqrt{-1}$  that is algebraically equivalent.

```
 \begin{array}{ll} In [4]: \mbox{ Clear[r];} & \\ & \mbox{ans2 = Last[ans] //. {r1 -> r, r2 -> r} // FullSimplify;} & \\ & \mbox{ans2 //. Sqrt[-D1^4 + 4D1^2r^2] -> D1 * Sqrt[-D1^2 + 4r^2]} \\ Out [4] & \mbox{d} \rightarrow \sqrt{-D1^2 + 4r^2} \end{array} \}
```
# cir3pts.nb

## Circle Through Three Points

#### **Exploration**

Show that the equation of the circle through the three points (0,0), (a,0) and (0,b) is  $x^2 + y^2 - ax - by = 0$ .

## Approach

Find the quadratic (circle) through the three points, then convert it to an equation.

## Solution

Construct the quadratic.

```
In[1]: Clear[a, b];
    Q = Quadratic2D[Point2D[0, 0], Point2D[a, 0], Point2D[0, b]]
Out[1] Quadratic2D[ab, 0, ab, -a<sup>2</sup>b, -ab<sup>2</sup>, 0]
```

Simplify and convert the quadratic to an equation.

```
In[2]: Clear[x, y];
Equation2D[
        Quadratic2D@@ SimplifyCoefficients2D[List@@Q],
        {x, y}]
Out[2] -a x + x<sup>2</sup> - b y + y<sup>2</sup> == 0
```

## circarea.nb

## One-Third of a Circle's Area

#### **Exploration**

Show that the angle,  $\theta$ , subtended by a segment of a circle whose area is one-third of the full circle is the root of the equation

$$\frac{\pi}{3} = \frac{\theta - \sin\theta}{2}.$$

Also, show that  $\theta$  is within 1/2 percent of  $5\pi/6$  radians.

## Approach

Create an expression for the area,  $A_1$ , of a segment in terms of a generic angular span,  $\theta_1$ . Create an expression for the area of a full circle,  $A_2$ . Solve the equation  $A_1 = A_2/3$  for  $\theta_1$ .

#### Solution

Find the area of a circle's segment.

In[1]: Clear[r, t1];
 A1 = SegmentArea2D[c1 = Circle2D[{0, 0}, r], {0, t1}]
Out[1] <sup>1</sup>/<sub>2</sub> r<sup>2</sup> (t1 - Sin[t1])

Find the area of a full circle.

*In*[2]: **A2 = Area2D[c1]** *Out*[2] πr<sup>2</sup>

Form the equation.

In[3]: eql = A1 - A2 / 3 == 0 $Out[3] - \frac{\pi r^2}{3} + \frac{1}{2} r^2 (t1 - Sin[t1]) == 0$ 

Divide both sides by  $r^2$ .

In[4]: eq2 = eq1 /. r^2 -> 1 Out[4]  $-\frac{\pi}{3} + \frac{1}{2}$  (t1 - Sin[t1]) == 0

## Discussion

Solve the equation  $A_1 = A_2/3$  for  $\theta_1$ .

Show that  $t_1$  is close to  $5\pi/6$ .

```
In[6]: Rationalize[N[t1/Pi] /. rt, .005]
Out[6] 5/6
```

Perform a numerical check.

## cirptmid.nb

Circle–Point Midpoint Theorem

## Exploration



Show that the locus of midpoints from a fixed point  $P_0$  to a circle  $C_1$  of radius  $r_1$ , is a circle of radius  $r_1/2$ . Furthermore, show that the center point of the locus is the midpoint of the segment between  $P_0$  and the center of  $C_1$ .

## Approach

Without loss of generality, choose the point  $P_0$  to be the origin and the circle  $C_1$  to have center  $(h_1, 0)$ . Construct the locus of midpoints and examine its form.

## Solution

Construct the circle and the locus of points.

```
In[1]: Clear[h1, r1, t];
         C1 = Circle2D[{h1, 0}, r1];
         pts = Point2D[Point2D[0, 0], Point2D[C1[t]]]
\textit{Out[1] Point2D}\left[\left\{\frac{1}{2} (hl + rl \cos[t]), \frac{1}{2} rl \sin[t]\right\}\right]
```

This locus is clearly a circle of radius  $r_1/2$  centered at  $(h_1/2, 0)$ , which is the midpoint of the line segment from the point to the circle's center.

## Discussion

Here's a function that computes the midpoint circle in the special position.

```
In[2]: Circle2D[Circle2D[{h_, 0}, r_]] :=
       Circle2D[{h/2,0},r/2];
```

The first plot is a numerical example with the origin outside the circle  $(r_1 = 1)$ , while the second plot's origin is inside the circle  $(r_1 = 3)$ .

```
In[3]: Map[(p0 = Point2D[0, 0];
           p1 = Point2D[C1[Pi / 6]];
           11 = Segment2D[p0, Point2D[C1[Pi / 6]]];
           C2 = Circle2D[C1];
           P = Point2D[p0, p1];
           Sketch2D[{C1, C2, p0, p1, l1, P} /. #])&,
           \{ \{ h1 \rightarrow 2, r1 \rightarrow 1 \}, \{ h1 \rightarrow 2, r1 \rightarrow 3 \} \} ];
```



0

# cramer2.nb

## Cramer's Rule (Two Equations)

## **Exploration**

Show that the solution to the system of two linear equations in two unknowns

$$a_1x + b_1y + c_1 = 0 a_2x + b_2y + c_2 = 0$$

is given by

$$x = \frac{\begin{vmatrix} -c_1 & b_1 \\ -c_2 & b_2 \end{vmatrix}}{D} \text{ and } y = \frac{\begin{vmatrix} a_1 & -c_1 \\ a_2 & -c_2 \end{vmatrix}}{D},$$

where

$$D = \left| \begin{array}{cc} a_1 & b_1 \\ a_2 & b_2 \end{array} \right| \,.$$

## Approach

Use the *Mathematica* **Det** command to compute the appropriate determinants and then substitute the solutions back into the original equations to demonstrate that they solve the equations.

## Solution

Compute the necessary determinants.

```
In[1]: Clear[a1, b1, c1, a2, b2, c2];
    dx = Det[{{-c1, b1}, {-c2, b2}}];
    dy = Det[{{a1, -c1}, {a2, -c2}}];
    dD = Det[{{a1, b1}, {a2, b2}}];
```

Compute the solutions.

 $In[2]: \{x1, y1\} = \{dx / dD, dy / dD\}$  $Out[2] \{\frac{-b2 c1 + b1 c2}{-a2 b1 + a1 b2}, \frac{a2 c1 - a1 c2}{-a2 b1 + a1 b2}\}$ 

Show that the solutions solve the original equations.

```
In[3]: Clear[x, y];
        {a1*x+b1*y+c1, a2*x+b2*y+c2} /.
        {x -> x1, y -> y1} // Simplify
Out[3] {0, 0}
```

## Discussion

The Solve command produces the same result in *Mathematica* Version 3.0.1. Version 4.0 computes a slightly different result that is algebraically equivalent.

```
In[4]: Solve[\{a1 * x + b1 * y + c1 == 0, \\ a2 * x + b2 * y + c2 == 0\}, \{x, y\}] // SimplifyOut[4] \left\{ \left\{ x \rightarrow \frac{-b2 c1 + b1 c2}{-a2 b1 + a1 b2}, y \rightarrow \frac{a2 c1 - a1 c2}{-a2 b1 + a1 b2} \right\} \right\}
```

# cramer3.nb

## Cramer's Rule (Three Equations)

#### **Exploration**

Show that the solution to the system of three linear equations in three unknowns

 $a_1x + b_1y + c_1z + d_1 = 0$   $a_2x + b_2y + c_2z + d_2 = 0$  $a_3x + b_3y + c_3z + d_3 = 0$ 

is given by

$$x = \frac{\begin{vmatrix} -d_1 & b_1 & c_1 \\ -d_2 & b_2 & c_2 \\ -d_3 & b_3 & c_3 \end{vmatrix}}{D}, y = \frac{\begin{vmatrix} a_1 & -d_1 & c_1 \\ a_2 & -d_2 & c_2 \\ a_3 & -d_3 & c_3 \end{vmatrix}}{D} \text{ and } z = \frac{\begin{vmatrix} a_1 & b_1 & -d_1 \\ a_2 & b_2 & -d_2 \\ a_3 & b_3 & -d_3 \end{vmatrix}}{D}$$

where

$$D = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}.$$

## Approach

Use the *Mathematica* **Det** command to compute the appropriate determinants and then substitute the solutions back into the original equations to demonstrate that they solve the equations.

## Solution

Compute the necessary determinants.

```
In[1]: Clear[a1, b1, c1, d1, a2, b2, c2, d2, a3, b3, c3, d3];
dx = Det[{-d1, b1, c1}, {-d2, b2, c2}, {-d3, b3, c3}];
dy = Det[{a1, -d1, c1}, {a2, -d2, c2}, {a3, -d3, c3}];
dz = Det[{a1, b1, -d1}, {a2, b2, -d2}, {a3, b3, -d3}];
dD = Det[{a1, b1, c1}, {a2, b2, c2}, {a3, b3, c3}];
```

Compute the solutions.

```
In[2]: {x1, y1, z1} = {dx / dD, dy / dD, dz / dD}
```

```
Out [2] \left\{ \begin{array}{l} b3 c2 d1 - b2 c3 d1 - b3 c1 d2 + b1 c3 d2 + b2 c1 d3 - b1 c2 d3 \\ \hline -a3 b2 c1 + a2 b3 c1 + a3 b1 c2 - a1 b3 c2 - a2 b1 c3 + a1 b2 c3 \\ \hline -a3 c2 d1 + a2 c3 d1 + a3 c1 d2 - a1 c3 d2 - a2 c1 d3 + a1 c2 d3 \\ \hline -a3 b2 c1 + a2 b3 c1 + a3 b1 c2 - a1 b3 c2 - a2 b1 c3 + a1 b2 c3 \\ \hline a3 b2 d1 - a2 b3 d1 - a3 b1 d2 + a1 b3 d2 + a2 b1 d3 - a1 b2 d3 \\ \hline -a3 b2 c1 + a2 b3 c1 + a3 b1 c2 - a1 b3 c2 - a2 b1 c3 + a1 b2 c3 \\ \hline -a3 b2 c1 + a2 b3 c1 + a3 b1 c2 - a1 b3 c2 - a2 b1 c3 + a1 b2 c3 \\ \hline \end{array} \right\}
```

Show that the solutions solve the original equations.

```
In[3]: Clear[x, y];
    {a1 * x + b1 * y + c1 * z + d1,
        a2 * x + b2 * y + c2 * z + d2,
        a3 * x + b3 * y + c3 * z + d3 } /.
        {x -> x1, y -> y1, z -> z1} // Simplify
Out[3] {0, 0, 0}
```

#### Discussion

The Solve command produces the same result in *Mathematica* Version 3.0.1. Version 4.0 computes a slightly different expression that is algebraically equivalent.

```
 \begin{array}{l} \text{In}[4]: \mbox{Clear}[z]; \\ \mbox{simplify}[ \\ \mbox{Solve}[\{al * x + bl * y + cl * z + dl == 0, \\ & a2 * x + b2 * y + c2 * z + d2 == 0, \\ & a3 * x + b3 * y + c3 * z + d3 == 0\}, \{x, y, z\}] \\ \mbox{J} \\ \mbox{Out}[4] \end{tabular} \left\{ \left\{ x \rightarrow \frac{b3 c2 dl - b2 c3 dl - b3 c1 d2 + b1 c3 d2 + b2 c1 d3 - b1 c2 d3}{-a3 b2 c1 + a2 b3 c1 + a3 b1 c2 - a1 b3 c2 - a2 b1 c3 + a1 b2 c3}, \\ & y \rightarrow \frac{-a3 c2 dl + a2 c3 dl + a3 c1 d2 - a1 c3 d2 - a2 c1 d3 + a1 c2 d3}{-a3 b2 c1 + a2 b3 c1 + a3 b1 c2 - a1 b3 c2 - a2 b1 c3 + a1 b2 c3}, \\ & z \rightarrow \frac{a3 b2 dl - a2 b3 dl - a3 b1 d2 + a1 b3 d2 + a2 b1 d3 - a1 b2 d3}{-a3 b2 c1 + a2 b3 c1 + a3 b1 c2 - a1 b3 c2 - a2 b1 c3 + a1 b2 c3} \} \end{array}
```

# deter.nb

## Determinants

### **Exploration**

Determinants often provide a concise notation for expressing relationships in analytic geometry. Show that the expanded algebraic form for the  $2 \times 2$  determinant

| $a_1$ | $b_1$ |
|-------|-------|
| $a_2$ | $b_2$ |

is given by  $-a_2b_1 + a_1b_2$ .

Show that the expanded algebraic form for the  $3 \times 3$  determinant

| $a_1$ | $b_1$ | $c_1$ |
|-------|-------|-------|
| $a_2$ | $b_2$ | $c_2$ |
| $a_3$ | $b_3$ | $c_3$ |

is given by  $-a_3b_2c_1 + a_2b_3c_1 + a_3b_1c_2 - a_1b_3c_2 - a_2b_1c_3 + a_1b_2c_3$ .

## Approach

Use the Mathematica Det command to compute the desired determinants.

## Solution

The Det command produces the desired results directly.

```
In[1]: Clear[a1, b1, a2, b2];
Det[{{a1, b1}, {a2, b2}}]
Out[1] -a2 b1 + a1 b2
```

In[2]: Clear[a1, b1, c1, a2, b2, c2, a3, b3, c3]; Det[{a1, b1, c1}, {a2, b2, c2}, {a3, b3, c3}]

Out [2] -a3 b2 c1 + a2 b3 c1 + a3 b1 c2 - a1 b3 c2 - a2 b1 c3 + a1 b2 c3

# elfocdir.nb

## Focus of Ellipse is Pole of Directrix

#### **Exploration**

Show that the focus of an ellipse is the pole of the corresponding directrix.

## Approach

Construct the directrix and the pole of the focus and verify that they are the same lines.

## Solution

Construct the required geometry.

```
In [1]: Clear[a, b];
e1 = Ellipse2D[{0, 0}, a, b, 0];

fpts = Foci2D[e1];

dlns = Directrices2D[e1] // Simplify

Out [1] {Line2D[1, 0, -\frac{a^2}{\sqrt{a^2-b^2}}], Line2D[1, 0, \frac{a^2}{\sqrt{a^2-b^2}}]}
```

Construct the polars of the foci.

```
In[2]: lns = {Line2D[fpts[[1]], e1], Line2D[fpts[[2]], e1]} // Simplify
```

 $\textit{Out[2]} \ \left\{ \texttt{Line2D} \left[ \sqrt{a^2 - b^2} \ , \ 0 \ , \ -a^2 \right] \ , \ \texttt{Line2D} \left[ -\sqrt{a^2 - b^2} \ , \ 0 \ , \ -a^2 \right] \right\}$ 

The lines in pairs are coincident.

## Discussion

This is a plot of a numerical example with a = 1.5 and b = 1.

In[4]: Sketch2D[{el, fpts, dlns} /. {a -> 1.5, b -> 1}, CurveLength2D -> 3];



# elimlin.nb

## Eliminate Linear Terms

#### **Exploration**

Show that applying the change in variables

$$x' = x - \frac{d}{2a}$$
 and  $y' = y - \frac{e}{2a}$ 

to the quadratic equation  $ax^2 + cy^2 + dx + ey + f = 0$  yields the quadratic

$$ax'^{2} + cy'^{2} - \frac{d^{2}}{4a} - \frac{e^{2}}{4c} + f = 0$$

whose linear terms have vanished.

## Approach

Apply the transformation rules directly to the quadratic equation.

## Solution

Apply the transformation rules to the equation.

In[1]: Clear[a, c, d, e, f, x, y];a\*x^2+c\*y^2+d\*x+e\*y+f /. {x -> x - d/(2a), y -> y - e/(2c)} // Expand Out[1] -  $\frac{d^2}{4a} - \frac{e^2}{4c} + f + ax^2 + cy^2$ 

# elimxy1.nb

### Eliminate Cross-Term by Rotation

#### **Exploration**

Show that by rotating a quadratic  $ax^2+bxy+cy^2+dx+ey+f=0$  through an angle  $\theta$  given by

$$\tan(2\theta) = \frac{b}{c-a}$$

the xy term will vanish.

### Approach

Create a quadratic and rotate it by an angle  $\theta$ . Show that the coefficient of the xy term is zero.

## Solution

Create a quadratic.

In[1]: Clear[a, b, c, d, e, f];
 Q = Quadratic2D[a, b, c, d, e, f];

Rotate the quadratic.

In[2]: Q1 = Rotate2D[Q, ArcTan[b/(c-a)]/2];

Simplify the coefficient of the xy term.

In[3]: Q1[[2]] // Simplify
Out[3] 0

# elimxy2.nb

## Eliminate Cross-Term by Change in Variables

## **Exploration**

Show that applying the change in variables x' = kx + y and y' = ky - x, where

$$k = \frac{(c-a)}{b} + \sqrt{\left(\frac{c-a}{b}\right)^2 + 1},$$

to the equation  $ax^2 + bxy + cy^2 + dx + ey + f = 0$  will cause the xy term to vanish and a new quadratic with the following coefficients will be formed:

$$a' = ak^{2} - bk + c$$
$$b' = 0$$
$$c' = ck^{2} + bk + a$$
$$d' = dk - e$$
$$e' = ek + d$$
$$f' = f.$$

## Approach

Create a quadratic and form a quadratic equation. Apply the change in variables and examine the coefficients.

## Solution

Create a quadratic.

```
In[1]: Clear[a, b, c, d, e, f];
Q1 = Quadratic2D[a, b, c, d, e, f];
```

Form the quadratic equation and apply the change in variables.

Examine the resulting coefficients.

```
In[3]: Q2 = Quadratic2D[eq1, {x, y}]
Out[3] Quadratic2D[c - b k + a k<sup>2</sup>, -b + 2 a k - 2 c k + b k<sup>2</sup>, a + b k + c k<sup>2</sup>, -e + d k, d + e k, f]
```

The xy term is zero.

```
In[4]: Q2[[2]] /. k -> (c - a) /b + Sqrt[((c - a) /b) ^2 + 1] // Simplify
Out[4] 0
```

# elimxy3.nb

## Eliminate Cross-Term by Change in Variables

#### **Exploration**

Show that applying the change in variables x' = kx + y and y' = ky - x, where

$$k = \frac{(c-a)}{b} + \sqrt{\left(\frac{c-a}{b}\right)^2 + 1},$$

to the equation  $ax^2 + bxy + cy^2 + dx + ey + f = 0$  is equivalent to rotating the quadratic by an angle  $\theta$  given by

$$\tan \theta = \frac{1}{k}$$

and scaling the quadratic by a scale factor

$$s = \frac{1}{\sqrt{1+k^2}}.$$

## Approach

Create a quadratic and rotate and scale it as specified. Compare the result to the result of elimxy2.nb.

## Solution

Create a quadratic.

Rotate it by the specified angle. The results shown in the next few steps were computed using *Mathematica* Version 3.0.1. Version 4.0 produces similar results except the coefficients are multiplied by a constant. Both versions produce the same result in the final step.

```
\begin{split} In[2]: \mbox{ Clear[k];} \\ & \mbox{Q2 = Rotate2D[Q1, ArcTan[1/k]] // Simplify} \\ Out[2] \mbox{ Quadratic2D[c+k(-b+ak), 2(a-c)k+b(-1+k^2), a+k(b+ck), } \\ & \ \sqrt{1+\frac{1}{k^2}} \ k(-e+dk), \ \sqrt{1+\frac{1}{k^2}} \ k(d+ek), \ f(1+k^2) \ ] \end{split}
```

As shown in elimxy2.nb, the xy term must vanish.

```
In[3]: \mathbf{Q2[[2]]} = \mathbf{0}; \mathbf{Q2}
Out[3] \quad Quadratic2D[c+k(-b+ak), 0, a+k(b+ck), \sqrt{1+\frac{1}{k^2}}k(-e+dk), \sqrt{1+\frac{1}{k^2}}k(-e+dk), \sqrt{1+\frac{1}{k^2}}k(d+ek), f(1+k^2)]
```

Scale as specified.

```
 \begin{array}{l} In\left[4\right]: \ \mathbf{Q3} = \mathbf{Scale2D}[\mathbf{Q2}, \mathbf{1} / \mathbf{Sqrt}[\mathbf{1} + \mathbf{k}^{\mathbf{2}}\mathbf{2}] ] \ // \ \mathbf{Simplify} \\ \\ Out\left[4\right] \ \mathsf{Quadratic2D}\Big[ \left(1 + \mathbf{k}^{2}\right) \ \left(\mathbf{c} + \mathbf{k} \ \left(-\mathbf{b} + \mathbf{a} \, \mathbf{k} \right)\right), \ \mathbf{0}, \ \left(1 + \mathbf{k}^{2}\right) \ \left(\mathbf{a} + \mathbf{k} \ \left(\mathbf{b} + \mathbf{c} \, \mathbf{k} \right)\right), \\ \\ \sqrt{1 + \frac{1}{\mathbf{k}^{2}}} \ \mathbf{k} \ \left(-\mathbf{e} + \mathbf{d} \, \mathbf{k} \right) \ \sqrt{1 + \mathbf{k}^{2}}, \ \sqrt{1 + \frac{1}{\mathbf{k}^{2}}} \ \mathbf{k} \ \left(\mathbf{d} + \mathbf{e} \, \mathbf{k} \right) \ \sqrt{1 + \mathbf{k}^{2}}, \ \mathbf{f} \ \left(1 + \mathbf{k}^{2}\right) \Big] \\ \end{array}
```

Simplify, showing the same result as elimxy2.nb.

```
In[5]: Q4 = Q3 /. {Sqrt[1 + k^ (-2)] *k -> Sqrt[1 + k^2]} // Simplify
Out[5] Quadratic2D[c + k (-b + a k), 0, a + k (b + c k), -e + d k, d + e k, f]
```

## elldist.nb

### Ellipse Locus, Distance from Two Lines

#### **Exploration**

A point moves so that the sum of the squares of its distances from two intersecting straight lines is a constant. Prove that its locus is an ellipse.

## Approach

Compute the distances from a generic point (x, y) to the lines and show that the equation must be an ellipse.

## Solution

Create the two lines and a generic point.

Sum of distances squared is a constant, K.

```
In [2]: Clear[K];

eq1 = Distance2D[pt, 11]^2 + Distance2D[pt, 12]^2 - K

Out [2] -K + \frac{(C1 + A1 x + B1 y)^2}{A1^2 + B1^2} + \frac{(C2 + A2 x + B2 y)^2}{A2^2 + B2^2}
```

Form the quadratic equation (without loss of generality, assume the lines are normalized).

```
In[3]: Q1 = Quadratic2D[eq1, {x, y}] /.
        {A1^2 + B1^2 -> 1, A2^2 + B2^2 -> 1}
Out[3] Quadratic2D[A1<sup>2</sup> + A2<sup>2</sup>, 2 A1 B1 + 2 A2 B2, B1<sup>2</sup> + B2<sup>2</sup>, 2 A1 C1 + 2 A2 C2,
        2 B1 C1 + 2 B2 C2, C1<sup>2</sup> + C2<sup>2</sup> - K]
```

Compute the discriminant of the quadratic,  $B^2 - 4AC$ .

```
In[4]: disc = Q1[[2]] ^ 2 - 4 * Q1[[1]] * Q1[[3]] // Simplify
Out[4] -4 (A2 B1 - A1 B2)<sup>2</sup>
```

The discriminant of the quadratic,  $B^2 - 4AC$ , is negative; therefore, the curve is an ellipse. Note that the expression  $(A_2B_1 - A_1B_2)^2$  cannot be zero if the lines intersect.

## Discussion

This is a plot of a numerical example using three different values of K.

```
In[5]: Sketch2D[{l1, l2,
Map[(Q1 /. #)&, {K -> 2, K -> 3, K -> 6}]} /. {
A1 -> 1, B1 -> 1.5, C1 -> -1,
A2 -> -0.5, B2 -> 2.5, C2 -> -1},
CurveLength2D -> 5];
```

-2

-1

0

1

2

# ellfd.nb

#### Ellipse from Focus and Directrix

#### **Exploration**

Show that the ellipse with focus  $F(x_1, y_1)$ , directrix line  $L \equiv px + qy + r = 0$  and eccentricity, 0 < e < 1, is defined by the constants

$$h = x_1 + \frac{paeD}{d}, \quad k = y_1 + \frac{qaeD}{d},$$
$$a = d\frac{e}{(1 - e^2)}, \quad b = a\sqrt{1 - e^2}, \quad \theta = \tan^{-1}(p, q),$$

where

$$d = \sqrt{\frac{(px_1 + qy_1 + r)^2}{p^2 + q^2}}$$
 and  $D = \frac{px_1 + qy_1 + r}{p^2 + q^2}$ .

## Approach

Apply the definition of an ellipse to the supplied focus and directrix for a general point (x, y) and show that the derived locus is an ellipse.

## Solution

The rotation angle of the ellipse is the angle the line perpendicular to L makes with the +x-axis (in Mathematica ArcTan[p, q] is ArcTan[q/p], the first form takes into account the quadrant of the point (p,q)).

```
In[1]: Clear[p,q,r];
L = Line2D[p,q,r];
theta = Angle2D[Line2D[0,1,0], Line2D[Point2D[0,0],L]];
theta // Simplify
Out[1] ArcTan[ <sup>q</sup>/<sub>p</sub>]
```

Now we must show that the lengths a and b are given by the formulas. In standard position the distance from the focus of an ellipse to its directrix is given by d = a/e - ae. Solving for a gives the following result in *Mathematica* Version 3.0.1. Version 4.0 produces a slightly different result that is algebraically equivalent.

```
In [2]: Clear[d, a, e];
Solve[d == a/e - a * e, a] // Simplify
Out [2] \left\{ \left\{ a \rightarrow \frac{d e}{1 - e^2} \right\} \right\}
```

Also, the eccentricity is given by  $e = \sqrt{a^2 - b^2}/a$  and solving for b gives (take the positive result).

```
\begin{split} & \text{In}\left[3\right]: \text{ Solve}\left[e = = \text{Sqrt}\left[a^2 - b^2\right] / a, b\right] \\ & \text{Out}\left[3\right] \; \left\{ \left\{b \rightarrow -a \sqrt{1 - e^2}\right\}, \; \left\{b \rightarrow a \sqrt{1 - e^2}\right\} \right\} \end{split}
```

The eccentricity is the ratio of the distance from a general point to the focus to the distance to the directrix.

```
In[4]: Clear[x1, y1, x, y]; 
F = Point2D[x1, y1]; 
P = Point2D[x, y]; 
{dF = Distance2D[P, F], 
dL = Distance2D[P, L]} 
Out[4] {<math>\sqrt{(x-x1)^{2} + (y-y1)^{2}}, \sqrt{\frac{(r+px+qy)^{2}}{p^{2}+q^{2}}}}
```

Form the equation for the eccentricity squared.

```
In[5]: eql = e^2 * dL^2 - dF^2 // Expand // Together
Out[5] \frac{1}{p^2 + q^2} (e^2 r^2 + 2 e^2 p r x - p^2 x^2 + e^2 p^2 x^2 - q^2 x^2 + 2 p^2 x xl + 2 q^2 x xl - p^2 xl^2 - q^2 xl^2 + 2 e^2 q r y + 2 e^2 p q x y - p^2 y^2 - q^2 y^2 + e^2 q^2 y^2 + 2 p^2 y yl + 2 q^2 y yl - p^2 yl^2 - q^2 yl^2)
```

Find the coordinates  $(h_1, k_1)$  of the center of the quadratic.

```
In[6]: \{h1, k1\} = Coordinates2D[

Point2D[

Q1 = Quadratic2D[eq1, {x, y}] // Simplify] // Simplify

Out[6] \left\{ \frac{-(p^2 + q^2) x1 + e^2 (q^2 x1 - p (r + q y1))}{(-1 + e^2) (p^2 + q^2)}, - \frac{(p^2 + q^2) y1 + e^2 (q (r + p x1) - p^2 y1)}{(-1 + e^2) (p^2 + q^2)} \right\}
```

Find the coordinates of the center using the formula provided.

```
 \begin{array}{l} In [7]: \mbox{Clear[D1];} \\ \mbox{ } \{h2, k2\} = \{x1 + p \ast a \ast e \ast D1 / d, \ y1 + q \ast a \ast e \ast D1 / d\} \ //. \\ \mbox{ } \{a -> d \ast e / (1 - e^2), \\ \mbox{ } b -> a \ast Sqrt[1 - e^2], \\ \mbox{ } d -> Sqrt[(p \ast x1 + q \ast y1 + r)^2 / (p^2 + q^2)], \\ \mbox{ } D1 -> (p \ast x1 + q \ast y1 + r) / (p^2 + q^2)\} \\ Out [7] \ \left\{x1 + \frac{e^2 p \ (r + p \ x1 + q \ y1)}{(1 - e^2) \ (p^2 + q^2)}, \ y1 + \frac{e^2 q \ (r + p \ x1 + q \ y1)}{(1 - e^2) \ (p^2 + q^2)}\right\} \end{array}
```

This shows that the center indeed has the same coordinates as the point from the formula.

```
In[8]: {h1 - h2, k1 - k2} // Simplify
Out[8] {0, 0}
```

## Discussion

An example showing the construction of an ellipse from its focus, directrix and eccentricity.

```
In [9]: focus1 = Point2D[{1/2, 1}];
directrix1 = Line2D[5, 8, -20];
eccentricity1 = 3/4;
ellipse1 = Ellipse2D[focus1, directrix1, eccentricity1]
Out [9] Ellipse2D[{-\frac{116}{623}, -\frac{61}{623}}, \frac{114}{7\sqrt{89}}, \frac{57}{2\sqrt{623}}, ArcTan[\frac{8}{5}]]In [10]: Sketch2D[{focus1, directrix1, ellipse1}, CurveLength2D -> 5];
```



# ellips2a.nb

#### Sum of Focal Distances of an Ellipse

#### **Exploration**

Show that the sum of the distances from the two foci to any point on an ellipse is 2a, where a is the length of the semi-major axis.

### Approach

Construct a generic point on an ellipse. Construct the two foci of the ellipse. Find the distance from each focus to the generic point. Show that the sum of the distances is 2a.

## Solution

Create the ellipse and a generic point on it.

```
In[1]: Clear[a, b, t];
e1 = Ellipse2D[{0, 0}, a, b, 0];
p1 = Point2D[e1[t]]
Out[1] Point2D[{a Cos[t], b Sin[t]}]
```

Construct the focus points of the ellipse.

```
In[2]: \{ f1, f2 \} = Foci2D[e1]Out[2] \{ Point2D[\{\sqrt{a^2 - b^2}, 0\}], Point2D[\{-\sqrt{a^2 - b^2}, 0\}] \}
```

Find the sum of the distances from the generic point to the foci.

```
In[3]: suml = Distance2D[pl, fl] + Distance2D[pl, f2]Out[3] \sqrt{\left(-\sqrt{a^2 - b^2} + a \cos[t]\right)^2 + b^2 \sin[t]^2} + \sqrt{\left(\sqrt{a^2 - b^2} + a \cos[t]\right)^2 + b^2 \sin[t]^2}
```

Work on the expressions under the radicals.

```
In[4]: \{el, e2\} = Map[Expand[# /. Sin[t]^2 -> 1 - Cos[t]^2]\&, \\ \{sum1[[1, 1]], sum1[[2, 1]]\}\}Out[4] \{a^2 - 2 a \sqrt{a^2 - b^2} Cos[t] + a^2 Cos[t]^2 - b^2 Cos[t]^2, \\ a^2 + 2 a \sqrt{a^2 - b^2} Cos[t] + a^2 Cos[t]^2 - b^2 Cos[t]^2\}
```

This shows that both expressions factor into perfect squares.

Replace the expressions under the radicals with the equivalent perfect square expressions.

```
In[6]: sum2 = Sqrt[e3] + Sqrt[e4]
Out[6] \sqrt{\left(a - \sqrt{a^2 - b^2} \cos[t]\right)^2} + \sqrt{\left(a + \sqrt{a^2 - b^2} \cos[t]\right)^2}
```

Since a > b > 0, both expressions are clearly the square root of a squared positive number, which simply reduces to the positive number itself.

```
In[7]: Clear[E1];
    sum2 /. Sqrt[E1_^2] -> E1
Out[7] 2 a
```

## elllen.nb

#### Length of Ellipse Focal Chord

#### **Exploration**

Prove that the length of the focal chord of an ellipse is  $2b^2/a$ , where a is the length of the semi-major axis and b is the length of the semi-minor axis.

## Approach

Construct an ellipse in standard position. Construct a line perpendicular to the axis of the ellipse through one of the focal points (the line containing the focal chord). Compute the distance between the points of intersection of the ellipse and the line.

## Solution

Create the ellipse.

In[1]: Clear[a1, b1]; e1 = Ellipse2D[{0, 0}, a1, b1, 0];

Construct one of the focal points.

In[2]: fpt = First[Foci2D[e1]] $Out[2] Point2D[\{\sqrt{al^2 - bl^2}, 0\}]$ 

Construct a line perpendicular to the x-axis through the focus.

In[3]: fln = Line2D[fpt, Infinity] $Out[3] Line2D[1, 0, -\sqrt{al^2 - bl^2}]$ 

Intersect the line with the ellipse.

```
In[4]: pts = Points2D[fln, e1]
```

 $\textit{Out[4]} \hspace{0.1cm} \big\{ \texttt{Point2D}\big[ \big\{ \sqrt{\texttt{a1}^2 - \texttt{b1}^2} \hspace{0.1cm}, \hspace{0.1cm} - \frac{\texttt{b1}^2}{\texttt{a1}} \big\} \big] \hspace{0.1cm}, \hspace{0.1cm} \texttt{Point2D}\big[ \big\{ \sqrt{\texttt{a1}^2 - \texttt{b1}^2} \hspace{0.1cm}, \hspace{0.1cm} \frac{\texttt{b1}^2}{\texttt{a1}} \big\} \big] \big\}$ 

The length of the focal chord is the distance between the intersection points.

```
In[5]: d = Distance2D[Sequence @@ pts]
Out[5] 2 \sqrt{\frac{b1^4}{a1^2}}
```

Notice that since a > 0 and b > 0 the solution reduces to  $2b^2/a$ .

# ellrad.nb

## Apoapsis and Periapsis of an Ellipse

## **Exploration**

Show that the greatest, *apoapsis*, and least, *periapsis*, radial distance of a point on an ellipse as measured from a focus point is given by r = a(1 + e) and r = a(1 - e), respectively, where e is the eccentricity and a is the length of the semi-major axis of the ellipse.

### Approach

Create a standard ellipse centered at the origin and create an expression representing the distance from a focus point to a point on the ellipse (in terms of the eccentricity and semimajor axis). Find the parameter value on the ellipse where the distance is a minimum or a maximum.

## Solution

The eccentricity is given by  $e = \sqrt{a^2 - b^2}/a$ ; therefore,  $\sqrt{a^2 - b^2} = ea$ . Find the focus points and use this substitution.

```
In[1]: Clear[a, b, e];
e1 = Ellipse2D[{0, 0}, a, b, 0];
fpts1 = Foci2D[e1] /. Sqrt[a<sup>2</sup>-b<sup>2</sup>] -> a*e
Out[1] {Point2D[{ae, 0}], Point2D[{-ae, 0}]}
```

Find a general point on the ellipse in terms parameter t.

```
In[2]: Clear[t];
    pt = Point2D[e1[t]]
Out[2] Point2D[{a Cos[t], b Sin[t]}]
```

Solve for b in terms of a and e, where b > 0.

```
In[3]: ans = Solve[Sqrt[a^2 - b^2] / a == e, b]Out[3] \left\{ \left\{ b \rightarrow -a \sqrt{1 - e^2} \right\}, \left\{ b \rightarrow a \sqrt{1 - e^2} \right\} \right\}
```

Determine the distance, d, from the point to the focus.

```
In [4]: d = Distance2D[fpts1[[1]], pt] /. ans[[2]] // Simplify
Out [4] \sqrt{a^2 (-1 + e \cos[t])^2}
```

The maximum value of d occurs when  $\cos(\theta) = -1$ , or  $\theta = \pi$ ; the minimum value of d occurs when  $\cos(\theta) = 1$ , or  $\theta = 0$ .

 $In[5]: {apoapsis, periapsis} = Map[(d /. #)&, {Cos[t] -> -1, Cos[t] -> 1}]$ Out[5] { $\sqrt{a^2 (-1-e)^2}$ ,  $\sqrt{a^2 (-1+e)^2}$ }

Since e < 1, the sign must be reversed outside the radical.

```
In[6]: Clear[E1];
    {apoapsis, periapsis} /.
        Sqrt[a^2*E1_^2] -> a* (-E1) // Factor
Out[6] {a (1+e), -a (-1+e)}
```

## ellsim.nb

## Similar Ellipses

## **Exploration**

All ellipses of equal eccentricity are essentially *similar* in that by a proper choice of scales (and axes) they can be made to coincide. Show this property is true for two ellipses of equal eccentricity centered at the origin.

## Approach

Construct two ellipses with equal eccentricity. Show that one can be scaled to coincide with the other.

## Solution

Construct the two ellipses by vertex points.

Scale the first to coincide with the second

```
In[2]: Scale2D[E1, a2 / a1]
Out[2] Ellipse2D[{0, 0}, a2, a2 \sqrt{1-el^2}, ArcTan[2a1, 0]]
```

## Discussion

This is a plot of a pair of similar ellipses.

In[3]: Sketch2D[{E1, E2} /. {a1 -> 1, a2 -> 2, e1 -> .75}];


## ellslp.nb

## Tangent to an Ellipse with Slope

#### **Exploration**

Show that the lines tangent to the ellipse  $x^2/a^2 + y^2/b^2 = 1$  with slope *m* are given by  $y = mx \pm \sqrt{a^2m^2 + b^2}$ .

#### Approach

Construct a line with slope m and use the function TangentLines2D[ln, quad] to construct the desired tangent lines.

#### Solution

Construct a line with slope m.

Construct the lines tangent to the ellipse and parallel to the line.

```
In[2]: Clear[a, b];
tln = TangentLines2D[11, e1 = Ellipse2D[{0, 0}, a, b, 0]]
Out[2] {Line2D[m, -1, -\sqrt{b^2 + a^2 m^2}], Line2D[m, -1, \sqrt{b^2 + a^2 m^2}]
```

Show the lines in equation form.

```
In[3]: Map[Equation2D[#, {x, y}]\&, tln]
Out[3] {-\sqrt{b^2 + a^2 m^2} + m x - y == 0, \sqrt{b^2 + a^2 m^2} + m x - y == 0}
```

## Discussion

Plot a numerical example.

## eqarea.nb

#### **Equal Areas Point**

#### **Exploration**

Given  $\triangle ABC$  with vertices  $A(x_A, x_B)$ ,  $B(x_B, y_B)$  and  $C(x_C, y_C)$  show that there are four positions of a point  $P_n(x, y)$  such that  $\triangle APB$ ,  $\triangle APC$  and  $\triangle BPC$  have equal areas. The coordinates of  $P_n$  are given by

 $\begin{array}{l} P_0(\left(x_A + x_B + x_C\right)/3, \left(y_A + y_B + y_C\right)/3) \\ P_1(-x_A + x_B + x_C, -y_A + y_B + y_C) \\ P_2(+x_A - x_B + x_C, +y_A - y_B + y_C) \\ P_3(+x_A + x_B - x_C, +y_A + y_B - y_C). \end{array}$ 

 $P_0$  is the centroid of  $\triangle ABC$  and  $\triangle P_1P_2P_3$ .  $\triangle ABC$  connects the midpoints of the sides of  $\triangle P_1P_2P_3$ .

#### Approach

Construct the geometry and solve a system of equations that equates the areas of the three triangles. Compare the centroid and midpoints as specified.

#### Solution

Create the points.

```
In[1]: Clear[xA, yA, xB, yB, xC, yC, x, y];
A1 = Point2D[xA, yA];
B1 = Point2D[xB, yB];
C1 = Point2D[xC, yC];
P = Point2D[x, y];
```

Compute the areas of the triangles.

Form equations by equating the areas (squared). Squaring is required because the area calculation may produce a symbolically negative number for the area.

```
In[3]: \{eq1 = a1^{2} = a2^{2}, eq2 = a2^{2} = a3^{2}\}

Out [3] \{\frac{1}{4} (xAy - xBy - xyA + xByA + xyB - xAyB)^{2} = = \frac{1}{4} (xAy - xCy - xyA + xCyA + xyC - xAyC)^{2}, \frac{1}{4} (xAy - xCy - xyA + xCyA + xyC - xAyC)^{2} = = \frac{1}{4} (xBy - xCy - xyB + xCyB + xyC - xByC)^{2}\}
```

Solve the system of equations.

```
\begin{split} &In[4]: \text{ ans = Solve[{eq1, eq2}, {x, y}]} \\ &Out[4] \; \left\{ \left\{ x \rightarrow xA + xB - xC, \; y \rightarrow yA + yB - yC \right\}, \; \left\{ x \rightarrow xA - xB + xC, \; y \rightarrow yA - yB + yC \right\}, \\ & \quad \left\{ x \rightarrow -xA + xB + xC, \; y \rightarrow -yA + yB + yC \right\}, \; \left\{ x \rightarrow \frac{1}{3} \; \left( xA + xB + xC \right), \; y \rightarrow \frac{1}{3} \; \left( yA + yB + yC \right) \right\} \right\} \end{split}
```

Construct points at the solutions.

Show that  $P_0$  is the centroid of  $\triangle ABC$  and  $\triangle P_1P_2P_3$ .

```
In[6]: Point2D[Triangle2D[A1, B1, C1], Centroid2D]

Out[6] Point2D[\{\frac{1}{3} (xA + xB + xC), \frac{1}{3} (yA + yB + yC)\}]

In[7]: Point2D[Triangle2D[P1, P2, P3], Centroid2D]

Out[7] Point2D[\{\frac{1}{3} (xA + xB + xC), \frac{1}{3} (yA + yB + yC)\}]
```

Show that  $\triangle ABC$  connects the midpoints of the sides of  $\triangle P_1 P_2 P_3$ .

```
In[8]: {Point2D[P1, P2], Point2D[P1, P3], Point2D[P2, P3]}
Out[8] {Point2D[{xC, yC}], Point2D[{xB, yB}], Point2D[{xA, yA}]}
```

## Discussion

This is a plot of a numerical example.



# eyeball.nb

### Eyeball Theorem

## Exploration



The tangents to each of two circles from the center of the other are drawn as shown in the figure. Prove that the chords illustrated are equal in length.

## Approach

Construct the chords and compare their lengths.

## Solution

Without loss of generality, scale the circles so that the distance between the centers is 1. Position them at the origin and along the positive x-axis.

```
In[1]: Clear[r1, r2];
c1 = Circle2D[{0, 0}, r1];
c2 = Circle2D[{1, 0}, r2];
l12 = TangentLines2D[Point2D[c1], c2];
l21 = TangentLines2D[Point2D[c2], c1];
```

Compute the tangent points.

```
In[2]: pt1 = TangentPoints2D[Point2D[c1], c2];
    pt2 = TangentPoints2D[Point2D[c2], c1];
```

Show that (half) the heights of the segments are equal

```
In[3]: sin1 = YCoordinate2D[pt1[[1]]] /
    Distance2D[Point2D[0, 0], pt1[[1]]];
    sin2 = YCoordinate2D[pt2[[2]]] /
    Distance2D[Point2D[1, 0], pt2[[2]]];
    {h1 = Simplify[r1 * sin1],
    h2 = Simplify[r2 * sin2]}
Out[3] {r1 r2, r1 r2}
```

#### Discussion

This is a plot of a numerical example.

```
In[4]: example = {r1 -> 0.25, r2 -> 0.375};
    Sketch2D[
        {c1, c2, 112, 121, pt1, pt2} /.
        example,
        PlotRange -> {{-1/2, 2}, {-1, 1}};
```



## gergonne.nb

#### Gergonne Point of a Triangle

#### Exploration

Let  $Q_{12}$ ,  $Q_{13}$  and  $Q_{23}$  be the points of contact of the inscribed circle of  $\triangle P_1 P_2 P_3$  with sides  $L_{12}$ ,  $L_{13}$  and  $L_{23}$ , respectively. Show that lines  $P_1 Q_{23}$ ,  $P_2 Q_{13}$  and  $P_3 Q_{12}$  are concurrent. The point of concurrency is called the Gergonne Point of the triangle after J.D. Gergonne (1771–1859), founder-editor of the mathematics journal Annales de Mathematiques.

#### Approach

Create the triangle in a simplified position and construct the inscribed circle. Construct the tangency points and the prescribed lines. Show that the lines are concurrent.

#### Solution

Without loss of generality, create the triangle's vertex points and the triangle itself in a convenient position.

```
In[1]: Clear[a, b];
P1 = Point2D[0, 0];
P2 = Point2D[a, b];
P3 = Point2D[1, 0];
T1 = Triangle2D[P1, P2, P3];
```

Construct the circle inscribed in the triangle.

```
In[2]: C1 = Circle2D[T1, Inscribed2D] // FullSimplify;
```

Construct the lines through the sides of the triangle.

```
In[3]: {L12 = Line2D[P1, P2], L13 = Line2D[P1, P3], L23 = Line2D[P2, P3]}
Out[3] {Line2D[-b, a, 0], Line2D[0, 1, 0], Line2D[b, 1 - a, -b]}
```

Construct tangency points which are the poles of the sides with respect to the inscribed circle.

```
In[4]: {Q12, Q13, Q23} =
    Map[FullSimplify,
        {Point2D[L12, C1], Point2D[L13, C1], Point2D[L23, C1]}];
```

Construct the lines defining the Gergonne point.

```
In[5]: {L1, L2, L3} =
    Map[FullSimplify,
    {Line2D[P1, Q23], Line2D[P2, Q13], Line2D[P3, Q12]}];
```

The three lines are concurrent if the determinant of their coefficients is zero.

```
In[6]: Det[{List@@L1, List@@L2, List@@L3}] // FullSimplify
Out[6] 0
```

### Discussion

This plots a numerical example with specific points.



## heron.nb

#### Heron's Formula

#### **Exploration**

Show that the area, K, of a  $\triangle ABC$  is given by

$$K = \sqrt{s(s-a)(s-b)(s-c)}$$

where the semi-perimeter s = (a + b + c)/2 and a, b and c are the lengths of the sides.

#### Approach

In a  $\triangle ABC$  with side lengths a, b and c, derive an expression for  $\cos A$  (the cosine of the angle at vertex A of the triangle) using the Law of Cosines. Using the identity  $(\sin A)^2 = 1 - (\cos A)^2$  the area can be computed using  $K = (1/2) bc \sin A$ . Simplify the resulting expression for the area, K, to Heron's formula.

#### Solution

Find an expression for the  $\cos A$  using the Law of Cosines.

Find an expression for the  $\sin A$  using the previous expression for  $\cos A$ . Use the positive result.

```
In[2]: \mathbf{sA} = Solve[(sinA^2 + cosA^2 = = 1) /. cA, sinA] // LastOut[2] \left\{ sinA \rightarrow \frac{1}{2} \sqrt{2 + \frac{2a^2}{b^2} + \frac{2a^2}{c^2} - \frac{a^4}{b^2c^2} - \frac{b^2}{c^2} - \frac{c^2}{b^2}} \right\}
```

Compute the area of the triangle from one-half the product of the base and height.

In [3]: K1 = b \* c \* sinA / 2 /. sA // FullSimplify Out [3]  $\frac{1}{4}$  b c  $\sqrt{-\frac{(-a+b-c)(a+b-c)(-a+b+c)(a+b+c)}{b^2 c^2}}$ 

Simplify to Heron's formula. The following steps were computed using *Mathematica* Version 3.0.1. Version 4.0 produces slightly different results that are algebraically equivalent.

```
In[4]: K2 = K1 /. Sqrt[E1_] :> Sqrt[Factor[E1]]
Out[4] \frac{1}{4} b c \sqrt{-\frac{(-a+b-c)(a+b-c)(-a+b+c)(a+b+c)}{b^2 c^2}}
In[5]: K3 = K2 /. Sqrt[-E1_/(b^2 * c^2)] :> Sqrt[-E1]/(b * c)
Out[5] \frac{1}{4} \sqrt{-(-a+b-c)(a+b-c)(-a+b+c)(a+b+c)}
In[6]: K4 = K3 //. \{a+b-2 * s-c, b+c-2 * s-a, -a-c-2 + (2 * s-b)\} // FullSimplify
Out[6] \sqrt{s (-a+s)(-b+s)(-c+s)}
```

## hyp2a.nb

#### Focal Distances of a Hyperbola

#### **Exploration**

Show that the difference of the distances from the two foci to any point on a hyperbola is 2a, where a is the length of the semi-transverse axis.

#### Approach

Construct a generic point on a hyperbola. Construct the two foci of the hyperbola. Find the distance from each focus to the generic point. Show that the difference of the distances is 2a.

#### Solution

Create the hyperbola and a generic point on it.

```
In[1]: Clear[a, b, t];
h1 = Hyperbola2D[{0, 0}, a, b, 0];
p1 = Point2D[a * Cosh[t], b * Sinh[t]]
Out[1] Point2D[{a Cosh[t], b Sinh[t]}]
```

Create the focus points of the hyperbola.

```
In[2]: \{f1, f2\} = Foci2D[h1]
Out[2] {Point2D[{\sqrt{a^2 + b^2}, 0}], Point2D[{-\sqrt{a^2 + b^2}, 0}]}
```

Compute the difference of the distances from the generic point to the foci.

In[3]: diff1 = Distance2D[p1, f2] - Distance2D[p1, f1] $Out[3] - \sqrt{\left(-\sqrt{a^2 + b^2} + a \operatorname{Cosh}[t]\right)^2 + b^2 \operatorname{Sinh}[t]^2} + \sqrt{\left(\sqrt{a^2 + b^2} + a \operatorname{Cosh}[t]\right)^2 + b^2 \operatorname{Sinh}[t]^2}$ 

Work on the expressions under the radicals.

```
In[4]: \{e1, e2\} = Map[Expand[# /. Sinh[t]^2 -> Cosh[t]^2 - 1]\&, \\ \{diff1[[1, 2, 1]], diff1[[2, 1]]\}\}Out[4] \{a^2 - 2a\sqrt{a^2 + b^2} Cosh[t] + a^2 Cosh[t]^2 + b^2 Cosh[t]^2, \\ a^2 + 2a\sqrt{a^2 + b^2} Cosh[t] + a^2 Cosh[t]^2 + b^2 Cosh[t]^2\}
```

This shows that both expressions factor into perfect squares.

Replace the expressions under the radicals with the equivalent perfect square expressions.

```
In[6]: diff2 = -Sqrt[e3] + Sqrt[e4]Out[6] - \sqrt{(a - \sqrt{a^2 + b^2} \operatorname{Cosh}[t])^2} + \sqrt{(a + \sqrt{a^2 + b^2} \operatorname{Cosh}[t])^2}
```

Since a > 0, b > 0 and  $\cosh(\theta) \ge 1$  the expression under the radicals are reduced as follows.

```
In[7]: Clear[E1, E2];
diff2 /. {-Sqrt[E1_^2] + Sqrt[E2_^2] -> - (-E1) + E2}
Out[7] 2 a
```

## hyp4pts.nb

#### Equilateral Hyperbolas

#### **Exploration**

Describe a method for finding the equilateral hyperbola(s) passing through four points. Show that the technique produces the correct results for the points (2, 1), (-1, 1), (-2, -1) and (4, -3) by plotting the hyperbola(s) and the four points.

### Approach

Form a quadratic, parameterized by the variable k, representing the pencil of quadratics passing through the four points. The first and third coefficients of the quadratic, a and c, must satisfy the relationship a = -c, if the quadratic represents an equilateral hyperbola. Solve the equation for k.

#### Solution

This is a function that implements the approach.

#### Discussion

Here's the plot of the solution for the four points specified.



# hyparea.nb

Areas Related to Hyperbolas

Exploration



Referring to the figures, use calculus to verify that the areas between two parameters  $t_1$  and  $t_2$  of a segment and a sector of a hyperbola are given by

$$A_{segment} = \frac{ab}{2} \left( \sinh(s(t_2 - t_1)) - s(t_2 - t_1) \right)$$
$$A_{sector} = \frac{abs}{2} \left( t_2 - t_1 \right)$$

where a and b are the lengths of the semi-transverse and semi-conjugate axes, respectively,  $s = \cosh^{-1}(e)$  and e is the eccentricity of the hyperbola (assuming the parameterization Descarta<sub>2D</sub> uses for a hyperbola).

#### Approach

Find the coordinates of  $x_1$  and  $x_2$ , the coordinates of the ends of the infinitesimal rectangle. Integrate  $(x_2 - x_1) dx$  from  $y_1$  to  $y_2$  to find the area of the segment. Find the area of the  $\triangle OP_1P_2$  from its vertex points. Subtract the area of the segment from the area of the triangle to find the area of the sector.

#### Solution

The x-coordinate of a point on the hyperbola (found by solving  $x^2/a^2 - y^2/b^2 = 1$  for x) in terms of the y-coordinate.

In[1]: Clear[x, y, a, b];  
X1 = a \* Sqrt[1 + y^2/b^2]  
Out[1] a 
$$\sqrt{1 + \frac{y^2}{b^2}}$$

The x-coordinate of a point on a line between  $(x_1, y_1)$  and  $(x_2, y_2)$ . This is found by intersecting a horizontal line through the point on the hyperbola with the line between  $P_1$  and  $P_2$ . This step and the one immediately following show the results computed by *Mathematica* Version 3.0.1. Version 4.0 produces slightly different results that are algebraically equivalent.

L is the length of the horizontal line segment between the hyperbola and the line through  $P_1$  and  $P_2$ .

In [3]: **L** = **X2** - **X1** // Fullsimplify  
Out [3] -a 
$$\sqrt{1 + \frac{y^2}{b^2}} + \frac{x^2(-y+y1) + x1(y-y2)}{y1-y2}$$

Find the indefinite integral  $L \bigtriangleup y$ , which represents an infinitesimal rectangular area.

```
In[4]: II = Integrate[L, y] // FullSimplify
Out[4] \frac{y \left(-x2y + 2x2y1 + x1(y - 2y2) + a\sqrt{1 + \frac{y^2}{b^2}} (-y1 + y2)\right) + ab(-y1 + y2) ArcSinh[\frac{y}{b}]}{2(y1 - y2)}
```

Find the area of the hyperbolic segment between the chord and the hyperbola by evaluating the integral at the vertical limits. Simplify.

```
In[5]: Al = (I1 /. y \rightarrow y2) - (I1 /. y \rightarrow y1) // Fullsimplify
Out[5] \frac{1}{2} \left( a y1 \sqrt{1 + \frac{y1^2}{b^2}} - (x1 + x2) (y1 - y2) - a y2 \sqrt{1 + \frac{y2^2}{b^2}} + a b \left( ArcSinh \left[ \frac{y1}{b} \right] - ArcSinh \left[ \frac{y2}{b} \right] \right) \right)
In[6]: A2 = A1 //. \{ a * Sqrt[1 + y1^2/b^2] \rightarrow x1, a * Sqrt[1 + y2^2/b^2] \rightarrow x2 \}
```

$$\textit{Out[6]} \quad \frac{1}{2} \left( \texttt{x1 y1} - (\texttt{x1} + \texttt{x2}) \ (\texttt{y1} - \texttt{y2}) - \texttt{x2 y2} + \texttt{ab} \left( \texttt{ArcSinh} \left[ \frac{\texttt{y1}}{\texttt{b}} \right] - \texttt{ArcSinh} \left[ \frac{\texttt{y2}}{\texttt{b}} \right] \right) \right)$$

Create the hyperbola.

In[7]: H1 = Hyperbola2D[{0, 0}, a, b, 0];

Find the coordinates of a point at a general parameter t on a hyperbola.

```
In[8]: Clear[s];
    P = H1[t] /. ArcCosh[Sqrt[a^2+b^2]/a] -> s
Out[8] {a Cosh[st], b Sinh[st]}
```

Simplify.

```
In[9]: Clear[t1, t2];
A3 = A2 //. {
    x1 -> (P[[1]] /. t -> t1),
    x2 -> (P[[1]] /. t -> t2),
    y1 -> (P[[2]] /. t -> t1),
    y2 -> (P[[2]] /. t -> t2)} // FullSimplify
```

 $Out[9] - \frac{1}{2} ab(-ArcSinh[Sinh[st1]] + ArcSinh[Sinh[st2]] + Sinh[s(t1 - t2)])$ 

```
In[10]: Clear[E1];
A4 = A3 /.
ArcSinh[Sinh[E1_]] -> E1 // FullSimplify
Out[10] - 1/2 ab (s (-t1 + t2) + Sinh[s (t1 - t2)])
In[11]: AreaSegment = A4 /.
Sinh[s (t1 - t2)] -> -Sinh[s (t2 - t1)]
Out[11] - 1/2 ab (s (-t1 + t2) - Sinh[s (-t1 + t2)])
```

Find the area of the triangle  $OP_1P_2$ .

```
In[12]: AreaTriangle = Area2D[
    Triangle2D[{0, 0}, {x1, y1}, {x2, y2}]] /. {
    x1 -> (P[[1]] /. t -> t1),
    x2 -> (P[[1]] /. t -> t2),
    y1 -> (P[[2]] /. t -> t1),
    y2 -> (P[[2]] /. t -> t2)} // FullSimplify
Out[12] - 1/2 abSinh[s (t1-t2)]
```

The area of the sector is the difference.

```
In[13]: AreaSector = AreaTriangle - AreaSegment // Simplify
Out[13] 1/2 abs (-t1 + t2)
```

## hypeccen.nb

### Eccentricities of Conjugate Hyperbolas

#### **Exploration**

Show that if  $e_1$  and  $e_2$  are the eccentricities of a hyperbola and its conjugate, then

$$\frac{1}{e_1^2} + \frac{1}{e_2^2} = 1$$

### Approach

Create the hyperbolas, compute their eccentricities and verify the relationship.

### Solution

Create a hyperbola and its conjugate.

```
In[1]: Clear[a, b];
h1 = Hyperbola2D[{0, 0}, a, b, 0];
h2 = Hyperbola2D[h1, Conjugate2D];
```

Compute the eccentricities of the hyperbolas.

```
In[2]: \{el = Eccentricity2D[h1], e2 = Eccentricity2D[h2]\}Out[2] \{\frac{\sqrt{a^2 + b^2}}{a}, \frac{\sqrt{a^2 + b^2}}{b}\}
```

Verify the relationship.

```
In[3]: 1/e1^2+1/e2^2 // Simplify
Out[3] 1
```

## hypfd.nb

#### Hyperbola from Focus and Directrix

#### **Exploration**

Show that the hyperbola with focus  $F(x_1, y_1)$ , directrix line  $L \equiv px + qy + r = 0$  and eccentricity e > 1, is defined by the constants

$$h = x_1 - \frac{paeD}{d}, \quad k = y_1 - \frac{qaeD}{d},$$
$$a = d\frac{e}{(e^2 - 1)}, \quad b = a\sqrt{e^2 - 1}, \quad \theta = \tan^{-1}(p, q),$$

where

$$d = \sqrt{\frac{(px_1 + qy_1 + r)^2}{p^2 + q^2}}$$
 and  $D = \frac{px_1 + qy_1 + r}{p^2 + q^2}$ .

#### Approach

Apply the definition of a hyperbola to the supplied focus and directrix for a general point (x, y) and show that the derived locus is a hyperbola.

#### Solution

The rotation angle of the hyperbola is the angle the line perpendicular to L makes with the +x-axis (in Mathematica ArcTan[p, q] is ArcTan[q/p], the first form takes into account the quadrant of the point (p,q)).

```
In[1]: Clear[p,q,r];
L = Line2D[p,q,r];
theta = Angle2D[Line2D[0,1,0], Line2D[Point2D[0,0],L]];
theta // Simplify
Out[1] ArcTan[ <sup>q</sup>/<sub>p</sub>]
```

Now we must show that the lengths a and b are given by the formulas. In standard position the distance from the focus of an ellipse to its directrix is given by d = ae - a/e. Solving for a gives the following.

```
In[2]: Clear[d, a, e];
Solve[d == a * e - a / e, a] // Simplify
Out[2] \left\{ \left\{ a \rightarrow \frac{de}{-1 + e^2} \right\} \right\}
```

Also, the eccentricity is given by  $e = \sqrt{a^2 + b^2}/a$  and solving for b gives (take the positive result).

```
 \begin{split} & \text{In[3]: Solve[e == Sqrt[a^2 + b^2] / a, b]} \\ & \text{Out[3]} \; \left\{ \left\{ b \to -a \; \sqrt{-1 + e^2} \; \right\}, \; \left\{ b \to a \; \sqrt{-1 + e^2} \; \right\} \right\} \end{split}
```

The eccentricity is the ratio of the distance from a general point to the focus to the distance to the directrix.

```
In[4]: Clear[x1, y1, x, y]; 
F = Point2D[x1, y1]; 
P = Point2D[x, y]; 
{dF = Distance2D[P, F], 
dL = Distance2D[P, L]} 
Out[4] {<math>\sqrt{(x-x1)^{2} + (y-y1)^{2}}, \sqrt{\frac{(r+px+qy)^{2}}{p^{2}+q^{2}}}}
```

Form the equation for the eccentricity squared.

```
In[5]: eql = e^2 * dL^2 - dF^2 // Expand // Together
Out [5] \frac{1}{p^2 + q^2} (e^2 r^2 + 2 e^2 p r x - p^2 x^2 + e^2 p^2 x^2 - q^2 x^2 + 2 p^2 x x1 + 2 q^2 x x1 - p^2 x1^2 - q^2 x1^2 + 2 e^2 q r y + 2 e^2 p q x y - p^2 y^2 - q^2 y^2 + e^2 q^2 y^2 + 2 p^2 y y1 + 2 q^2 y y1 - p^2 y1^2 - q^2 y1^2)
```

Find the coordinates  $(h_1, k_1)$  of the center of the quadratic.

```
In[6]: \{h1, k1\} = Coordinates2D[

Point2D[

Q1 = Quadratic2D[eq1, {x, y}] // Simplify] // Simplify

Out[6] \{ \frac{-(p^2 + q^2) x1 + e^2 (q^2 x1 - p (r + q y1))}{(-1 + e^2) (p^2 + q^2)}, - \frac{(p^2 + q^2) y1 + e^2 (q (r + p x1) - p^2 y1)}{(-1 + e^2) (p^2 + q^2)} \}
```

Find the coordinates of the center using the formula provided.

```
 \begin{array}{l} In [7]: \mbox{Clear[D1];} \\ \mbox{ } \{h2, k2\} = \{x1 - p*a*e*D1/d, \ y1 - q*a*e*D1/d\} \ //. \\ \mbox{ } \{a -> d*e/(e^2 - 1), \\ \mbox{ } b -> a*Sqrt[e^2 - 1], \\ \mbox{ } d -> Sqrt[(p*x1 + q*y1 + r)^2/(p^2 + q^2)], \\ \mbox{ } D1 -> (p*x1 + q*y1 + r)/(p^2 + q^2)\} \\ Out [7] \ \left\{x1 - \frac{e^2 \ p \ (r + p \ x1 + q \ y1)}{(-1 + e^2) \ (p^2 + q^2)}, \ y1 - \frac{e^2 \ q \ (r + p \ x1 + q \ y1)}{(-1 + e^2) \ (p^2 + q^2)}\right\} \end{array}
```

This shows that the center indeed has the same coordinates as the point from the formula.

```
In[8]: {h1 - h2, k1 - k2} // Simplify
Out[8] {0, 0}
```

#### Discussion

An example showing the construction of a hyperbola from its focus, directrix and eccentricity.

```
In[9]: focus1 = Point2D[\{1/2, 1\}];
directrix1 = Line2D[5, 8, -15];
eccentricity1 = 5/4;
hyperbola1 = Hyperbola2D[focus1, directrix1, eccentricity1]
Out[9] Hyperbola2D[\{\frac{107}{89}, \frac{189}{89}\}, \frac{10}{\sqrt{89}}, \frac{15}{2\sqrt{89}}, ArcTan[\frac{8}{5}]]
In[10]: Sketch2D[\{focus1, directrix1, hyperbola1\}, CurveLength2D -> 5];
```



## hypinv.nb

#### **Rectangular Hyperbola Distances**

#### **Exploration**

Show that the distance of any point on a rectangular hyperbola from its center varies inversely as the perpendicular distance from its polar to the center.

#### Approach

Construct a generic point on a rectangular hyperbola and compare the appropriate distances.

#### Solution

Create a generic point on a rectangular hyperbola.

```
In[1]: Clear[a, t];
h1 = Hyperbola2D[{0, 0}, a, a, 0];
p1 = Point2D[a * Cosh[t], a * Sinh[t]]
Out[1] Point2D[{a Cosh[t], a Sinh[t]}]
```

Compute the distances.

```
In[2]: p0 = Point2D[0, 0];
{D1, D2} =
{Distance2D[p0, p1],
Distance2D[p0, 11 = Line2D[p1, h1]]} // Simplify
Out[2] {\sqrt{a^2 Cosh[2 t]}, \sqrt{a^2 Sech[2 t]}}
```

Use a trigonometric identity.

```
In[3]: 1/Sech[2t] //Simplify
Out[3] Cosh[2t]
```

Therefore, since  $D_1D_2$  is a constant,  $D_1$  varies inversely as  $D_2$ .

```
In[4]: Clear[E1];
D1 * D2 //. {
    Sqrt[a^2 * E1_] -> a * Sqrt[E1],
    Sqrt[Cosh[E1_]] * Sqrt[Sech[E1_]] -> 1}
```

```
Out[4] a^2
```

## Discussion

This is a plot of a numerical example of the geometric objects.

```
In[5]: Sketch2D[{h1, p1, p0, l1} /. {a -> 1, t -> 0.5}];
```



## hyplen.nb

#### Length of Hyperbola Focal Chord

#### **Exploration**

Prove that the length of the focal chord of a hyperbola is  $2b^2/a$ , where a is the length of the semi-transverse axis and b is the length of the semi-conjugate axis.

#### Approach

Construct a hyperbola in standard position. Construct a line perpendicular to the axis of the hyperbola through one of the focal points (the line containing the focal chord). Compute the distance between the points of intersection of the hyperbola and the line.

#### Solution

Create the hyperbola.

```
In[1]: Clear[a1, b1];
h1 = Hyperbola2D[{0, 0}, a1, b1, 0];
```

Construct one of the focal points.

```
In[2]: fpt = First[Foci2D[h1]]Out[2] Point2D[\{\sqrt{al^2 + bl^2}, 0\}]
```

Construct a line perpendicular to the x-axis through the focus.

```
In[3]: fln = Line2D[fpt, Line2D[0, 1, 0], Perpendicular2D]Out[3] Line2D[1, 0, -\sqrt{al^2 + bl^2}]
```

Intersect the line with the hyperbola.

```
In[4]: pts = Points2D[fln, h1]
```

 $\textit{Out[4]} \quad \left\{\texttt{Point2D}\left[\left\{\sqrt{\texttt{al}^2+\texttt{bl}^2}\text{ , }-\frac{\texttt{bl}^2}{\texttt{al}}\right\}\right]\text{, }\texttt{Point2D}\left[\left\{\sqrt{\texttt{al}^2+\texttt{bl}^2}\text{ , }\frac{\texttt{bl}^2}{\texttt{al}}\right\}\right]\right\}$ 

The length of the focal chord is the distance between the intersection points.

```
In [5]: d1 = Distance2D[Sequence @@ pts]
Out [5] 2\sqrt{\frac{b1^4}{a1^2}}
```

Notice that since a > 0 and b > 0 the solution reduces to  $2b^2/a$ .

## hypslp.nb

#### Tangent to a Hyperbola with Given Slope

#### **Exploration**

Show that the lines tangent to the hyperbola  $x^2/a^2 - y^2/b^2 = 1$  with slope *m* are given by  $y = mx \pm \sqrt{a^2m^2 - b^2}$ .

### Approach

Construct a line with slope m and use TangentLines2D[ln, quad] to construct the desired tangent lines.

#### Solution

Construct a line with slope m.

Construct the lines tangent to the hyperbola and parallel to the line.

```
In[2]: Clear[a, b];
tln = TangentLines2D[11, h1 = Hyperbola2D[{0, 0}, a, b, 0]]
Out[2] {Line2D[m, -1, -\sqrt{-b^2 + a^2 m^2}], Line2D[m, -1, \sqrt{-b^2 + a^2 m^2}]}
```

Show the lines in equation form.

```
In [3]: Map[Equation2D[#, {x, y}]&, tln]
Out[3] {-\sqrt{-b^2 + a^2 m^2} + m x - y == 0, \sqrt{-b^2 + a^2 m^2} + m x - y == 0}
```

## Discussion

This is a plot of a numerical example.

In [4]: Sketch2D[{tln, h1} /. {m -> 1, a -> 2, b -> 1}];



# hyptrig.nb

## **Trigonometric Parametric Equations**

#### **Exploration**

Show that the parametric equations

 $x = a \sec(\theta)$  $y = b \tan(\theta)$ 

represent the hyperbola

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1.$$

### Approach

Demonstrate that the parametric equations satisfy the equation of the hyperbola.

#### Solution

Substitute the parametric values into the equation and observe that the equation is satisfied.

# intrsct.nb

## Intersection of Lines in Intercept Form

## Exploration

Show that the point of intersection of the lines

$$\frac{x}{a} + \frac{y}{b} = 1$$
 and  $\frac{x}{b} + \frac{y}{a} = 1$ 

is

$$\bigg(\frac{ab}{(a+b)},\frac{ab}{(a+b)}\bigg).$$

#### Approach

Create the two lines and intersect them.

## Solution

Create the two lines.

Intersect the lines.

```
In[2]: Point2D[11, 12] // SimplifyOut[2] Point2D\left[\left\{\frac{a b}{a+b}, \frac{a b}{a+b}\right\}\right]
```

## Discussion

Notice that the formula cannot be used if  $a = \pm b$ , because in both cases the two lines are coincident. This limitation is more obvious if we do not simplify the equation for the point of intersection (the denominators are zero when  $a = \pm b$ ).

In[3]: Point2D[11, 12] Out[3] Point2D[ $\left\{\frac{-a^2b+ab^2}{-a^2+b^2}, \frac{-a^2b+ab^2}{-a^2+b^2}\right\}$ ]
### inverse.nb

#### Inversion

#### **Exploration**

A point P'(x', y') is said to be the *inverse* of a point P(x, y) in the circle

$$C \equiv (x - h)^{2} + (y - k)^{2} = r^{2}$$

if points O(h, k), P and P' are collinear and  $|OP||OP'| = r^2$ . Using this definition show that A. The coordinates of P'(x', y') are

$$x' = h + \frac{r^2(x-h)}{(x-h)^2 + (y-k)^2}$$
 and  $y' = k + \frac{r^2(y-k)}{(x-h)^2 + (y-k)^2}$ .

B. If the circle of inversion is  $x^2 + y^2 = 1$ , the coordinates of P' are

$$x' = \frac{x}{x^2 + y^2}$$
 and  $y' = \frac{y}{x^2 + y^2}$ .

C. If the circle of inversion is  $x^2 + y^2 = 1$ , the inverse of the line  $L \equiv A_1x + B_1y + C_1 = 0$ , assuming L does not pass through the origin, is the circle

$$\left(x + \frac{A_1}{2C_1}\right)^2 + \left(y + \frac{B_1}{2C_1}\right)^2 = \frac{A_1^2 + B_1^2}{4C_1^2}.$$

D. If the circle of inversion is  $x^2 + y^2 = 1$ , the inverse of the line  $L \equiv A_1 x + B_1 y + C_1 = 0$ , assuming L passes through the origin  $(C_1 = 0)$ , is L itself.

E. If the circle of inversion is  $x^2 + y^2 = 1$ , the inverse of the circle  $(x - h_1)^2 + (y - k)^2 = r_1^2$  is

$$\left(x - \frac{h_1}{D}\right)^2 + \left(y - \frac{k_1}{D}\right)^2 = \frac{r_1^2}{D},$$

where  $D = h_1^2 + k_1^2 - r_1^2$ .

F. If the circle of inversion is  $x^2 + y^2 = 1$ , the inversion of  $C \equiv (x - h)^2 + (y - k)^2 = h_1^2 + k_1^2$ , which passes through the origin, is the line  $L \equiv 2h_1x + 2k_1y = 1$ . L is parallel to the tangent line to C through the origin. The equation of the tangent line is  $2h_1x + 2k_1y = 0$ .

G. Inversion is clearly a *non-rigid* transformation.

#### Approach

See the commentary below.

#### Solution

Use the definition of *inversion* to find the coordinates of a point (x, y) inverted in the circle  $(x-h)^2 + (y-k)^2 = r^2$ . This is the solution to proposition A as stated above.

```
 \begin{array}{l} In[1]: \mbox{ Clear[x, y, h, k, r];} \\ \mbox{ Coordinates2D[} \\ \mbox{ Point2D[, k],} \\ \mbox{ Point2D[h, k],} \\ \mbox{ Point2D[x, y],} \\ \mbox{ r^2/Sqrt[(x-h)^2 + (y-k)^2]]]} \\ Out[1] \ \left\{h + \frac{r^2 \ (-h+x)}{(-h+x)^2 + (-k+y)^2}, \ k + \frac{r^2 \ (-k+y)}{(-h+x)^2 + (-k+y)^2}\right\} \end{array}
```

Define a function for inverting coordinates.

$$In[2]: Inverse2D[{x_, y_}, Circle2D[{h_, k_}, r_]] := {h + (r^2 * (x - h)) / ((x - h)^2 + (y - k)^2), k + (r^2 * (y - k)) / ((x - h)^2 + (y - k)^2)}$$

Here's the inversion in a unit circle at the origin.

```
 \begin{array}{l} \mbox{In[3]: invPts = Inverse2D[{x, y}, Circle2D[{0, 0}, 1]]} \\ \label{eq:Inf} \mbox{Out[3] } \Big\{ \frac{x}{x^2 + y^2}, \ \frac{y}{x^2 + y^2} \Big\} \end{array}
```

Determine the inverse inversion equations. This is the solution to proposition B as stated above.

```
\begin{split} \text{In[4]: Clear[x1, y1];} \\ & \text{eqn1 = Solve[{x1, y1} == invPts, {x, y}]} \\ \text{Out[4]} \; \left\{ \left\{ x \to \frac{x1}{x1^2 + y1^2}, \; y \to \frac{y1}{x1^2 + y1^2} \right\} \right\} \end{split}
```

Find the inversion of a line.

```
In [5]: Clear[A1, B1, C1];
eq1 = A1 * x + B1 * y + C1 /. First[eqn1]
Out [5] C1 + \frac{A1 \times 1}{\times 1^2 + y1^2} + \frac{B1 \times 1}{\times 1^2 + y1^2}
```

Clear the denominators of the equations.

```
In[6]: eq2 = eq1 * (x1^2 + y1^2) // Simplify
Out[6] Al x1 + Bl y1 + C1 (x1<sup>2</sup> + y1<sup>2</sup>)
```

Determine the quadratic (a circle). This is the solution to proposition C as stated above.

```
In[7]: Circle2D[Quadratic2D[eq2, {x1, y1}]]Out[7] Circle2D[\left\{-\frac{A1}{2 C1}, -\frac{B1}{2 C1}\right\}, \frac{1}{2} \sqrt{\frac{A1^2 + B1^2}{C1^2}} \right]
```

Find a line passing through the center of inversion (0, 0).

```
In[8]: eq3 = A1 * x + B1 * y /. First[eqn1]
Out[8] \frac{A1 x1}{x1^2 + y1^2} + \frac{B1 y1}{x1^2 + y1^2}
```

Clear the denominator.

In[9]: eq4 = eq3 /. {x1^2 + y1^2 -> 1}
Out[9] Al x1 + Bl y1

The line inverts into itself. This is the solution to proposition D as stated above.

In[10]: Line2D[eq4, {x1, y1}]
Out[10] Line2D[A1, B1, 0]

Inversion of a circle.

```
In[11]: Clear[h1, k1, r1];
eq5 = (x - h1)^{2} + (y - k1)^{2} - r1^{2} /. First[eqn1]
Out[11] - r1^{2} + \left(-h1 + \frac{x1}{x1^{2} + y1^{2}}\right)^{2} + \left(-k1 + \frac{y1}{x1^{2} + y1^{2}}\right)^{2}
```

Clear the denominators.

```
In[12]: eq6 = eq5 * (x1^2 + y1^2)^2 // Together;
eq7 = eq6[[3]]
Out[12] -1 + 2 h1 x1 - h1<sup>2</sup> x1<sup>2</sup> - k1<sup>2</sup> x1<sup>2</sup> + r1<sup>2</sup> x1<sup>2</sup> + 2 k1 y1 - h1<sup>2</sup> y1<sup>2</sup> - k1<sup>2</sup> y1<sup>2</sup> + r1<sup>2</sup> y1<sup>2</sup>
```

Find the circle. If the resulting denominators are zero, then the circle passes through the center of inversion and the inversion is invalid. This is the solution to proposition E as stated above.

```
In[13]: Circle2D[Quadratic2D[eq7, {x1, y1}]] // Simplify
```

```
\textit{Out[13] Circle2D}\Big[\Big\{\frac{h1}{h1^2+k1^2-r1^2}\,,\,\,\frac{k1}{h1^2+k1^2-r1^2}\Big\}\,,\,\,\sqrt{\frac{r1^2}{\left(h1^2+k1^2-r1^2\right)^2}}\,\Big]
```

A circle not passing through the origin.

```
In[14]: eq8 = (x - h1)^{2} + (y - k1)^{2} - (h1^{2} + k1^{2}) /. First[eqn1] // SimplifyOut[14] \frac{1 - 2h1x1 - 2k1y1}{x1^{2} + y1^{2}}
```

Clear the denominator and find the line. This is the solution to the first part of proposition F as stated above.

```
In[15]: eq9 = Numerator[eq8];
Line2D[eq9, {x1, y1}]
Out[15] Line2D[-2h1, -2k1, 1]
```

The line is parallel to the tangent at the origin. This is the solution to the second part of proposition F as stated above.

```
In[16]: Line2D[Point2D[0, 0],
Circle2D[{h1, k1}, Sqrt[h1^2+k1^2]]]
Out[16] Line2D[-2 h1, -2 k1, 0]
```

### johnson.nb

#### Johnson's Congruent Circle Theorem

#### **Exploration**

Take any three circles  $C_1$ ,  $C_2$  and  $C_3$  which pass through the origin, have equal radii, r, and intersect in pairs in two distinct points (one of the points is, by construction, the origin). Prove that the circle passing through the other three points of intersection between the circles taken in pairs is congruent to the original three circles (that is, this circle has a radius of r).

#### Approach

Find the coordinates of the intersection points,  $P_1$ ,  $P_2$  and  $P_3$ . Use the circle through three points function to find Johnson's Circle. Show that the radius of this circle is r.

#### Solution

Without loss of generality, assume the circles have a radius of one and one of them has its center at (1,0). The centers of the other two circles can be written as functions of the angles the lines through (0,0) and the centers makes with the +x axis.

```
In[1]: Clear[t2, t3];
P1 = Point2D[1, 0];
P2 = Point2D[Cos[t2], Sin[t2]];
P3 = Point2D[Cos[t3], Sin[t3]];
```

Create the circles.

```
In[2]: C1 = Circle2D[P1, 1];
C2 = Circle2D[P2, 1];
C3 = Circle2D[P3, 1];
```

Intersect the first and second circle to find the intersection points. The head ImPoint2D is introduced to avoid failures during simplification when the coordinates of the points pass through a temporary phase involving complex numbers.

```
In[3]: pts12 = Points2D[C1, C2];
pts12 = Map[FullSimplify, Map[(ImPoint2D @@ #)&, pts12]];
pts12 = Map[(Point2D @@ #)&, pts12]
Out[3] \{Point2D[\{\frac{1}{2} Cot[\frac{t2}{2}] (Sin[t2] - \sqrt{Sin[t2]^2}), \frac{1}{2} (Sin[t2] - \sqrt{Sin[t2]^2})\}],
Point2D[\{\frac{1}{2} Cot[\frac{t2}{2}] (Sin[t2] + \sqrt{Sin[t2]^2}), \frac{1}{2} (Sin[t2] + \sqrt{Sin[t2]^2})\}]\}
```

Intersect the first and third circle to find the intersection points.

```
\begin{split} & \text{In[4]: } \texttt{pts13} = \texttt{Points2D[C1, C3];} \\ & \text{pts13} = \texttt{Map[FullSimplify, Map[(ImPoint2D @@ #)&, pts13]];} \\ & \text{pts13} = \texttt{Map[(Point2D @@ #)&, pts13]} \\ & \text{Out[4]} \quad \left\{ \texttt{Point2D}\left[\left\{\frac{1}{2} \operatorname{Cot}\left[\frac{\texttt{t3}}{2}\right] \left(\operatorname{Sin[t3]} - \sqrt{\operatorname{Sin[t3]}^2}\right), \frac{1}{2} \left(\operatorname{Sin[t3]} - \sqrt{\operatorname{Sin[t3]}^2}\right)\right\}\right], \\ & \text{Point2D}\left[\left\{\frac{1}{2} \operatorname{Cot}\left[\frac{\texttt{t3}}{2}\right] \left(\operatorname{Sin[t3]} + \sqrt{\operatorname{Sin[t3]}^2}\right), \frac{1}{2} \left(\operatorname{Sin[t3]} + \sqrt{\operatorname{Sin[t3]}^2}\right)\right\}\right] \right\} \end{split}
```

Intersect the second and third circle to find the intersection points.

```
 \begin{split} & \ln[5]: \, \texttt{pts23} = \texttt{Points2D[C2, C3];} \\ & \texttt{pts23} = \texttt{Map[FullSimplify, Map[(ImPoint2D @@ #)&, \texttt{pts23}];} \\ & \texttt{pts23} = \texttt{Map[(Point2D @@ #)&, \texttt{pts23}]} \\ & \textit{Out[5]} \quad \left\{ \texttt{Point2D}\left[ \left\{ \frac{1}{2} \cos\left[ \frac{\texttt{t2} + \texttt{t3}}{2} \right] \csc\left[ \frac{\texttt{t2} - \texttt{t3}}{2} \right] \left( \sin[\texttt{t2} - \texttt{t3}] + \sqrt{\sin[\texttt{t2} - \texttt{t3}]^2} \right), \right. \\ & \left. \frac{1}{2} \csc\left[ \frac{\texttt{t2} - \texttt{t3}}{2} \right] \left( \sin[\texttt{t2} - \texttt{t3}] + \sqrt{\sin[\texttt{t2} - \texttt{t3}]^2} \right) \sin\left[ \frac{\texttt{t2} + \texttt{t3}}{2} \right] \right\} \right], \\ & \texttt{Point2D}\left[ \left\{ -\frac{1}{2} \cos\left[ \frac{\texttt{t2} + \texttt{t3}}{2} \right] \csc\left[ \frac{\texttt{t2} - \texttt{t3}}{2} \right] \left( -\sin[\texttt{t2} - \texttt{t3}] + \sqrt{\sin[\texttt{t2} - \texttt{t3}]^2} \right), \\ & \left. -\frac{1}{2} \csc\left[ \frac{\texttt{t2} - \texttt{t3}}{2} \right] \left( -\sin[\texttt{t2} - \texttt{t3}] + \sqrt{\sin[\texttt{t2} - \texttt{t3}]^2} \right) \sin\left[ \frac{\texttt{t2} + \texttt{t3}}{2} \right] \right\} \right] \right\} \end{split}
```

One of the intersection points must be the origin. Which one depends on whether the expression under the radical is positive or negative. We introduce the sign variables  $s_1$ ,  $s_2$  and  $s_3$  which may only take values of  $\pm 1$  to cover all the cases.

```
In[6]: Clear[s2, s3, s23];
pts12 = pts12 //. Sqrt[Sin[t2]^2] -> s2*Sin[t2];
pts13 = pts13 //. Sqrt[Sin[t3]^2] -> s3*Sin[t3];
pts23 = pts23 //. Sqrt[Sin[t2-t3]^2] -> s23*Sin[t2-t3];
```

Each pair of intersection points must include the origin as one point. Notice that the other point has the same coordinates no matter which sign is used.

 $In[7]: pts12 = Map[(pts12 /. s2 -> #)&, \{-1, 1\}]$   $Out[7] \left\{ \left\{ Point2D[\left\{ Cot[\frac{t2}{2}\right] Sin[t2], Sin[t2] \right\} \right\}, Point2D[\left\{ 0, 0 \right\} ], \\ \left\{ Point2D[\left\{ 0, 0 \right\} \right], Point2D[\left\{ Cot[\frac{t2}{2}\right] Sin[t2], Sin[t2] \right\} \right\} \right\}$ 

```
\begin{split} &In[8]: \mathbf{pts13} = \mathbf{Map[(pts13 /. s3 -> \#)\&, \{-1, 1\}]} \\ &Out[8] \left\{ \left\{ \text{Point2D}[\left\{ \text{Cot}[\frac{t3}{2}\right] \text{Sin[t3]}, \text{Sin[t3]} \right\} \right], \text{Point2D}[\left\{ 0, 0 \right\} \right], \\ &\left\{ \text{Point2D}[\left\{ 0, 0 \right\} \right], \text{Point2D}[\left\{ \text{Cot}[\frac{t3}{2}\right] \text{Sin[t3]}, \text{Sin[t3]} \right\} \right] \right\} \\ &In[9]: \mathbf{pts23} = \mathbf{Map[(pts23 /. s23 -> \#)\&, \{-1, 1\}]} \\ &Out[9] \left\{ \left\{ \text{Point2D}[\left\{ 0, 0 \right\} \right], \text{Point2D}[\left\{ \text{Cos}[\frac{t2 + t3}{2}\right] \text{Csc}[\frac{t2 - t3}{2}\right] \text{Sin[t2 - t3]}, \\ & \quad \text{Csc}[\frac{t2 - t3}{2}\right] \text{Sin[t2 - t3]} \text{Sin}[\frac{t2 + t3}{2}\right] \right\} \right\}, \\ &\left\{ \text{Point2D}[\left\{ \text{Cos}[\frac{t2 + t3}{2}\right] \text{Csc}[\frac{t2 - t3}{2}\right] \text{Sin[t2 - t3]}, \\ & \quad \text{Csc}[\frac{t2 - t3}{2}\right] \text{Sin[t2 - t3]} \text{Sin}[\frac{t2 + t3}{2}\right] \right\} \right], \\ & \quad \text{Point2D}[\left\{ 0, 0 \right\} \right\} \end{split}
```

Use one of the non-origin points from each of the intersection lists.

```
In[10]: p12 = Union[Flatten[pts12]][[2]];
p13 = Union[Flatten[pts13]][[2]];
p23 = Union[Flatten[pts23]][[2]];
```

Construct a circle through the three points and examine its radius. Since its radius is one, the circle through the three points is congruent to the other three.

In[11]: Radius2D[Cl23 = Circle2D[pl2, pl3, p23]] // FullSimplify
Out[11] 1

#### Discussion

This is a plot of a numerical example.



# knotin.nb

#### Incenter on Knot Circle

#### **Exploration**

Show that the incenter of a triangle (the center point of the circle inscribed in the triangle) is on one of the knot circles for the biarc configuration defined by the triangle.

#### Approach

Construct a triangle in a simplified position. Construct the incenter. Construct the knot circles. Show that the incenter is on one of the knot circles.

#### Solution

Define a function to compute the knot circles.

```
In[1]: KnotCircles2D[
    t1: Triangle2D[p1: {x1_, y1_},
        pA: {xA_, yA_},
        p2: {x2_, y2_}]] :=
    Module[{pt1, pt2},
        pt1 = Point2D[t1, Inscribed2D];
        pt2 = Point2D[Point2D[pA],
            Point2D[p1],
            -Distance2D[p2, pA]];
    Map[Circle2D[Point2D[p1],
            Point2D[p2], #]&,
        {pt1, pt2}]];
```

Create the triangle.

```
In[2]: Clear[a, b];
t1 = Triangle2D[{0, 0}, {a, b}, {1, 0}];
```

Construct the incenter of the triangle

```
In[3]: pt1 = Point2D[t1, Inscribed2D] // Simplify
```

 $\textit{Out[3] Point2D}\Big[\Big\{\frac{a+\sqrt{a^2+b^2}}{1+\sqrt{(-1+a)^2+b^2}+\sqrt{a^2+b^2}}\,,\,\,\frac{b}{1+\sqrt{(-1+a)^2+b^2}+\sqrt{a^2+b^2}}\,\Big\}\Big]$ 

Construct knot circles for the triangle. This result was computed using *Mathematica* Version 3.0.1. Version 4.0 computes a different expression for the circle's radius that is algebraically equivalent. The result shown in the final step is not affected by this difference.

```
In[4]: kc1 = KnotCircles2D[t1][[1]] // FullSimplify
```

```
Out[4] \ \ Circle2D\Big[\Big\{\frac{1}{2}, \ \frac{(-1+a)\ a+b^2-\sqrt{(-1+a)\ ^2+b^2}\ \sqrt{a^2+b^2}}{2\ b}\Big\}, \ \sqrt{\frac{1}{2+\frac{2\ ((-1+a)\ a+b^2)}{\sqrt{(-1+a)\ ^2+b^2}\ \sqrt{a^2+b^2}}}\Big]
```

Show that the incenter is on the circle.

```
In[5]: eq1 = Polynomial2D[
    Quadratic2D[kc1],
    Coordinates2D[pt1]] // Simplify
Out[5] 0
```

# Indet.nb

#### Line General Equation Determinant

#### **Exploration**

Show that the general equation of a line Ax + By + C = 0 is coincident with the line

$$\begin{vmatrix} x & y & 1 \\ -AC & -BC & A^2 + B^2 \\ B & -A & 0 \end{vmatrix} = 0$$

given in determinant form.

### Approach

Evaluate the determinant and show that the result is equivalent to the equation of the specified line.

### Solution

Use the **Det** command to form the determinant.

The line represented by the determinant is coincident to the given line.

```
In[2]: Line2D[d, {x, y}] // Simplify
Out[2] Line2D[A1, B1, C1]
```

#### Discussion

If the constant coefficient of the line is non-zero  $(C \neq 0)$  then a simpler determinant represents the line and is given by

```
\begin{vmatrix} x & y & 1 \\ -C & 0 & A \\ 0 & -C & B \end{vmatrix} = 0.
In[3]: d = Det[{{x, y, 1}, {-C1, 0, A1}, {0, -C1, B1}]
out[3] C1<sup>2</sup> + Al C1 x + Bl C1 y
In[4]: Line2D[d, {x, y}] // Simplify
out[4] Line2D[A1, B1, C1]
```

# Indist.nb

### Vertical/Horizontal Distance to a Line

#### **Exploration**

Show that the *vertical* distance,  $d_v$ , from a point  $(x_1, y_1)$  to a line whose equation is

$$Ax + By + C = 0$$

is given by

$$d_v = \left| \frac{(Ax_1 + By_1 + C)}{B} \right|$$

and the *horizontal* distance,  $d_h$ , is given by

$$d_h = \left| \frac{(Ax_1 + By_1 + C)}{A} \right|.$$

#### Approach

Construct a vertical (horizontal) line through the given point. Intersect the vertical (horizontal) line with the given line. The required distance,  $d_v$  ( $d_h$ ), is the distance between  $P_1$  and the intersection point.

#### Solution

Construct the vertical (horizontal) line through the given point.

```
In[1]: Clear[x1, y1];
    p1 = Point2D[x1, y1];
    {lv = Line2D[p1, Infinity], lh = Line2D[p1, 0]}
Out[1] {Line2D[1, 0, -x1], Line2D[0, -1, y1]}
```

Intersect the vertical (horizontal) line with the given line.

Find the distance between the intersection point and  $P_1$ . The expressions given by Mathematica are equivalent to the desired results.

```
In[3]: \{ \text{Distance2D[p1, pv]}, \text{Distance2D[p1, ph]} \}Out[3] \left\{ \sqrt{\left(\frac{c+a x 1}{b} + y 1\right)^2}, \sqrt{\left(x 1 + \frac{c+b y 1}{a}\right)^2} \right\}
```

#### Discussion

If the point is on the line, then both distances are clearly zero since the point satisfies the equation of the line. If the line has a slope of  $\pm 1$   $(A = \pm B)$ , then  $d_v = d_h$ . If the given line is vertical (horizontal), then the vertical (horizontal) distance formula is invalid (i.e. A or B is zero). Here's a function for vertical distance. The function for horizontal distance would be similar.

```
In[4]: Distance2D[Point2D[{x1_, y1_}],
        Line2D[A2_, B2_, C2_],
        Vertical2D] :=
        Abs[(A2 * x1 + B2 * y1 + C2) / B2] /;
        Not[IsZero2D[B2]]
```

This computes the vertical distance from (9, 2) to the line 2x - 4y - 3 = 0.

## InIndist.nb

#### Line Segment Cut by Two Lines

#### **Exploration**

Let  $L_1$  and  $L_2$  be two intersecting lines and  $P_0$  a point. Describe a procedure for finding the lines through  $P_0$  such that  $L_1$  and  $L_2$  cut off a line segment of length S > 0. Implement the solution as a *Mathematica* function.

#### Approach

Translate  $L_1$  and  $L_2$  so that their intersection point is at the origin.  $L_1$  and  $L_2$  can then be written as  $A_1x + B_1y = 0$  and  $A_2x + B_2y = 0$ . The line  $L_{12}$  through  $P_0$  can then be written as  $A_{12}x + B_{12}y + 1 = 0$ , since  $L_{12}$  cannot pass through the origin. Since  $P_0$  is on  $L_{12}$ ,  $A_{12}x_0 + B_{12}y_0 + 1 = 0$ . A second equation can be formed using the condition that the distance between the points of intersection of  $(L_1 \text{ and } L_{12})$  and  $(L_2 \text{ and } L_{12})$  must be S. Solve the two equations for  $A_{12}$  and  $B_{12}$ . There are two or four solutions depending on the geometric configuration and the value of S. Translate the resulting solutions back to the original position.

#### Solution

Special case first, the lines intersect at the origin. The equations are solved using NSolve to avoid complicated exact solutions.

Here's the general case. It uses the special case for the core computation.

#### Discussion

Here's an example of the special case that has two solutions:  $L_1 \equiv 2x - 3y = 0$  and  $L_2 \equiv 4x + 3y = 0$  with S = 2 through the point (2, -1).

```
In[3]: P0 = Point2D[2, -1];
L1 = Line2D[2, -3, 0];
L2 = Line2D[4, 3, 0];
L12 = Line2D[P0, L1, L2, 2]
Out[3] {Line2D[-1.32353, -1.64706, 1], Line2D[0.0466406, 1.09328, 1]}
```

```
In[4]: Sketch2D[{P0, L1, L2, L12}];
```



Here's an example of the general case that has four solutions:  $L_1 \equiv 2x + y - 2 = 0$  and  $L_2 \equiv -x + 3y - 1 = 0$  with S = 4 through the point (-1, 2).

```
In[5]: P0 = Point2D[1, 2];
L1 = Line2D[2, 1, -2];
L2 = Line2D[-1, 3, -1];
L12 = Line2D[P0, L1, L2, 4]
```

Out[5] {Line2D[-1.39545, -0.42091, 2.23727], Line2D[-0.54985, -0.59003, 1.72991], Line2D[-0.086206, -0.682759, 1.45172], Line2D[0.531505, -0.806301, 1.0811]}

In[6]: Sketch2D[{P0, L1, L2, L12}];



## Inquad.nb

#### Line Normal to a Quadratic

#### **Exploration**

Show that the normal line passing through the point  $(x_1, y_1)$  on the quadratic whose equation is  $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$  is given by

$$k_1x - k_2y - k_1x_1 + k_2y_1 = 0$$

where

$$k_1 = Bx_1 + 2Cy_1 + E$$
 and  $k_2 = 2Ax_1 + By_1 + D$ .

#### Approach

Construct the polar line of the quadratic with respect to the quadratic. Construct the line normal to the polar through the point. This is the desired normal line.

#### Solution

Construct the polar line (which is tangent to the quadratic if the point is on the quadratic).

```
In[1]: Clear[x1, y1, a, b, c, d, e, f];
    p1 = Point2D[x1, y1];
    q1 = Quadratic2D[a, b, c, d, e, f];
    l1 = Line2D[p1, q1]
Out[1] Line2D[d+2axl+by1, e+bx1+2cy1, 2f+dxl+ey1]
```

Construct the normal line.

```
In[2]: l2 = Line2D[p1, l1] // Simplify
Out[2] Line2D[e+bxl+2cyl, -d-2axl-byl, yl (d+2axl+byl) - xl (e+bxl+2cyl)]
```

#### Discussion

Define a function for constructing the normal line.

```
In[3]: Line2D[
    p1:Point2D[{x1_, y1_}],
    q1:Quadratic2D[a_, b_, c_, d_, e_, f_],
    Normal2D] :=
    Simplify[Line2D[p1, Line2D[p1, q1]]];
```

This is the plot of a numerical example.

```
In[4]: ql = Quadratic2D[el = Ellipse2D[{0, 0}, 2, 1, 0]];
pl = Point2D[el[Pi/9]];
ll = Line2D[pl, ql, Normal2D]
Out[4] Line2D[2 Sin[<math>\frac{\pi}{9}], -Cos[\frac{\pi}{9}], -3 Cos[\frac{\pi}{9}] Sin[\frac{\pi}{9}]]
```

In[5]: Sketch2D[{el, pl, ll}, CurveLength2D -> 7];



# Insdst.nb

#### **Distance Between Parallel Lines**

#### **Exploration**

Demonstrate that the distance, d, between two parallel lines

 $Ax + By + C_1 = 0$  and  $Ax + By + C_2 = 0$ 

is given by

$$d = \sqrt{\frac{(C_2 - C_1)^2}{A^2 + B^2}}.$$

#### Approach

Create two lines perpendicular to both given lines and passing through the origin. Find the points of intersection between the original lines and the perpendicular lines. Compute the distance between the intersection points, with is the distance between the parallel lines.

#### Solution

Create the two given lines.

Construct two lines perpendicular to the given lines.

```
In[2]: L1 = Line2D[Point2D[0, 0], 11];
L2 = Line2D[Point2D[0, 0], 12];
```

Intersect the lines in pairs to find the intersection points.

Find the distance between the intersection points.

```
In [4]: Distance2D[p1, p2] // Simplify
Out [4] \sqrt{\frac{(C1 - C2)^2}{A^2 + B^2}}
```

#### Discussion

Define a new function to compute the distance between two parallel lines. A more general function could be developed that allows parallel lines whose linear coefficients are multiples of each other.

In[5]: Distance2D[Line2D[A\_, B\_, C1\_], Line2D[A\_, B\_, C2\_]] := Sqrt[(C1 - C2) ^2 / (A^2 + B^2)];

Find the distance between the two lines 2x + 3y + 4 = 0 and 2x + 3y - 3 = 0 using the new function.

## Insegint.nb

#### Intersection Parameters of Two Line Segments

#### **Exploration**

Show that the parameter values,  $t_1$  and  $t_2$ , of the intersection point of two line segments in terms of the end point coordinates is given by

$$t_1 = (x_1(y_3 - y_4) - x_3(y_1 - y_4) + x_4(y_1 - y_2)) / D$$
$$t_2 = (-x_1(y_2 - y_3) + x_2(y_1 - y_3) - x_3(y_1 - y_2)) / D$$

where

$$D = (x_1 - x_2) (y_3 - y_4) (x_3 - x_4) (y_1 - y_2).$$

What is the significance of the values of  $t_1$  and  $t_2$  with respect to the standard parameter range for a line segment?

#### Approach

Create the two line segments and express points on each parametrically. Set the x- and y-coordinates equal to each other and solve for  $t_1$  and  $t_2$ .

#### Solution

Create the two line segments.

```
In[1]: Clear[x1, y1, x2, y2, x3, y3, x4, y4];
L1 = Segment2D[{x1, y1}, {x2, y2}];
L2 = Segment2D[{x3, y3}, {x4, y4}];
```

Find the point coordinates in terms of parameters.

```
In[2]: Clear[t1, t2];
    {pt1 = Point2D[L1[t1]], pt2 = Point2D[L2[t2]]}
Out[2] {Point2D[{x1 + t1 (-x1 + x2), y1 + t1 (-y1 + y2)}],
        Point2D[{x3 + t2 (-x3 + x4), y3 + t2 (-y3 + y4)}]}
```

Equate the abscissas and ordinates and solve for the parameters.

```
\begin{split} \text{In[3]: ans = Solve[{XCoordinate2D[pt1] == XCoordinate2D[pt2],} \\ & \text{YCoordinate2D[pt1] == YCoordinate2D[pt2]}, \\ & \{\texttt{t1, t2}\} \ // \ \text{FullSimplify} \end{split} \\ \text{Out[3] } \left\{ \left\{ \texttt{t1} \rightarrow \frac{\texttt{x4} \ (\texttt{y1} - \texttt{y3}) + \texttt{x1} \ (\texttt{y3} - \texttt{y4}) + \texttt{x3} \ (-\texttt{y1} + \texttt{y4}) \\ - (\texttt{x3} - \texttt{x4}) \ (\texttt{y1} - \texttt{y2}) + (\texttt{x1} - \texttt{x2}) \ (\texttt{y3} - \texttt{y4}) \\ & \texttt{t2} \rightarrow \frac{\texttt{x3} \ (\texttt{y1} - \texttt{y2}) + \texttt{x1} \ (\texttt{y2} - \texttt{y3}) + \texttt{x2} \ (-\texttt{y1} + \texttt{y3}) \\ & \texttt{x3} \ (\texttt{y1} - \texttt{y2}) - (\texttt{x1} - \texttt{x2}) \ (\texttt{y3} - \texttt{y4}) \\ \end{array} \right\} \Big\} \end{split}
```

#### Discussion

The significance of the values of  $t_1$  and  $t_2$  lies in the range of values which determine if the two line segments actually intersect. If  $0 \le t_1 \le 1$  at the intersection point, then the intersection point is on the first line segment; if  $0 \le t_2 \le 1$  at the intersection point, then the intersection point is on the second line segment.

# Insegpt.nb

#### Intersection Point of Two Line Segments

#### **Exploration**

Show that the intersection point of the lines underlying two line segments  $P_1P_2$  and  $P_3P_4$  in terms of the coordinates of the four points is given by

$$x = \frac{(x_2 - x_1)(x_3y_4 - x_4y_3) - (x_4 - x_3)(x_1y_2 - x_2y_1)}{(x_4 - x_3)(y_1 - y_2) - (x_2 - x_1)(y_3 - y_4)}$$
$$y = \frac{(y_3 - y_4)(x_1y_2 - x_2y_1) - (y_1 - y_2)(x_3y_4 - x_4y_3)}{(x_4 - x_3)(y_1 - y_2) - (x_2 - x_1)(y_3 - y_4)}.$$

#### Approach

Construct the two lines underlying the line segments and intersect the lines.

#### Solution

Define the lines underlying the two line segments.

```
In[1]: Clear[x1, y1, x2, y2, x3, y3, x4, y4];
L1 = Line2D[p1 = {x1, y1}, p2 = {x2, y2}];
L2 = Line2D[p3 = {x3, y3}, p4 = {x4, y4}];
```

Compute the intersection point.

```
\begin{split} & \text{In[2]: pt = Point2D[L1, L2]} \\ & \text{Out[2] Point2D}\Big[ \Big\{ \frac{-(-x3+x4) \ (-x2\,y1+x1\,y2) + (-x1+x2) \ (-x4\,y3+x3\,y4)}{(-x3+x4) \ (y1-y2) \ (-x1+x2) \ (y3-y4)} \,, \\ & \frac{(-x2\,y1+x1\,y2) \ (y3-y4) - (y1-y2) \ (-x4\,y3+x3\,y4)}{(-x3+x4) \ (y1-y2) \ (-x1+x2) \ (y3-y4)} \Big\} \Big] \end{split}
```

#### Discussion

Notice that the denominators of the abscissa and ordinate are equal, and that these denominators cannot be zero unless the line segments are parallel, in which case the underlying lines do not intersect. The following is a plot of a numerical example.



## Insperp.nb

#### **Equations of Perpendicular Lines**

#### **Exploration**

Show that the pair of lines ax + by + c = 0 and bx - ay + c' = 0 are perpendicular. Show that the pair

$$ax + by + c = 0$$
 and  $\frac{x}{a} - \frac{y}{b} + c' = 0$ 

is also perpendicular.

#### Approach

The two lines  $A_1x + B_1y + C_1 = 0$  and  $A_2x + B_2y + C_2 = 0$  are perpendicular if the equation  $A_1A_2 + B_1B_2 = 0$  is true. The two pairs of lines given can be shown to be perpendicular by examining this equation.

#### Solution

Formulate the perpendicular condition for the first pair of lines.

```
In[1]: Clear[A1, B1, A2, B2, a, b];
A1*A2+B1*B2 /. {
A1->a, B1->b, A2->b, B2->-a}
Out[1] 0
```

Formulate the perpendicular condition for the second pair of lines.

```
In[2]: A1*A2+B1*B2 /. {
    A1->a, B1->b, A2->1/a, B2->-1/b}
Out[2] 0
```

#### Discussion

Notice that the second pair of lines can be derived from the first by dividing the first equation by the quantity ab. However, this is invalid if either a or b is zero. The relationship shown for the first pair is valid for all lines. The *Descarta2D* function IsPerpendicular2D also verifies that the pairs are perpendicular.

```
In[3]: Clear[c1];
    {IsPerpendicular2D[Line2D[a, b, c], Line2D[b, -a, c1]],
        IsPerpendicular2D[Line2D[a, b, c], Line2D[1/a, -1/b, c1]]}
Out[3] {True, True}
```

694

# Intancir.nb

#### Line Tangent to a Circle

#### **Exploration**

Show that the line  $y = m(x - a) + a\sqrt{1 + m^2}$  is tangent to the circle  $x^2 + y^2 = 2ax$  for all values of m.

#### Approach

Show that the pole point (which is the point of tangency if the line is tangent to the circle) is on the circle.

#### Solution

Construct the line.

Construct the circle.

```
In[2]: c1 = Circle2D[q1 = Quadratic2D[x^2 + y^2 == 2a * x, {x, y}]]
Out[2] Circle2D[{a, 0}, \sqrt{a^2}]
```

Construct the pole point.

```
In[3]: p1 = Point2D[11, c1] // SimplifyOut[3] Point2D[\left\{a - \frac{am}{\sqrt{1+m^2}}, \frac{a}{\sqrt{1+m^2}}\right\}]
```

The coordinates of the pole point satisfy the equation of the circle.

In[4]: Polynomial2D[q1, Coordinates2D[p1]] // Simplify
Out[4] 0

### Discussion

This is a plot of a numerical example.

```
In[5]: Sketch2D[{cl /. a -> 1,
    Map[({ll, pl} /. {a -> 1, m -> #})&,
    {0, .5, 1, 2, 5, -5, -2, -1, -.5}]},
    CurveLength2D -> 4];
```



## Intancon.nb

### Line Tangent to a Conic

#### **Exploration**

Use Mathematica to show that the relationship between the coefficients of a line

$$px + qy + r = 0$$

tangent to the conic

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

is given by

$$(4CF - E^2) p^2 + (4AF - D^2) q^2 + (4AC - B^2) r^2 + (BD - 2AE) qr + 2 (BE - 2CD) pr + 2 (DE - 2BF) pq = 0.$$

#### Approach

Intersect the line and the conic and force the intersection points to be coincident by setting the appropriate terms in the resulting expression to zero.

#### Solution

Solve the equation of the line for y.

$$\label{eq:constraint} \begin{array}{l} \mbox{In[1]: Clear[p,q,r,x,y];} \\ \mbox{ansl = Solve[p * x + q * y + r == 0, y]} \end{array}$$
   
 
$$Out[1] \; \left\{ \left\{ y \rightarrow - \frac{r + p \, x}{q} \right\} \right\}$$

Substitute y into the equation of the conic.

```
In[2]: Clear[A1, B1, C1, D1, E1, F1];
eq1 = A1 * x^{2} + B1 * x * y + C1 * y^{2} + D1 * x + E1 * y + F1 /. First[ans1]
Out[2] F1 + D1 x + A1 x^{2} - \frac{E1 (r + p x)}{q} - \frac{B1 x (r + p x)}{q} + \frac{C1 (r + p x)^{2}}{q^{2}}
```

Solve the equation for x.

```
 \begin{split} &In [3]: \mbox{ ans2 = Solve[eql == 0, x] } \\ &Out [3] \ \left\{ \left\{ x \rightarrow \left( \text{El } p \ q - \text{Dl } q^2 - 2 \ \text{Cl } p \ r + \text{Bl } q \ r - \sqrt{\left( \left( -\text{El } p \ q + \text{Dl } q^2 + 2 \ \text{Cl } p \ r - \text{Bl } q \ r \right)^2 - 4 \ \left( \text{Cl } p^2 - \text{Bl } p \ q + \text{Al } q^2 \right) \ \left( \text{Fl } q^2 - \text{El } q \ r + \text{Cl } r^2 \right) \right) \right) / \\ & \left\{ 2 \ \left( \text{Cl } p^2 - \text{Bl } p \ q + \text{Al } q^2 \right) \ \right\}, \\ & \left\{ x \rightarrow \left( \text{El } p \ q - \text{Dl } q^2 - 2 \ \text{Cl } p \ r + \text{Bl } q \ r + \sqrt{\left( \left( -\text{El } p \ q + \text{Dl } q^2 + 2 \ \text{Cl } p \ r - \text{Bl } q \ r \right)^2 - 4 \ \left( \text{Cl } p^2 - \text{Dl } q^2 - 2 \ \text{Cl } p \ r + \text{Bl } q \ r + \sqrt{\left( \left( -\text{El } p \ q + \text{Dl } q^2 + 2 \ \text{Cl } p \ r - \text{Bl } q \ r \right)^2 - 4 \ \left( \text{Cl } p^2 - \text{Bl } p \ q + \text{Al } q^2 \right) \ \left( \text{Fl } q^2 - \text{El } q \ r + \text{Cl } r^2 \right) \right) \right) / \\ & \left\{ 2 \ \left( \text{Cl } p^2 - \text{Bl } p \ q + \text{Al } q^2 \right) \ \left( \text{Fl } q^2 - \text{El } q \ r + \text{Cl } r^2 \right) \right) \right\} \end{split}
```

The line is tangent if the expression under the radical is zero (i.e. the points of tangency are coincident).

```
In[4]: expr1 = ans2[[1, 1, 2, 3, 5, 2, 1]]
Out [4] (-Elpq+Dlq<sup>2</sup>+2Clpr-Blqr)<sup>2</sup>-4 (Clp<sup>2</sup>-Blpq+Alq<sup>2</sup>) (Flq<sup>2</sup>-Elqr+Clr<sup>2</sup>)
```

Put the expression into the desired form.

```
In[5]: expr2 = expr1 // Expand // Factor
Out[5] q<sup>2</sup> (E1<sup>2</sup> p<sup>2</sup> - 4 C1 F1 p<sup>2</sup> - 2 D1 E1 p q + 4 B1 F1 p q + D1<sup>2</sup> q<sup>2</sup> - 4 A1 F1 q<sup>2</sup> + 4 C1 D1 p r -
2 B1 E1 p r - 2 B1 D1 q r + 4 A1 E1 q r + B1<sup>2</sup> r<sup>2</sup> - 4 A1 C1 r<sup>2</sup>)
```

Divide both sides by  $q^2$ .

```
In[6]: expr3 = expr2[[2]]
Out[6] E1<sup>2</sup> p<sup>2</sup> - 4 C1 F1 p<sup>2</sup> - 2 D1 E1 p q + 4 B1 F1 p q + D1<sup>2</sup> q<sup>2</sup> - 4 A1 F1 q<sup>2</sup> + 4 C1 D1 p r -
2 B1 E1 p r - 2 B1 D1 q r + 4 A1 E1 q r + B1<sup>2</sup> r<sup>2</sup> - 4 A1 C1 r<sup>2</sup>
```

Pick out the coefficients of each of the desired terms (multiplied by -1 to match the desired sign).

# mdcircir.nb

### Medial Curve, Circle–Circle

### Exploration

Show that the two quadratics whose equations are given by

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

where

$$A = 4 ((h_1 - h_2)^2 - R),$$
  

$$B = 8 (h_1 - h_2) (k_1 - k_2),$$
  

$$C = 4 ((k_1 - k_2)^2 - R),$$
  

$$D = 4 (h_1(-D_1 + D_2 + R) + h_2(D_1 - D_2 + R)),$$
  

$$E = 4(k_1(-D_1 + D_2 + R) + k_2(D_1 - D_2 + R)) \text{ and }$$
  

$$F = (D_1 - D_2)^2 - 2 (D_1 + D_2) R + R^2$$

and

$$R = (r_1 - sr_2)^2,$$
  

$$D_1 = h_1^2 + k_1^2,$$
  

$$D_2 = h_2^2 + k_2^2 \text{ and }$$
  

$$s = \pm 1$$

are equidistant from the two circles

$$(x - h_1)^2 + (y - k_1)^2 = r_1^2$$
 and  $(x - h_2)^2 + (y - k_2)^2 = r_2^2$ .

#### Approach

Create the two circles and form an equation by equating the distance to each circle from a generic point.

#### Solution

Create the geometry.

```
In[1]: Clear[x, y, h1, k1, r1, h2, k2, r2];
    P = Point2D[x, y];
    C1 = Circle2D[{h1, k1}, r1];
    C2 = Circle2D[{h2, k2}, r2];
```

Find the distances to the two circles, where  $s_1 = \pm 1$  and  $s_2 = \pm 1$ .

```
In[2]: Clear[s1];
d1 = s1 * (Distance2D[P, Point2D[C1]] - r1)
Out[2] s1 (-r1 + \sqrt{(-h1 + x)^{2} + (-k1 + y)^{2}})
In[3]: Clear[s2];
d2 = s2 * (Distance2D[P, Point2D[C2]] - r2)
Out[3] s2 (-r2 + \sqrt{(-h2 + x)^{2} + (-k2 + y)^{2}})
```

Equate the two distances and simplify by making substitutions.

```
In [4]: Clear[E1]; \\eq1 = d1 - d2 /. Sqrt[E1_] :> Sqrt[Expand[E1]]
Out [4] s1 (-r1 + \sqrt{h1^2 + k1^2 - 2h1 x + x^2 - 2k1 y + y^2}) - s2 (-r2 + \sqrt{h2^2 + k2^2 - 2h2 x + x^2 - 2k2 y + y^2})
In [5]: Clear[D1, D2]; \\eq2 = eq1 /. {h1^2 + k1^2 - >D1, h2^2 + k2^2 - >D2}
Out [5] s1 (-r1 + \sqrt{D1 - 2h1 x + x^2 - 2k1 y + y^2}) - s2 (-r2 + \sqrt{D2 - 2h2 x + x^2 - 2k2 y + y^2})
```

Rearrange the equation and square both sides (twice).

```
In [6]: \{lhs = eq2[[1]] // Expand, rhs = eq2[[2]] // Expand\}
Out [6] \{-rl sl + sl \sqrt{Dl - 2hl x + x^{2} - 2kl y + y^{2}}, r2 s2 - s2 \sqrt{D2 - 2h2 x + x^{2} - 2k2 y + y^{2}} \}
In [7]: Clear [s, R];
eq3 = ((lhs [[1]] + rhs [[1]])^{2} - (lhs [[2]] + rhs [[2]])^{2} // Expand) //.
\{sl^{2} -> l, s2^{2} -> l, sl + s2 -> s, rl^{2} - 2 + s + rl + r2 + r2^{2} - 2 + R \}
Out [7] -Dl - D2 + R + 2hl x + 2h2 x - 2x^{2} + 2kl y + 2k2 y - 2y^{2} + 2s \sqrt{D1 - 2hl x + x^{2} - 2kl y + y^{2}} \sqrt{D2 - 2h2 x + x^{2} - 2k2 y + y^{2}}
```

```
In[8]: eq4 = Drop[eq3, -1]^{2} - Last[eq3]^{2}
Out[8] (-D1 - D2 + R + 2 h1 x + 2 h2 x - 2 x<sup>2</sup> + 2 k1 y + 2 k2 y - 2 y<sup>2</sup>)<sup>2</sup> - 4 s<sup>2</sup> (D1 - 2 h1 x + x<sup>2</sup> - 2 k1 y + y<sup>2</sup>) (D2 - 2 h2 x + x<sup>2</sup> - 2 k2 y + y<sup>2</sup>)
```

Form a quadratic and simplify.

```
In[9]: Q1 = Map[Factor,
Quadratic2D[eq4 /. s^2 -> 1, {x, y}]] //. {
    s^2 -> 1,
    h1^2 - 2*h1*h2+h2^2 -> (h1 - h2)^2,
    k1^2 - 2*k1*k2+k2^2 -> (k1 - k2)^2,
    D1^2 - 2*b1*b2+b2^2 -> (b1 - b2)^2]
Out[9] Quadratic2D[4 ((h1 - h2)<sup>2</sup> - R), 8 (h1 - h2) (k1 - k2),
    4 ((k1 - k2)<sup>2</sup> - R), -4 (D1 h1 - D2 h1 - D1 h2 + D2 h2 - h1 R - h2 R),
    -4 (D1 k1 - D2 k1 - D1 k2 + D2 k2 - k1 R - k2 R), (D1 - D2)<sup>2</sup> - 2 D1 R - 2 D2 R + R<sup>2</sup>]
```

By inspection, the resulting quadratic is the same as the desired one. This result was computed using *Mathematica* Version 3.0.1. Version 4.0 computes a slightly different result that is algebraically equivalent.
# mdlncir.nb

## Medial Curve, Line-Circle

## Exploration

Show that the two quadratics whose equations are given by

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

where

$$A = B_1^2,$$
  

$$B = -2A_1B_1,$$
  

$$C = A_1^2,$$
  

$$D = -2(h_2 + A_1(C_1 + sr_2)),$$
  

$$E = -2(k_2 + B_1(C_1 + sr_2)),$$
  

$$F = h_2^2 + k_2^2 - r_2^2 - C_1(C_1 + 2sr_2) \text{ and }$$
  

$$s = \pm 1.$$

are equidistant from the line

$$A_1x + B_1y + C_1 = 0$$

and the circle

$$(x-h)^{2} + (y-k)^{2} = r^{2}$$

assuming  $A_1^2 + B_1^2 = 1$ .

#### Approach

Create the line and the circle. Form an equation of the distances from a generic point to the line and circle.

#### Solution

Create the geometry.

```
In[1]: Clear[x, y, A1, B1, C1, h2, k2, r2];
    P = Point2D[x, y];
    l1 = Line2D[A1, B1, C1];
    c2 = Circle2D[{h2, k2}, r2];
```

Find the distance from the point to the line, where  $s_1 = \pm 1$ .

```
In[2]: Clear[s1, E1];
d1 = s1 * Distance2D[P, 11] //.
{A1^2 + B1^2 -> 1, Sqrt[E1_^2] -> E1}
```

Out[2] sl (Cl + Al x + Bl y)

Find the distance from the point to the circle, where  $s_2 = \pm 1$ .

```
In [3]: Clear[s2];
d2 = s2 * (Distance2D[P, Point2D[c2]] - r2) // Expand
Out [3] -r2 s2 + s2 \sqrt{(-h2 + x)^2 + (-k2 + y)^2}
```

Rearrange the equation  $d_1 = d_2$  and square both sides.

 $In[4]: eql = (dl - d2[[1]])^2 == d2[[2]]^2 /. \{sl^2 -> 1, s2^2 -> 1\}$  $Out[4] (r2 s2 + s1 (Cl + Al x + Bl y))^2 == (-h2 + x)^2 + (-k2 + y)^2$ 

Form a quadratic and simplify.

```
In[5]: Q1 = Quadratic2D[eq1, {x, y}] //.
        {s1^2 -> 1,
        s2^2 -> 1,
        A1^2 - 1 -> -B1^2,
        B1^2 - 1 -> -A1^2;
Out[5] Quadratic2D[-B1<sup>2</sup>, 2A1 B1, -A1<sup>2</sup>, 2A1 C1 + 2h2 + 2A1 r2 s1 s2,
        2 B1 C1 + 2k2 + 2B1 r2 s1 s2, C1<sup>2</sup> - h2<sup>2</sup> - k2<sup>2</sup> + r2<sup>2</sup> + 2C1 r2 s1 s2]
```

Put the quadratic into the desired form, and use  $s = s_1 s_2 = \pm 1$ .

# mdlnln.nb

### Medial Curve, Line-Line

#### **Exploration**

Show that the pair of lines whose equations are

$$\frac{A_1x + B_1y + C_1}{\sqrt{A_1^2 + B_1^2}} = \pm \frac{A_2x + B_2y + C_2}{\sqrt{A_2^2 + B_2^2}}$$

is equidistant from the two lines  $A_1x + B_1y + C_1 = 0$  and  $A_2x + B_2y + C_2 = 0$ .

#### Approach

Create both lines. Compute the distances to an arbitrary point. Form an equation by setting the distances equal to each other.

#### Solution

Create the two lines.

```
In[1]: Clear[x, y, A1, B1, C1, A2, B2, C2];
    P = Point2D[x, y];
    l1 = Line2D[A1, B1, C1];
    l2 = Line2D[A2, B2, C2];
```

Compute the distance from the first line. Use  $s_1 = \pm 1$  to eliminate the radical.

Compute the distance from the second line. Use  $s_2 = \pm 1$  to eliminate the radical.

```
In [3]: Clear[s2];

d2 = Distance2D[P, 12] /.

Sqrt[E1_^2/E2_] :> s2 * E1 / Sqrt[E2]

Out [3] \frac{s2 (C2 + A2 x + B2 y)}{\sqrt{A2^2 + B2^2}}
```

Form the equation.

In[4]: eql = dl == d2Out[4]  $\frac{sl (Cl + Al x + Bl y)}{\sqrt{Al^2 + Bl^2}} = \frac{s2 (C2 + A2 x + B2 y)}{\sqrt{A2^2 + B2^2}}$ 

Combine  $s_1$  and  $s_2$  into a single sign constant  $s = \pm 1$ .

In [5]: Clear[s];  
eq1 /. {s1 -> 1, s2 -> s}  
Out [5] 
$$\frac{C1 + A1 x + B1 y}{\sqrt{A1^2 + B1^2}} = \frac{s (C2 + A2 x + B2 y)}{\sqrt{A2^2 + B2^2}}$$

# mdptcir.nb

## Medial Curve, Point-Circle

## Exploration

Show that the quadratic equation

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

where

$$A = 4 \left( (x_1 - h_2)^2 - r_2^2 \right),$$
  

$$B = 8 (x_1 - h_2) (y_1 - k_2),$$
  

$$C = 4 \left( (y_1 - k_2)^2 - r_2^2 \right),$$
  

$$D = 4 \left( R(x_1 - h_2) + 2r_2^2 x_1 \right),$$
  

$$E = 4 \left( R(y_1 - k_2) + 2r_2^2 y_1 \right),$$
  

$$F = R^2 - 4r_2^2 (x_1^2 + y_1^2) \text{ and}$$
  

$$R = \left( h_2^2 + k_2^2 \right) - \left( x_1^2 + y_1^2 \right) - r_2^2$$

is equidistant from the point  $P_1(x_1, y_1)$  and the circle

$$(x - h_2)^2 + (y - k_2)^2 = r_2^2.$$

## Approach

Create the point and the circle. Compute the distances to an arbitrary point. Set the distances equal to form the equation.

#### Solution

Create the point and the circle.

```
In[1]: Clear[x, y, x1, y1, h2, k2, r2];
    P = Point2D[x, y];
    p1 = Point2D[x1, y1];
    c2 = Circle2D[{h2, k2}, r2];
```

Compute the distance between the two points.

```
In[2]: dl = Distance2D[P, p1]^2
Out[2] (x - x1)^2 + (y - y1)^2
```

The distance to the circle is either D - r or r - D, where D is the distance from the point to the center of the circle. Squaring removes the ambiguity.

```
In [3]: Clear[a, b]; 
d2 = (Distance2D[P, Point2D[c2]] - r2) ^2 /. 
(a_ + b_) ^2 -> a^2 + 2 a * b + b^2 
Out [3] r2<sup>2</sup> + (-h2 + x)<sup>2</sup> + (-k2 + y)<sup>2</sup> - 2 r2 <math>\sqrt{(-h2 + x)^2 + (-k2 + y)^2}
```

Simplify the equations  $d_1 = d_2$ .

```
In[4]: {ls1 = d1 - d2[[{1, 2, 3}]] // Expand,
      rs1 = d2[[4]]}
Out [4] \left\{-h2^2 - k2^2 - r2^2 + 2h2x - 2xx1 + x1^2 + 2k2y - 2yy1 + y1^2\right\}
       -2 r2 \sqrt{(-h2 + x)^{2} + (-k2 + y)^{2}}
In[5]: Clear[R];
      ls2 = ls1 /.
       \{-h2^{2} - k2^{2} + x1^{2} + y1^{2} - r2^{2} - -2 + r2^{2} - R\}
Out[5] -R-2r2<sup>2</sup> + 2h2x - 2xx1 + 2k2y - 2yy1
In[6]: Q1 = Quadratic2D[ls2^2 == rs1^2, {x, y}]
R^2 - 4 h2^2 r2^2 - 4 k2^2 r2^2 + 4 R r2^2 + 4 r2^4]
In[7]: Clear[E1, E2];
      a = Factor[Q1[[1]]] //. {
        (E1_ - E2_) (E1_ + E2_) -> E1^2 - E2^2,
       (h2 - x1)^{2} \rightarrow (x1 - h2)^{2}
Out[7] 4 (-r2^2 + (-h2 + x1)^2)
```

```
In[8]: b = Factor[Q1[[2]]] /. (h2 - x1) (k2 - y1) \rightarrow (x1 - h2) (y1 - k2)
Out[8] = 8(-h2 + x1)(-k2 + y1)
In[9]: c = Factor[Q1[[3]]] //. {
         (E1_ - E2_) (E1_ + E2_) -> E1^2 - E2^2,
          (k2 - y1)^{2} \rightarrow (y1 - k2)^{2}
Out[9] 4 (-r2^2 + (-k2 + y1)^2)
In[10]: d = Factor[Q1[[4]]] //. {
            h2 * R - x1 * R -> R (h2 - x1),
            -4 (R (h2 - x1) - 2 * r2^{2} * x1) \rightarrow
            4 (R (x1 - h2) + 2 * r2^{2} * x1)}
Out[10] 4 (2r2^2x1 + R(-h2 + x1))
In[11]: e = Factor[Q1[[5]]] //. {
           k2 * R - y1 * R -> R (k2 - y1),
            -4 (R (k2 - y1) - 2 * r2^2 * y1) \rightarrow
            4 (R (y1 - k2) + 2 * r2 * 2 * y1) }
Out[11] 4 (2 r2<sup>2</sup> y1 + R (-k2 + y1))
In[12]: f = Factor[Q1[[6]]] /.
           4 * R * r2^2 - 4 * r2^2 * (h2^2 + k2^2 - (x2^2 + y2^2) - r2^2) // Expand;
          f1 = f[[1]] + Factor[f[[{2, 3}]]]
\textit{Out[12]} \ \texttt{R}^2 - 4 \ \texttt{r2}^2 \ (\texttt{x2}^2 + \texttt{y2}^2)
In[13]: Quadratic2D[a, b, c, d, e, f1]
\textit{Out[13]} \quad \texttt{Quadratic2D[4} \ (-r2^2 + \ (-h2 + x1)^2) \ , \ \texttt{8} \ (-h2 + x1) \ (-k2 + y1) \ , \ \texttt{4} \ (-r2^2 + \ (-k2 + y1)^2) \ ,
```

 $4 \ (2 \ r2^2 \ x1 + \texttt{R} \ (-h2 + x1) \ ) \ \text{,} \ 4 \ (2 \ r2^2 \ y1 + \texttt{R} \ (-k2 + y1) \ ) \ \text{,} \ \texttt{R}^2 - 4 \ r2^2 \ (x2^2 + y2^2) \ ]$ 

# mdptln.nb

### Medial Curve, Point-Line

## Exploration

Show that the quadratic equation

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

where

$$A = B_2^2,$$
  

$$B = -2A_2B_2,$$
  

$$C = A_2^2,$$
  

$$D = -2(x_1 + A_2C_2),$$
  

$$E = -2(y_1 + B_2C_2) \text{ and }$$
  

$$F = x_1^2 + y_1^2 - C_2^2$$

is equidistant from the point  $P_1(x_1, y_1)$  and the line  $L \equiv A_1 x + B_1 y + C_1 = 0$ , assuming that L is normalized  $(A_2^2 + B_2^2 = 1)$ .

## Approach

Create the point and the line. Compute distances to an arbitrary point. Form an equation by setting the distances equal to each other.

### Solution

Create the point and the line.

```
In[1]: Clear[x, y, x1, y1, A2, B2, C2];
    P = Point2D[x, y];
    p1 = Point2D[x1, y1];
    l2 = Line2D[A2, B2, C2];
```

Form an equation by setting the distances (squared) equal to each other.

In [2]: eq1 = Distance2D[P, p1] ^2 == Distance2D[P, 12] ^2 // Simplify Out [2]  $(x - x1)^2 + (y - y1)^2 == \frac{(C2 + A2 x + B2 y)^2}{A2^2 + B2^2}$ 

Form the quadratic and simplify.

```
In[3]: Q1 = Quadratic2D[eq1, {x, y}] //. {
    A2^2 + B2^2 -> 1,
    1 - A2^2 -> B2^2,
    1 - B2^2 -> A2^2;
    Map[Factor, Q1]
```

```
\textit{Out[3]} \quad \texttt{Quadratic2D[B2^2, -2 A2 B2, A2^2, -2 (A2 C2 + x1), -2 (B2 C2 + y1), -C2^2 + x1^2 + y1^2]}
```

## mdptpt.nb

#### Medial Curve, Point–Point

#### **Exploration**

Show that the line  $2(x_2 - x_1)x + 2(y_2 - y_1)y + (x_1^2 + y_1^2) - (x_2^2 + y_2^2) = 0$  is equidistant from the points  $P_1(x_1, y_1)$  and  $P_2(x_2, y_2)$ .

## Approach

Create the points and compute distances to an arbitrary point. Form an equation by setting the distances equal to each other.

## Solution

Create the points.

```
In[1]: Clear[x, y, x1, y1, x2, y2];
        P = Point2D[x, y];
        p1 = Point2D[x1, y1];
        p2 = Point2D[x2, y2];
```

Form an equation by setting the distances (squared) to the arbitrary point equal to each other.

```
In[2]: eq1 = Distance2D[P, p1]^2 ==
Distance2D[P, p2]^2
Out[2] (x - x1)<sup>2</sup> + (y - y1)<sup>2</sup> == (x - x2)<sup>2</sup> + (y - y2)<sup>2</sup>
```

Construct a line from the equation and simplify.

In[3]: Map[Factor, Line2D[eq1, {x, y}] // Simplify] Out[3] Line2D[-2 (x1 - x2), -2 (y1 - y2), x1<sup>2</sup> - x2<sup>2</sup> + y1<sup>2</sup> - y2<sup>2</sup>]

## mdtype.nb

#### Medial Curve Type

#### Exploration

Show that the medial curve equidistant from a point and a circle is a hyperbola when the point is outside the circle, and it is an ellipse when the point is inside the circle. (Hint: Examine the value of the discriminant  $B^2 - 4AC$  of the medial quadratic.)

#### Approach

Create the expression  $B^2 - 4AC$  from the coefficients of the medial quadratic. Consider  $B^2 - 4AC$  with the circle at the origin. Show that the expression is negative when the point is inside the circle and positive when the point is outside the circle.

### Solution

Set the coefficients of the quadratic (from equations listed in the book).

```
In[1]: Clear[x1, y1, h2, k2, r2];
a = 4 * ((x1 - h2) ^2 - r2^2);
b = 8 * (x1 - h2) (y1 - k2);
c = 4 * ((y1 - k2) ^2 - r2^2);
```

Find the discriminant,  $B^2 - 4AC$ , at the origin.

 $In[2]: disc = b^2 - 4 * a * c /. \{h2 \rightarrow 0, k2 \rightarrow 0\} // Simplify$  $Out[2] 64 r2^2 (-r2^2 + x1^2 + y1^2)$ 

Using the distance formula, the point is outside the circle than the discriminant is positive, implying a hyperbola; if the point is inside the circle than the discriminant is negative, implying an ellipse.

## monge.nb

#### Monge's Theorem

## **Exploration**

Given three circles and the external tangent lines of the circles taken in pairs, show that the three intersection points of the three tangent pairs lie on a straight line.

## Approach

Construct the three circles in a simplified position (without loss of generality). Construct the intersection point of the tangent pairs of lines. Show that the points are collinear.

### Solution

Create the three circles.

```
In[1]: Clear[r1, r2, r3, a, b, d];
c1 = Circle2D[{0, 0}, r1];
c2 = Circle2D[{d, 0}, r2];
c3 = Circle2D[{a, b}, r3];
```

Find the intersection point of the external tangents for the first pair.

Find the intersection point of the external tangents for the second pair.

Find the intersection point of the external tangents for the third pair.

The three points are collinear as shown by the zero value of the determinant. The function IsCollinear2D produces the same result.

```
In[5]: MakeRow$2D[Point2D[{x_, y_}]] := {x, y, 1};
    {Det[Map[MakeRow$2D, {p12, p23, p13}]] // Simplify,
    IsCollinear2D[p12, p23, p13]}
Out[5] {0, True}
```

#### Discussion

This is the plot of a numerical example.



## narclen.nb

### Approximate Arc Length of a Curve

#### **Exploration**

The arc length of a smooth, parametrically defined curve can be approximated by a polygon connecting a sequence of points on the curve. Write a *Mathematica* function of the form NArcLength2D[*crv*, { $t_1$ ,  $t_2$ }, n] that approximates the arc length of a curve between two parameter values using a specified number of coordinates at equal parameter intervals between the two given parameters. Produce a graph illustrating the convergence of the approximation to the *Descarta2D* function ArcLength2D[*crv*, { $t_1$ ,  $t_2$ }].

#### Approach

Sum the distances between the points on the curve.

## Solution

Define a function for the arc length by polygonal approximation.

```
In[1]: Off[General::spell1];
NArcLength2D[obj_, {t1_, t2_}, n_] :=
Module[{incr = (t2 - t1) / n},
Sum[
Distance2D[
obj[t1 + i * incr] // N,
obj[t1 + (i + 1) * incr] // N],
{i, 0, n - 1}];
On[General::spell1];
```

Create an object for validating the function.

 $In[2]: cal = ConicArc2D[{0, 0}, {2, 1}, {3, 0}, {3 / 4}];$ 

Create a table of coordinates to plot. The x-coordinate is the number of points used in the approximation and the y-coordinate is the difference between the Descarta2D function and the polygonal approximation.

Plot the results.

In[4]: ListPlot[pts, PlotJoined -> True];



## normal.nb

#### Normals and Minimum Distance

## Exploration

Given a point  $P_0(x_0, y_0)$  and a quadratic Q, find the point(s) P(x, y) on Q such that the line  $PP_0$  is perpendicular to the line tangent to Q at P. Such a line  $PP_0$  is called a *normal* to the quadratic. Use the points P to find the minimum distance from  $P_0$  to Q. Assume that  $P_0$  and Q are defined numerically.

#### Approach

The point P(x, y) is clearly on Q, so P must satisfy the equation for Q. In addition, since the normal line  $PP_0$  is perpendicular to the tangent line, its slope must be the negative reciprocal of the tangent line's slope. The tangent line is the polar of P with respect to Q. These two equations can be solved simultaneously for the (x, y) coordinates of the point P.

#### Solution

Define a function to construct the point(s) P based on the approach described above.

```
In[1]: NormalPoints2D[
    P0: Point2D[{x0_, y0_}],
    Q: Quadratic2D[a_, b_, c_, d_, e_, f_]] :=
    Module[{P, x, y, ln, eq1, eq2, ans},
    eq1 = Equation2D[Q, {x, y}];
    P = Point2D[{x, y}];
    ln = Line2D[P, -1/Slope2D[Line2D[P0, P]]];
    eq2 = TangentEquation2D[ln, Q];
    ans = Solve2D[{eq1, eq2}, {x, y}];
    ans = Select[ans, (IsReal2D[x /. #] &&
        IsReal2D[y /. #])&];
    Map[(P /. #)&, ans]] /;
    IsNumeric2D[{P0, Q}, NormalPoints2D]
```

Define a function to construct the normal lines using the normal points.

```
In[2]: NormalLines2D[
    P0:Point2D[{x0_, y0_}],
    Q:Quadratic2D[a_, b_, c_, d_, e_, f_]]:=
    Map[Line2D[#, Q]&, NormalPoints2D[P0, Q]] /;
    IsNumeric2D[{P0, Q}, NormalLines2D]
```

Select the minimum distance from the normal point(s).

```
In[3]: MinimumDistance2D[
    P0:Point2D[{x0_, y0_}],
    Q:Quadratic2D[a_, b_, c_, d_, e_, f_]]:=
    Min[Map[Distance2D[P0, #]&, NormalPoints2D[P0, Q]]] /;
    IsNumeric2D[{P0, Q}, MinimumDistance2D];
```

#### Discussion

Here we illustrate the solution with a numerical example. The following steps were computed using *Mathematica* Version 3.0.1. Version 4.0 computes the same points and lines without duplicating the solutions.

```
In[4]: p0 = Point2D[2., 2.];
q1 = Quadratic2D[Ellipse2D[{0, 0}, 2, 1, 0]];
pts = NormalPoints2D[p0, q1]
Out[4] {Point2D[{-1.92052, -0.279113}], Point2D[{-1.92052, -0.279113}],
Point2D[{1.38564, 0.72111}], Point2D[{1.38564, 0.72111}]}
In[5]: lns = NormalLines2D[p0, q1]
Out[5] {Line2D[-3.84103, -2.23291, -8], Line2D[-3.84103, -2.23291, -8],
Line2D[2.77128, 5.76888, -8], Line2D[2.77128, 5.76888, -8]}
In[6]: Sketch2D[{p0, pts, lns, q1}, CurveLength2D -> 6];
2
```



In[7]: MinimumDistance2D[p0, q1]

Out[7] 1.4188

# pb3pts.nb

#### Parabola Through Three Points

### **Exploration**

Show that the parabola passing through the points (0,0), (a,b) and (b,a) whose axis is parallel to the x-axis has vertex (h,k) and focal length f given by

$$h = \frac{(a^2 + ab + b^2)^2}{4ab(a+b)}, \ k = \frac{a^2 + ab + b^2}{2(a+b)} \ \text{ and } \ f = -\frac{ab}{4(a+b)}.$$

Furthermore, show that the quadratic representing the parabola is

$$(a+b) y^{2} + abx - (a^{2} + ab + b^{2}) y = 0.$$

## Approach

Create the equation of a parabola in standard position with variables (h, k) for the vertex point and f for the focal length. The three given points must satisfy the equation. Solve three equations in three unknowns (h, k and f). Find the quadratic representing the equation.

## Solution

Write the equation of the parabola in standard position.

In[1]: Clear[x, y, h, k, f];
eq1 = (y - k) ^2 == 4 f (x - h);

Solve for the constants.

```
\begin{split} In[2]: \mbox{ Clear[a,b];} \\ & \mbox{ans = Solve[Map[(eq1 /. \#)\&, \\ & & \{\{x \to 0, y \to 0\}, \\ & & \{x \to a, y \to b\}, \\ & & \{x \to b, y \to a\}\}\}, \\ & & \{h, k, f\}\} \ // \mbox{Simplify} \\ Out[2] \ & \Big\{ \Big\{ h \to \frac{(a^2 + a \, b + b^2)^2}{4 \, a \, b \, (a + b)}, \ f \to -\frac{a \, b}{4 \, (a + b)}, \ k \to \frac{a^2 + a \, b + b^2}{2 \, (a + b)} \Big\} \Big\} \end{split}
```

Form the quadratic representing the parabola.

```
In[3]: ql = Quadratic2D[eql /. ans[[1]], \{x, y\}] // SimplifyOut[3] Quadratic2D[0, 0, 1, \frac{ab}{a+b}, -\frac{a^2 + ab + b^2}{a+b}, 0]
```

Multiply through by (a + b) to arrive at the desired form of the equation.

```
In[4]: Equation2D[Map[(#*(a+b))&, q1], \{x, y\}]
Out[4] abx+(-a<sup>2</sup> - ab - b<sup>2</sup>) y+(a+b) y<sup>2</sup> == 0
```

#### Discussion

This is a plot of a numerical example with a = 2 and b = 3.



## pb4pts.nb

#### Parabola Through Four Points

## **Exploration**

Describe a method for finding the two parabolas passing through four points. Show that the technique produces the correct results for the points (2, 1), (-1, 1), (-2, -1) and (4, -3) by plotting the parabola and the four points.

## Approach

Form a quadratic, parameterized by the variable k, representing the pencil of quadratics passing through the four points. The first three coefficients of the quadratic, a, b, and c must satisfy the relationship  $b^2 = 4ab$  because the quadratic is a parabola. Solve the equation for k.

### Solution

Define a function that implements the approach.

```
In[1]: Quadratic2D[
    p1:Point2D[{x1_, y1_}],
    p2:Point2D[{x2_, y2_}],
    p3:Point2D[{x3_, y3_}],
    p4:Point2D[{x4_, y4_}],
    Parabola2D] :=
    Module[{q1, k, a, b, c},
        q1 = Quadratic2D[p1, p2, p3, p4, k, Pencil2D];
        {a, b, c} = List @@ Take[q1, 3];
        ans = Solve[b^2 == 4 * a * c, k];
        Map[(q1 /. #)&, ans]];
    }
}
```

#### Discussion

The following is a numerical example using the points specified.

In[4]: Sketch2D[{pts, par1}, CurveLength2D -> 20];



## pbang.nb

### Parabola Intersection Angle

#### **Exploration**

Show that the parabolas  $y^2 = ax$  and  $x^2 = by$  will cut each other at an angle  $\theta$  given by

$$\theta = -\tan^{-1}\left(\frac{1}{2}\frac{a^{1/3}}{b^{1/3}}\right) + \tan^{-1}\left(2\frac{b^{1/3}}{a^{1/3}}\right).$$

#### Approach

Find the (real) intersection points of the two parabolas (the origin point is an intersection, but the cut angle at the origin is  $\pi$ , so use one of the other (real) angles). Construct the polars to each parabola at the intersection point (the polars are the tangent lines). Find the angle between the polars.

#### Solution

Intersect the two parabolas.

```
\begin{split} & \text{In[1]: Clear[x, y, a, b];} \\ & \text{ ans = Solve[{y^2 == a * x, x^2 == b * y}, {x, y}] \\ & \text{Out[1]} \quad \{ \{x \to 0, y \to 0 \}, \\ & \quad \{x \to a^{1/3} b^{2/3}, y \to a^{2/3} b^{1/3} \}, \{x \to -(-1)^{1/3} a^{1/3} b^{2/3}, y \to (-1)^{2/3} a^{2/3} b^{1/3} \}, \\ & \quad \{x \to (-1)^{2/3} a^{1/3} b^{2/3}, y \to -(-1)^{1/3} a^{2/3} b^{1/3} \} \} \end{split}
```

The first solution is the origin, so it is excluded. The third and fourth solutions are imaginary, so they are ignored. The second solution is the desired one.

```
In[2]: p0 = Point2D[x, y] /. ans[[2]]
Out[2] Point2D[{a<sup>1/3</sup> b<sup>2/3</sup>, a<sup>2/3</sup> b<sup>1/3</sup>}]
```

Construct the two parabolas.

```
In[3]: \{parabl = Loci2D[q1 = Quadratic2D[y^2 = a + x, \{x, y\}]], parab2 = Loci2D[q2 = Quadratic2D[x^2 = b + y, \{x, y\}]] \}Out[3] \{ \{Parabola2D[\{0, 0\}, \frac{a}{4}, 0]\}, \{Parabola2D[\{0, 0\}, \frac{b}{4}, \frac{\pi}{2}]\} \}
```

Construct the tangent lines at the points.

```
\label{eq:In[4]: } \begin{split} &In[4]: \ \{l1=Line2D[p0,q1], l2=Line2D[p0,q2]\} \\ &Out[4] \ \{Line2D[-a, 2 \, a^{2/3} \, b^{1/3}, \, -a^{4/3} \, b^{2/3}], \, Line2D[2 \, a^{1/3} \, b^{2/3}, \, -b, \, -a^{2/3} \, b^{4/3}]\} \end{split}
```

Find the angle between the tangent lines.

```
In[5]: eq1 = Angle2D[11, 12]Out[5] - ArcTan\left[\frac{a^{1/3}}{2b^{1/3}}\right] + ArcTan\left[\frac{2a^{1/3}}{b^{1/3}}\right]
```

## Discussion

Here's an example with a = 1 and b = 2.

 $\label{eq:lin} {\tt In[6]: Sketch2D[{parab1, parab2, l1, l2, p0} /. {a -> 1, b -> 2}];}$ 



The angle is about 36 degrees.

```
In[7]: eq1 /. {a -> 1, b -> 2}
Out[7] -ArcTan[ 1/2 2<sup>1/3</sup>] + ArcTan[ 2<sup>2/3</sup>]
In[8]: (eq1 /. {a -> 1, b -> 2}) / Degree // N
Out[8] 36.145
```

# pbarch.nb

**Parabolic Arch** 

#### **Exploration**



Find the equation of the parabolic arch of base b and height h as shown in the figure. Assume that b and h are positive.

### Approach

Create a parabola rotated  $-\pi/2$  radians with variables (h, k) and f for the vertex point and focal length. Find the quadratic equation of the parabola. The three given points (0,0), (b/2, h) and (b, 0) must satisfy the equation. Solve three equations in the three unknowns h, k and f.

## Solution

Construct the parabola.

In[1]: Clear[h, k, f];
 par1 = Parabola2D[{h, k}, f, -Pi / 2];

Create the equation of the parabola.

```
In[2]: Clear[x, y];
eq1 = Equation2D[Quadratic2D[par1], {x, y}]
Out[2] h<sup>2</sup> - 4 f k - 2 h x + x<sup>2</sup> + 4 f y == 0
```

The three points must satisfy the equation of the parabola.

```
 \begin{array}{l} \mbox{In[3]: Clear[B, H];} \\ \mbox{ans = Solve[Map[(eq1 /. #)&, \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ &
```

Here's the equation of the parabolic arch.

In [4]: eq1 /. First[ans] Out [4] -B x + x<sup>2</sup> +  $\frac{B^2 y}{4 H} = 0$ 

### Discussion

This is an example of the arch with B = 4 and H = 3.

```
In[5]: Sketch2D[{parl /. First[ans] /. {B -> 4, H -> 3}},
CurveLength2D -> 9];
```



## pbarclen.nb

### Arc Length of a Parabola

#### **Exploration**

Show that the arc length, s, of a parabola whose parametric equations are

$$x = ft^2$$
 and  $y = 2ft$ 

is given by  $s = f(S_2 - S_1)$  where

$$S_n = t_n \sqrt{1 + t_n^2} + \sinh^{-1}(t_n).$$

## Approach

Directly apply the integral definition of arc length.

#### Solution

Compute the indefinite integral first. The results shown in the next few steps were computed using *Mathematica* Version 3.0.1. Version 4.0 computes slightly different results that are algebraically equivalent. Both versions compute the same final step.

```
In[1]: Clear[f, t];
I1 = Integrate[
Sqrt[D[f*t^2, t]^2 + D[2*f*t, t]^2],
t] // Simplify
Out[1] \sqrt{f^2 (1 + t^2)} \left(t + \frac{ArcSinh[t]}{\sqrt{1 + t^2}}\right)
```

Evaluate the indefinite integral at the limits.

```
In[2]: Clear[t1, t2];
s1 = (I1 /. t -> t2) - (I1 /. t -> t1) // Simplify
Out[2] -\sqrt{f^2 (1 + t1^2)} \left(t1 + \frac{ArcSinh[t1]}{\sqrt{1 + t1^2}}\right) + \sqrt{f^2 (1 + t2^2)} \left(t2 + \frac{ArcSinh[t2]}{\sqrt{1 + t2^2}}\right)
```

The focal length, f, is positive

```
In[3]: Clear[E1];
s2 = s1 /. Sqrt[f^2 * E1_] \rightarrow f * Sqrt[E1]
Out[3] -f \sqrt{1 + t1^2} \left(t1 + \frac{ArcSinh[t1]}{\sqrt{1 + t1^2}}\right) + f \sqrt{1 + t2^2} \left(t2 + \frac{ArcSinh[t2]}{\sqrt{1 + t2^2}}\right)
```

Simplify.

```
In[4]: s3 = Factor[s2]
```

 $Out[4] - f\left(t1\sqrt{1+t1^2} - t2\sqrt{1+t2^2} + ArcSinh[t1] - ArcSinh[t2]\right)$ 

In[5]: s4 = f \* Map[(-1 \* #)&, s3[[3]]]

 $Out[5] f(-t1\sqrt{1+t1^2} + t2\sqrt{1+t2^2} - ArcSinh[t1] + ArcSinh[t2])$ 

## pbdet.nb

#### Parabola Determinant

#### **Exploration**

Show that the determinant

$$\begin{vmatrix} y & x^2 & x & 1 \\ y_1 & x_1^2 & x_1 & 1 \\ y_2 & x_2^2 & x_2 & 1 \\ y_3 & x_3^2 & x_3 & 1 \end{vmatrix} = 0$$

represents a parabola  $Ax^2 + Dx + Ey + F = 0$  passing through the points  $(x_1, y_1)$ ,  $(x_2, y_2)$  and  $(x_3, y_3)$ .

## Approach

Expand the determinant. Convert it to a quadratic and show that the three points satisfy the equation.

### Solution

Expand the determinant and form a quadratic.

Form an equation of the quadratic.

```
In[2]: poly1 = Polynomial2D[q1, {x, y}]
Out[2] (x1<sup>2</sup> x2 - x1 x2<sup>2</sup> - x1<sup>2</sup> x3 + x2<sup>2</sup> x3 + x1 x3<sup>2</sup> - x2 x3<sup>2</sup>) y - x2<sup>2</sup> x3 y1 + x2 x3<sup>2</sup> y1 + x1<sup>2</sup> x3 y2 - x1 x3<sup>2</sup> y2 - x1<sup>2</sup> x2 y3 + x1 x2<sup>2</sup> y3 + x<sup>2</sup> (-x2 y1 + x3 y1 + x1 y2 - x3 y2 - x1 y3 + x2 y3) + x (x2<sup>2</sup> y1 - x3<sup>2</sup> y1 - x1<sup>2</sup> y2 + x3<sup>2</sup> y2 + x1<sup>2</sup> y3 - x2<sup>2</sup> y3)
```

Check if each of the points is on the quadratic.

## Discussion

This is a plot of a numerical example.

```
In[4]: pl = Point2D[{xl, yl}];
       p2 = Point2D[{x2, y2}];
       p3 = Point2D[{x3, y3}];
        Sketch2D[{p1, p2, p3, q1} //. {
         x1 \rightarrow 1, y1 \rightarrow 1, x2 \rightarrow 6, y2 \rightarrow -1,
         x3 -> 4, y3 -> 2}];
                              2
                              1
                              0
                            -1
                                               2
                                                                    5
                                                                          6
                                                      3
                                         1
                                                             4
                                  0
```

## pbfocchd.nb

### Length of Parabola Focal Chord

#### **Exploration**

Prove that the length of the focal chord of a parabola is 4f, where f is the focal length.

### Approach

Construct a parabola in a standard position. Construct a line perpendicular to the axis of the parabola through the focus point (the line containing the focal chord). Compute the distance between the points of intersection of the parabola and the line.

### Solution

Create the parabola.

```
In[1]: Clear[f1];
    par1 = Parabola2D[{0, 0}, f1, 0];
```

Construct the focus point.

In[2]: fpt = First[Foci2D[par1]]
Out[2] Point2D[{f1, 0}]

Construct a line perpendicular to the *x*-axis through the focus.

In[3]: fln = Line2D[fpt, Line2D[0, 1, 0], Perpendicular2D]
Out[3] Line2D[1, 0, -f1]

Intersect the line with the parabola.

```
In[4]: pts = Points2D[fln, par1]
Out[4] {Point2D[{fl, -2 fl}], Point2D[{fl, 2 fl}]}
```

The length of the focal chord is the distance between the points.

```
In[5]: Distance2D[Sequence @@ pts] /.
        Sqrt[f1^2] -> f1
Out[5] 4 f1
```

## pbslp.nb

### Tangent to a Parabola with a Given Slope

### **Exploration**

Show that the line tangent to the parabola  $y^2 = 4px$  with slope m is given by y = mx + p/m.

### Approach

Construct a line with slope m and use the function TangentLines2D[ln, quad] to construct the desired tangent line.

#### Solution

Construct a line with slope m.

Construct a line parallel to the line and tangent to the parabola. The tangent line has the form expected.

```
In[2]: Clear[p];
    12 = TangentLines2D[11,
        p1 = Parabola2D[{0,0},p,0]]
Out[2] {Line2D[m, -1, p/m]}
```

## Discussion

This is the plot of a numerical example.

In[3]: Sketch2D[{l2, pl} /. {p -> 1/2, m -> 2}];


## pbtancir.nb

#### Circle Tangent to a Parabola

#### **Exploration**

Any line through the point (-3a, 0) cuts the parabola  $y^2 = 4ax$  in the points P and Q. Prove that the circle through P, Q and the focus is tangent to the parabola.

## Approach

Construct the geometry and show that the lines tangent to the parabola and the circle at the intersection point are coincident.

#### Solution

Construct the point, parabola and a line through the point.

```
In[1]: Clear[a, m];
    p1 = Point2D[-3 a, 0];
    parabl = Parabola2D[{0, 0}, a, 0];
    l1 = Line2D[p1, m]
Out[1] Line2D[m, -1, 3 am]
```

Intersect the lines in pairs to find the intersection points, P and Q.

```
In[2]: \{P, Q\} = Points2D[11, parab1] // Simplify
Out[2] \{Point2D[\{\frac{-2\sqrt{a^2(1-3m^2)} + a(2-3m^2)}{m^2}, -\frac{2(-a+\sqrt{a^2(1-3m^2)})}{m}\}],
Point2D[\{\frac{2\sqrt{a^2(1-3m^2)} + a(2-3m^2)}{m^2}, \frac{2(a+\sqrt{a^2(1-3m^2)})}{m}\}]\}
```

Construct the circle through P, Q and the focus.

```
In[3]: fpt = Foci2D[parabl][[1]];
c1 = Circle2D[P, Q, fpt] // Simplify
Out[3] Circle2D\left[\left\{\frac{a(3+m^2)}{2m^2}, \frac{a-3am^2}{2m^3}\right\}, \frac{1}{2}\sqrt{\frac{a^2(1+m^2)^3}{m^6}}\right]
```

Intersect the circle and the parabola.

```
In[4]: pts = Points2D[c1, parab1] // Simplify
```

$$\begin{array}{l} \textit{Out[4]} & \left\{ \texttt{Point2D} \Big[ \left\{ \frac{a}{\mathfrak{m}^2} \,,\, -\frac{2\,a}{\mathfrak{m}} \right\} \Big] \,, \\ & \quad \texttt{Point2D} \Big[ \left\{ \frac{-2\,\sqrt{a^2\,\left(1-3\,\mathfrak{m}^2\right)}\,+a\,\left(2-3\,\mathfrak{m}^2\right)}{\mathfrak{m}^2} \,,\, -\frac{2\,\left(-a+\sqrt{a^2\,\left(1-3\,\mathfrak{m}^2\right)}\,\right)}{\mathfrak{m}} \,\right\} \Big] \\ & \quad \texttt{Point2D} \Big[ \left\{ \frac{2\,\sqrt{a^2\,\left(1-3\,\mathfrak{m}^2\right)}\,+a\,\left(2-3\,\mathfrak{m}^2\right)}{\mathfrak{m}^2} \,,\, \frac{2\,\left(a+\sqrt{a^2\,\left(1-3\,\mathfrak{m}^2\right)}\,\right)}{\mathfrak{m}} \,\right\} \Big] \right\} \end{array}$$

Two of the points are P and Q, as expected; the third must be the tangency point.

```
In[5]: pts = Points2D[c1, parab1] // Simplify
```

$$\begin{array}{l} \textit{Out[5]} & \left\{ \texttt{Point2D} \left[ \left\{ \frac{a}{\mathfrak{m}^{2}}, -\frac{2\,a}{\mathfrak{m}} \right\} \right], \\ & \texttt{Point2D} \left[ \left\{ \frac{-2\,\sqrt{a^{2}\,\left(1-3\,\mathfrak{m}^{2}\right)}\,+a\,\left(2-3\,\mathfrak{m}^{2}\right)}{\mathfrak{m}^{2}}, -\frac{2\,\left(-a+\sqrt{a^{2}\,\left(1-3\,\mathfrak{m}^{2}\right)}\,\right)}{\mathfrak{m}} \right\} \right] \\ & \texttt{Point2D} \left[ \left\{ \frac{2\,\sqrt{a^{2}\,\left(1-3\,\mathfrak{m}^{2}\right)}\,+a\,\left(2-3\,\mathfrak{m}^{2}\right)}{\mathfrak{m}^{2}}, \frac{2\,\left(a+\sqrt{a^{2}\,\left(1-3\,\mathfrak{m}^{2}\right)}\,\right)}{\mathfrak{m}} \right\} \right] \right\} \end{array}$$

Two of the points are P and Q, as expected; the third must be the tangency point.

```
In[6]: {IsCoincident2D[pts[[2]], P], IsCoincident2D[pts[[3]], Q]}
Out[6] {True, True}
```

Construct the tangents to the circle and the parabola at the intersection point.

The tangents are coincident, therefore, the parabola and the circle are tangent.

## Discussion

This is the plot of a numerical example.



# pbtnlns.nb

### Perpendicular Tangents to a Parabola

## **Exploration**

Show that if  $L_1$  and  $L_2$  are two lines tangent to a parabola that intersect on the directrix of the parabola, then  $L_1$  and  $L_2$  are perpendicular to each other.

## Approach

Since the shape (not the position or orientation) of the parabola is relevant, pick a parabola in standard position and a point on the parabola's directrix. Construct the tangent lines from the point to the parabola and show that the lines are perpendicular (i.e. their slopes are negative reciprocals).

### Solution

Create the parabola and its directrix.

```
In[1]: Clear[f];
    parab1 = Parabola2D[{0, 0}, f, 0];
    dln = First[Directrices2D[parab1]]
Out[1] Line2D[1, 0, f]
```

Construct a general point on the directrix.

```
In[2]: Clear[y];
    p1 = Point2D[-f, y];
```

Construct the two tangent lines from the point.

```
In[3]: \{11, 12\} = TangentLines2D[p1, parab1] // Simplify
Out[3] \{Line2D[2 f \sqrt{4 f^2 + y^2}, 4 f^2 + y (y - \sqrt{4 f^2 + y^2}), 2 f^2 (-2 y + \sqrt{4 f^2 + y^2}) + y^2 (-y + \sqrt{4 f^2 + y^2})], Line2D[-2 f \sqrt{4 f^2 + y^2}, 4 f^2 + y (y + \sqrt{4 f^2 + y^2}), -y^2 (y + \sqrt{4 f^2 + y^2}) - 2 f^2 (2 y + \sqrt{4 f^2 + y^2})]\}
```

Show that the slopes are negative reciprocal (therefore the lines are perpendicular to each other).

```
In[4]: Slope2D[11] * Slope2D[12] // Simplify
Out[4] -1
```

## Discussion

This is the plot of a numerical example.

```
In[5]: Sketch2D[{parabl, dln, p1, l1, l2} /.
    {f -> 1, y -> 2},
    CurveLength2D -> 20];
```



# polarcir.nb

## Polar Equation of a Circle

#### **Exploration**

Show that the polar equation of a circle centered at  $P(r_1, \theta_1)$  with radius R is given by

$$r^{2} + r_{1}^{2} - 2rr_{1}\cos(\theta - \theta_{1}) = R^{2}.$$

## Approach

Represent the circle in rectangular coordinates. Convert the equation to polar coordinates.

## Solution

Define a function to convert from polar coordinates to rectangular coordinates.

```
In[1]: Point2D[PolarPoint2D[r_, theta_]] :=
    Point2D[{r * Cos[theta], r * Sin[theta]}];
```

Create the circle.

```
In[2]: Clear[r1, t1, R];
    P = Point2D[PolarPoint2D[r1, t1]];
    C1 = Circle2D[P, R]
Out[2] Circle2D[{r1Cos[t1], r1Sin[t1]}, R]
```

Convert to a polynomial in polar coordinates.

```
In[3]: Clear[x, y, r, t];
eq1 = Polynomial2D[Quadratic2D[C1], {x, y}] /.
        {x -> r * Cos[t], y -> r * Sin[t]} // Fullsimplify
Out[3] r<sup>2</sup> - R<sup>2</sup> + rl<sup>2</sup> - 2 rrl Cos[t - t1]
```

# polarcol.nb

#### **Collinear Polar Coordinates**

#### **Exploration**

Show that the points  $P_1(r_1, \theta_1)$ ,  $P_2(r_2, \theta_2)$  and  $P_3(r_3, \theta_3)$  in polar coordinates are collinear if and only if

 $-r_1 r_2 \sin(\theta_1 - \theta_2) + r_1 r_3 \sin(\theta_1 - \theta_3) - r_2 r_3 \sin(\theta_2 - \theta_3) = 0.$ 

### Approach

Convert the given polar coordinates of the points to rectangular coordinates and then apply the condition for collinearity

| $x_1$ | $y_1$ | 1 |     |
|-------|-------|---|-----|
| $x_2$ | $y_2$ | 1 | =0. |
| $x_3$ | $y_3$ | 1 | -   |

## Solution

This is a function for converting a polar point to rectangular coordinates.

```
In[1]: Point2D[PolarPoint2D[r_, theta_]] :=
Point2D[{r * Cos[theta], r * Sin[theta]}];
```

Define three arbitrary points in polar coordinates.

```
In[2]: Clear[r1, r2, r3, theta1, theta2, theta3];
p1 = Point2D[PolarPoint2D[r1, theta1]];
p2 = Point2D[PolarPoint2D[r2, theta2]];
p3 = Point2D[PolarPoint2D[r3, theta3]];
```

Apply the condition for collinearity.

## Discussion

Here's a function based on the equation above that returns **True** if three points in polar coordinates are collinear.

```
In[4]: IsCollinear2D[
    p1:PolarPoint2D[r1_, theta1_],
    p2:PolarPoint2D[r2_, theta2_],
    p3:PolarPoint2D[r3_, theta3_]]:=
    IsZero2D[-r1*r2*Sin[theta1 - theta2] +
        r1*r3*Sin[theta1 - theta3] -
        r2*r3*Sin[theta2 - theta3]]
```

Show that the polar coordinate points  $(1, \pi/3)$ ,  $(3, \pi/3)$  and  $(5, 4\pi/3)$  are collinear using the new function.

```
In[5]: p1 = PolarPoint2D[1, Pi / 3];
    p2 = PolarPoint2D[3, Pi / 3];
    p3 = PolarPoint2D[5, 4 * Pi / 3];
    IsCollinear2D[p1, p2, p3]
```

```
Out[5] True
```

# polarcon.nb

#### Polar Equation of a Conic

#### **Exploration**

Let the focus F of a conic be at the pole of a polar coordinate system and the directrix D be perpendicular to the polar axis at a distance  $\rho$  to the left of the pole. Show that the polar equation of the conic is

$$r = \frac{e\rho}{1 - e\cos\theta}$$

where e is the eccentricity of the conic.

## Approach

Use the definition of eccentricity e = PF/PD and substitute the expressions for distances. Solve the resulting equations for r.

## Solution

Use the definition of eccentricity.

Substitute the distances for the segment lengths.

```
In[2]: Clear[r, p, t];
eq2 = eq1 /.
{PF -> r, PD -> p + r * Cos[t]}
Out[2] e == r
p + r Cos[t]
```

Solve for r.

 $\begin{array}{ll} & In[3]: \mbox{ Solve[eq2, r] // Simplify} \\ & Out[3] \ \left\{ \left\{ r \rightarrow \frac{e\,p}{1-e\,Cos\,[t]} \right\} \right\} \end{array} \end{array}$ 

## polardis.nb

#### **Distance Using Polar Coordinates**

#### **Exploration**

The location of a point in the plane may be specified using *polar coordinates*,  $(r, \theta)$ , where r is the distance from the origin to the point, and  $\theta$  is the angle the ray to the point from the origin makes with the +x-axis. Show that the distance, d, between two points  $(r_1, \theta_1)$  and  $(r_2, \theta_2)$  given in polar coordinates is

$$d = \sqrt{r_1^2 + r_2^2 - 2r_1r_2\cos(\theta_1 - \theta_2)}.$$

## Approach

Convert the given polar coordinates of the points to rectangular coordinates and then apply the Distance2D function to the converted points.

## Solution

This is a function for converting a polar point to rectangular coordinates.

```
In[1]: Point2D[PolarPoint2D[r_, theta_]] :=
Point2D[{r * Cos[theta], r * Sin[theta]}];
```

Define two arbitrary points in polar coordinates.

```
In[2]: Clear[r1, r2, theta1, theta2];
p1 = Point2D[P1 = PolarPoint2D[r1, theta1]];
p2 = Point2D[P2 = PolarPoint2D[r2, theta2]];
```

Find the distance between the two points.

In [3]: d = Distance2D[p1, p2] // Simplify Out [3]  $\sqrt{r1^2 + r2^2 - 2r1r2 \cos[\text{thetal} - \text{theta2}]}$ 

## Discussion

 $\tt Distance2D$  can be defined to handle polar coordinates directly as shown here.

```
In[4]: Distance2D[PolarPoint2D[r1_, theta1_],
            PolarPoint2D[r2_, theta2_]] :=
            Sqrt[r1^2 + r2^2 - r1 * r2 * Cos[theta1 - theta2]]
```

Try out the new Distance2D function.

In[5]: Distance2D[P1, P2] // Simplify

 $Out[5] \sqrt{rl^2 + r2^2 - rlr2 \cos[thetal - theta2]}$ 

# polarell.nb

## Polar Equation of an Ellipse

#### **Exploration**

Show that the polar equation of an ellipse with a horizontal major axis and centered at (0,0) is given by

$$r = \frac{ab}{\sqrt{a^2 \sin^2 \theta + b^2 \cos^2 \theta}}$$

where a and b are the lengths of the semi-major and semi-minor axes, respectively.

## Approach

Create the ellipse in rectangular coordinates. Convert the equation to polar coordinates.

## Solution

Create a quadratic representing the ellipse.

```
In[1]: Clear[a, b];
Q1 = Quadratic2D[Ellipse2D[{0, 0}, a, b, 0]]
Out[1] Quadratic2D[b<sup>2</sup>, 0, a<sup>2</sup>, 0, 0, -a<sup>2</sup> b<sup>2</sup>]
```

Convert the rectangular equation to a polar equation.

```
In[2]: Clear[x, y, r, theta]
eq1 = Equation2D[Q1, {x, y}] /.
        {x -> r * Cos[theta], y -> r * Sin[theta]}
Out[2] -a<sup>2</sup>b<sup>2</sup> + b<sup>2</sup>r<sup>2</sup> Cos[theta]<sup>2</sup> + a<sup>2</sup>r<sup>2</sup> Sin[theta]<sup>2</sup> == 0
```

Put into the desired form by solving for r (taking the positive result).

In[3]: Solve[eq1, r]

$$\begin{array}{l} \textit{Out[3]} & \left\{ \left\{ r \rightarrow -\frac{a \, b}{\sqrt{b^2 \, \text{Cos} \left[ \text{theta} \right]^2 + a^2 \, \text{Sin} \left[ \text{theta} \right]^2} \right\} \right\} \\ & \left\{ r \rightarrow \frac{a \, b}{\sqrt{b^2 \, \text{Cos} \left[ \text{theta} \right]^2 + a^2 \, \text{Sin} \left[ \text{theta} \right]^2}} \right\} \right\} \end{array}$$

## polareqn.nb

## **Polar Equations**

### **Exploration**

A curve in polar coordinates may have more than one equation. A given point may have either of two general coordinate representations

$$(r, \theta + 2k\pi)$$
$$(-r, \theta + (2k+1)\pi)$$

for any integer k. Hence a given curve  $r = f(\theta)$  may have either of the two equation forms

$$r = f(\theta + 2k\pi)$$
$$-r = f(\theta + (2k+1)\pi).$$

The first equation reduces to  $r = f(\theta)$  when k = 0, but may lead to an entirely different equation of the same curve for another value of k. Similarly, the second equation may yield other equations of the curve. Show that in spite of the potential for multiple equations in polar coordinates, a linear equation Ax + By + C = 0 has only one representation in polar coordinates given by

$$r(A\cos\theta + B\sin\theta) + C = 0.$$

#### Approach

Derive an equation for a linear equation in polar coordinates using the primary form  $(r, \theta)$ . Investigate and compare the primary form to the equation derived from the forms  $(r, \theta + 2k\pi)$ and  $(-r, \theta + (2k + 1)\pi)$ .

## Solution

Create the primary form of a linear equation in polar coordinates.

Compare to the form  $(r, \theta + 2k\pi)$ , using two trigonometric identities.

```
In[2]: Clear[r,t,k];
Al*x+Bl*y+Cl //.
{x->r*Cos[t+2k*Pi],
y->r*Sin[t+2k*Pi],
Cos[t+2k*Pi]->Cos[t],
Sin[t+2k*Pi]->Sin[t]}
Out[2] Cl+AlrCos[t]+BlrSin[t]
```

Compare to the form  $(-r, \theta + (2k+1)\pi)$ , using two trigonometric identities.

```
In[3]: A1*x+B1*y+C1 //.
    {x -> -r*Cos[t+(2k+1)*Pi],
        y -> -r*Sin[t+(2k+1)*Pi],
        Cos[t+(2k+1)*Pi] -> -Cos[t],
        Sin[t+(2k+1)*Pi] -> -Sin[t]}
Out[3] C1+A1rCos[t]+B1rSin[t]
```

## polarhyp.nb

### Polar Equation of a Hyperbola

#### **Exploration**

Show that the polar equation of a hyperbola with a horizontal transverse axis and centered at (0,0) is given by

$$r = \frac{ab}{\sqrt{b^2 \cos^2\theta - a^2 \sin^2\theta}}.$$

## Approach

Create the hyperbola in rectangular coordinates and convert the equation to polar coordinates.

## Solution

Define a quadratic representing the hyperbola.

```
In[1]: Clear[a, b];
Q1 = Quadratic2D[Hyperbola2D[{0, 0}, a, b, 0]]
Out[1] Quadratic2D[b<sup>2</sup>, 0, -a<sup>2</sup>, 0, 0, -a<sup>2</sup>b<sup>2</sup>]
```

Convert from rectangular to polar coordinates.

```
In[2]: Clear[x, y, r, theta];
eq1 = Equation2D[Q1, {x, y}] /.
{x -> r * Cos[theta], y -> r * Sin[theta]}
Out[2] -a<sup>2</sup> b<sup>2</sup> + b<sup>2</sup> r<sup>2</sup> Cos[theta]<sup>2</sup> - a<sup>2</sup> r<sup>2</sup> Sin[theta]<sup>2</sup> == 0
```

Solve for r to put the equation into the desired form. This result was computed using Mathematica Version 3.0.1. Version 4.0 produces a slightly different result that is algebraically equivalent with  $\sqrt{-1}$  already factored out.

$$\begin{split} &In[3]: \text{ ans = Solve[eq1, r]} \\ &Out[3] \; \left\{ \left\{ r \rightarrow -\frac{I \, a \, b}{\sqrt{-b^2 \, \text{Cos} \left[ \text{theta} \right]^2 + a^2 \, \text{Sin} \left[ \text{theta} \right]^2}} \right\}, \\ & \left\{ r \rightarrow \frac{I \, a \, b}{\sqrt{-b^2 \, \text{Cos} \left[ \text{theta} \right]^2 + a^2 \, \text{Sin} \left[ \text{theta} \right]^2}} \right\} \right\} \end{split}$$

Multiply the fraction by  $i = \sqrt{-1}$  to get the desired form.

$$\begin{split} & \text{In[4]: Clear[E1, E2];} \\ & \text{Last[ans] /. {I * E1_ / Sqrt[E2_] -> E1 / Sqrt[-E2]}} \\ & \text{Out[4] } \left\{ r \rightarrow \frac{a b}{\sqrt{b^2 \cos[\text{theta}]^2 - a^2 \sin[\text{theta}]^2}} \right\} \end{split}$$

# polarpb.nb

## Polar Equation of a Parabola

## **Exploration**

Show that the polar equation of a parabola opening to the right with vertex at (0,0) is given by

$$r = \frac{4f\cos\theta}{\sin^2\theta}$$

where f is the focal length of the parabola.

## Approach

Create the parabola in rectangular coordinates. Convert the equation to polar coordinates.

## Solution

Construct the quadratic representing the parabola.

In[1]: Clear[f]; Q1 = Quadratic2D[Parabola2D[{0, 0}, f, 0]] Out[1] Quadratic2D[0, 0, 1, -4 f, 0, 0]

Convert the equation from rectangular coordinates to polar coordinates.

```
In[2]: Clear[x, y, r, theta];
eq1 = Equation2D[Q1, {x, y}] /.
        {x -> r * Cos[theta], y -> r * Sin[theta]}
Out[2] -4 fr Cos[theta] + r<sup>2</sup> Sin[theta]<sup>2</sup> == 0
```

Solve for r to get the desired form of the equation.

In[3]: Solve[eq1, r]

 $Out[3] \{ \{r \rightarrow 0\}, \{r \rightarrow 4 \text{ f Cot}[\text{theta}] Csc[\text{theta}] \} \}$ 

The trigonometric identity  $4f \cot(\theta) \csc(\theta) = 4f \cos(\theta) / \sin^2(\theta)$  completes the demonstration.

In[4]: 4f\*Cos[theta]/Sin[theta]^2 // Simplify

Out[4] 4fCot[theta] Csc[theta]

## polarunq.nb

### Non-uniqueness of Polar Coordinates

#### **Exploration**

Show that the polar coordinates of a point  $(r, \theta)$  are not unique as all points of the form

 $(r, \theta + 2k\pi)$  and  $(-r, \theta + (2k+1)\pi)$ 

represent the same position in the plane for integer values of k.

## Approach

Convert the given polar coordinates of the points to rectangular coordinates and demonstrate that the rectangular coordinates are coincident.

### Solution

This is a function for converting a polar point to rectangular coordinates.

```
In[1]: Point2D[PolarPoint2D[r_, theta_]] :=
Point2D[{r * Cos[theta], r * Sin[theta]}];
```

Convert the two points to rectangular coordinates.

```
In[2]: Clear[r, theta, k];
    pts = {Point2D[PolarPoint2D[r, theta + 2 k * Pi]],
        Point2D[PolarPoint2D[-r, theta + (2 k + 1) Pi]]};
```

Simplifying shows that the points are identical for all values of k.

### Discussion

The *principal* polar coordinates of a point  $(r, \theta)$  are given when r > 0 and  $0 \le \theta < 2\pi$ . These functions convert a **PolarPoint2D** to principal coordinates.

```
In[4]: PolarPoint2D[r_?IsNegative2D, theta_] :=
    PolarPoint2D[-r, theta + Pi];
    PolarPoint2D[r_, theta_] :=
        PolarPoint2D[r, PrimaryAngle2D[theta]] /;
        theta =! = PrimaryAngle2D[theta]
```

Convert some polar points to principal form.

```
In[5]: \{ PolarPoint2D[-1, Pi/2], PolarPoint2D[2, -Pi/3] \}Out[5] \{ PolarPoint2D[1, \frac{3\pi}{2}], PolarPoint2D[2, \frac{5\pi}{3}] \}
```

# pquad.nb

## Parameterization of a Quadratic

### **Exploration**

Show that the quadratic  $Q \equiv ax^2 + bxy + cy^2 + dx + ey = 0$ , that passes through the origin, can be parameterized by the equations

$$x(t) = -\frac{d+et}{a+t(b+ct)}$$
 and  $y(t) = -\frac{t(d+et)}{a+t(b+ct)}$ 

where  $-\infty < t < +\infty$ .

## Approach

Let the parameter, t, be the slope of a line, L, passing through the origin. The coordinates of the point P(x(t), y(t)), which is the desired parameterization, is the intersection point of L with Q that is not coincident with the origin.

#### Solution

Define a function that returns parametric equations of a quadratic, given a point on the quadratic, in terms of a parameter, t.

If the point on the quadratic is the origin, (0, 0), then the equations are given by the following.

```
In[2]: Clear[a, b, c, d, e, t];
Parameterize2D[Quadratic2D[a, b, c, d, e, 0], Point2D[{0, 0}], t] // Simplify
Out[2] \left\{-\frac{d+et}{a+t(b+ct)}, -\frac{t(d+et)}{a+t(b+ct)}\right\}
```

#### Discussion

As an example, parameterize the quadratic  $5x^2 - 3\sqrt{3}xy + 4y^2 - 8x - 14y = 0$ .

```
In[3]: Clear[t];
Q = Quadratic2D[5, -3 * Sqrt[3], 4, -8, -14, 0];
XtYt = Parameterize2D[Q, Point2D[{0, 0}], t]
Out[3] { <math>\frac{2 (4+7t)}{5-3 \sqrt{3} t+4 t^2}, \frac{2 (4t+7t^2)}{5-3 \sqrt{3} t+4 t^2} }
```

Plot the quadratic using the parametric equations. Notice the gap in the graph as the parameter, t, approaches  $\pm \infty$ .

Determine the locus of the quadratic in standard form.

```
In[5]: crv = Loci2D[Q] // N
Out[5] {Ellipse2D[{2.58012, 3.42583}, 4.30102, 2.19094, 0.880461]}
```

An identical graph is produced from the equation in standard form. The gap is not present in this plot because the trigonometric parameterization of the ellipse, used to plot the standard form, avoids passing through infinity.

2345

In[6]: Sketch2D[{crv}];



# ptscol.nb

## **Collinear Points**

## **Exploration**

Show that the three points (3a, 0), (0, 3b) and (a, 2b) are collinear.

## Approach

Three points  $P_1(x_1, y_1)$ ,  $P_2(x_2, y_2)$  and  $P_3(x_3, y_3)$  are collinear if the determinant

| $ x_1 $ | $y_1$ | 1 |
|---------|-------|---|
| $x_2$   | $y_2$ | 1 |
| $x_3$   | $y_3$ | 1 |

is zero.

## Solution

Use the Mathematica Det command to evaluate the determinant.

## Discussion

The function  $\tt IsCollinear2D$  also reveals if three points are collinear.

```
In[2]: IsCollinear2D[
    p1 = Point2D[3 a, 0],
    p2 = Point2D[0, 3 b],
    p3 = Point2D[a, 2 b]]
Out[2] True
```

This is the plot of a numerical example.

```
In[3]: Sketch2D[{p1, p2, p3,
Line2D[p1, p3]} /. {a -> 2, b -> 1.75}];
```

# radaxis.nb

## Radical Axis of Two Circles

#### **Exploration**

Show that the two circles  $x^2 + y^2 + ax + by + c = 0$  and  $x^2 + y^2 + bx + ay + c = 0$  have the radical axis x - y = 0.

## Approach

Convert the equations to circles and find the radical axis of the circles.

## Solution

Construct the circles from the equations.

```
In[1]: Clear[a, b, c]; \\ \{C1, C2\} = \{Circle2D[Quadratic2D[1, 0, 1, a, b, c]], \\ Circle2D[Quadratic2D[1, 0, 1, b, a, c]]\} \\ Out[1] \{Circle2D[\{-\frac{a}{2}, -\frac{b}{2}\}, \frac{1}{2}\sqrt{a^2 + b^2 - 4c}], Circle2D[\{-\frac{b}{2}, -\frac{a}{2}\}, \frac{1}{2}\sqrt{a^2 + b^2 - 4c}]\} \end{cases}
```

Construct the radical axis.

In[2]: L1 = Line2D[C1, C2] // Simplify
Out[2] Line2D[1, -1, 0]

Convert the line to an equation.

In[3]: Clear[x, y]; Equation2D[L1, {x, y}] Out[3] x - y == 0

## Discussion

This is a plot of a numerical example with a = 1, b = 5 and c = -1.

In[4]: Sketch2D[{C1, C2, L1} /. {
 a -> 1, b -> 5, c -> -1}];



# radcntr.nb

## **Radical Center**

### **Exploration**

Prove that the radical axes of three circles taken in pairs intersect in a common point. This point is called the *radical center* of the three circles.

## Approach

Create the three radical axes, intersect them in pairs and show that the coordinates of the points of intersection are equal.

## Solution

Create three general circles.

```
In[1]: Clear[h1, k1, r1, h2, k2, r2, h3, k3, r3];
C1 = Circle2D[{h1, k1}, r1];
C2 = Circle2D[{h2, k2}, r2];
C3 = Circle2D[{h3, k3}, r3];
```

Construct the radical axis lines in pairs.

In[2]: L12 = Line2D[C1, C2]; L13 = Line2D[C1, C3]; L23 = Line2D[C2, C3];

Intersect the lines in pairs to find the intersection points.

Show that the coordinates of the intersection points are equal.

## Discussion

This is the plot of a numerical example.



# raratio.nb

#### Radical Axis Ratio

#### **Exploration**

Show that the point of intersection of the radical axis and the line of centers of two circles of radii  $r_1$  and  $r_2$  divides the segment between the two centers into the ratio

$$\frac{d^2 + r_1^2 - r_2^2}{d^2 - r_1^2 + r_2^2}$$

where d is the distance between the centers.

## Approach

Create the two circles in a simplified, but sufficiently general, position. Construct the radical axis and intersect it with the line segment between the centers. Inspect the appropriate ratio.

## Solution

Create the two circles, one with center at the origin, the other with center at (d, 0).

```
In[1]: Clear[r1, r2, d];
c1 = Circle2D[{0, 0}, r1];
c2 = Circle2D[{d, 0}, r2];
```

Construct the radical axis of the two circles.

```
In[2]: 11 = Line2D[c1, c2]
Out[2] Line2D[2d, 0, -d<sup>2</sup> - r1<sup>2</sup> + r2<sup>2</sup>]
```

Intersect the radical axis with the x-axis to find the point of division.

```
In[3]: pt = Point2D[11, Line2D[0, 1, 0]]Out[3] Point2D[\{\frac{d^2 + r1^2 - r2^2}{2d}, 0\}]
```

Form the desired ratio.

```
In[4]: ratio1 = Distance2D[Point2D[0, 0], pt] / Distance2D[pt, Point2D[d, 0]] // SimplifyOut[4] \frac{\sqrt{\frac{(d^2+r1^2-r22)^2}{d^2}}}{\sqrt{\frac{(d^2-r1^2+r22)^2}{d^2}}}
```

Since all the expressions under the radical are positive, we can simplify the radicals.

```
In[5]: Clear[E1, E2];
ratio2 = ratio1 //. {
Sqrt[E1_^2/E2_^2] -> E1/E2,
1/Sqrt[E1_^2/E2_^2] -> E2/E1}
Out[5] \frac{d^2 + r1^2 - r2^2}{d^2 - r1^2 + r2^2}
```

## reccir.nb

#### **Reciprocal of a Circle**

#### **Exploration**

Given a circle  $C_1 \equiv (x-h)^2 + (y-k)^2 = r^2$  show that its polar reciprocal in the auxiliary conic  $x^2 + y^2 = 1$  is given by the quadratic

$$Q \equiv (r^{2} - h^{2}) x^{2} - 2hkxy + (r^{2} - k^{2}) y^{2} + 2hx + 2ky - 1 = 0.$$

Furthermore, show that Q is an *ellipse* if the origin (0,0) is inside C; a *parabola*, if the origin is on C; and a *hyperbola*, if the origin is outside C.

### Approach

Create a general circle and the auxiliary conic. Construct five points on the circle. Construct five tangent lines at the points. Construct reciprocals of the lines (five points). Construct a quadratic through five points. Examine the discriminant of the quadratic.

## Solution

Create a general circle and an auxiliary circle.

Define five points on the circle.

```
\begin{split} In[2]: \ \texttt{pts} &= \texttt{Map}[\\ & \texttt{Point2D[cir1[#]]&,} \\ & \{\texttt{0, Pi/4, Pi/2, Pi, 3Pi/2\}] \ \textit{// Simplify} \\ \\ Out[2] \ \Big\{\texttt{Point2D[\{h+r, k\}], Point2D[\{h + \frac{r}{\sqrt{2}}, \, k + \frac{r}{\sqrt{2}}\}\Big], \ \texttt{Point2D[\{h, k+r\}],} \\ & \texttt{Point2D[\{h-r, k\}], Point2D[\{h, k-r\}]} \Big\} \end{split}
```

Determine the tangent lines at the points. This result was computed using *Mathematica* Version 3.0.1. Version 4.0 computes a simpler result that is algebraically equivalent to this one.

```
In[3]: lns1 = Map[Line2D[#, cir1]&, pts] // Simplify
Out[3] {Line2D[1, 0, -h - r], Line2D[\sqrt{2}, \sqrt{2}, -\sqrt{2} h - \sqrt{2} k - 2 r], Line2D[0, 1, -k - r],
Line2D[-1, 0, h - r], Line2D[0, -1, k - r]}
```

Define the reciprocal function.

```
In[4]: Reciprocal2D[
    Line2D[A1_, B1_, C1_],
    Circle2D[{0, 0}, 1]] :=
    Point2D[{-A1/C1, -B1/C1};
```

Find the reciprocal points.

```
In[5]: ptsl = Map[Reciprocal2D[#, c1]&, lnsl] // SimplifyOut[5] \{Point2D[\{\frac{1}{h+r}, 0\}], Point2D[\{\frac{\sqrt{2}}{\sqrt{2}h+\sqrt{2}k+2r}, \frac{\sqrt{2}}{\sqrt{2}h+\sqrt{2}k+2r}\}],Point2D[\{0, \frac{1}{k+r}\}], Point2D[\{\frac{1}{h-r}, 0\}], Point2D[\{0, \frac{1}{k-r}\}]\}
```

Find the quadratic through the points.

```
In[6]: q1 = Quadratic2D[Sequence @@ pts1] // Simplify;
Map[(-1*#/2)&, q1]
Out[6] Quadratic2D[(h-r) (h+r), 2hk, -(-k+r) (k+r), -2h, -2k, 1]
```

#### Discussion

Examine the discriminant, d.

```
In[7]: disc1 = q1[[2]]^2 - 4*q1[[1]]*q1[[3]] // Simplify
Out[7] 16 r<sup>2</sup> (h<sup>2</sup> + k<sup>2</sup> - r<sup>2</sup>)
```

If d < 0 the quadratic is an ellipse and (0,0) is inside the circle; if d = 1 the quadratic is a parabola and (0,0) is on the circle; and if d > 1 the quadratic is a hyperbola and (0,0) is outside the circle.
## recptln.nb

#### **Reciprocals of Points and Lines**

#### **Exploration**

Show that the polar reciprocal of  $A_1x + B_1y + C_1 = 0$  in the auxiliary conic  $C \equiv x^2 + y^2 = 1$  is the point  $(-A_1/C_1, -B_1/C_1)$ , assuming that the line does not pass through the origin. Also, show that the line x + y - 1 = 0 is the polar reciprocal of the point (x, y) with respect to C.

## Approach

Create the auxiliary conic, C. The pole point is the reciprocal of the line. The polar line is the reciprocal of the point.

### Solution

Define the auxiliary conic (circle), C.

In[1]: c1 = Circle2D[{0, 0}, 1];

The pole point is the reciprocal.

In [2]: Clear[Al, Bl, Cl];Point2D[Line2D[Al, Bl, Cl], cl]  $Out [2] Point2D[\left\{-\frac{A1}{C1}, -\frac{B1}{C1}\right\}]$ 

The polar line is the reciprocal.

```
In[3]: Clear[x, y];
Line2D[Point2D[x, y], c1] // Simplify
Out[3] Line2D[x, y, -1]
```

## recquad.nb

### Reciprocal of a Quadratic

#### **Exploration**

Given the general quadratic  $Q = ax^2 + bxy + cy^2 + dx + ey + f = 0$ , show that the reciprocal of Q in C is the quadratic

$$(4cf - e^{2})x^{2} + (2de - 4bf)xy + (4af - d^{2})y^{2} + (4cd - 2be)x + (4ae - 2bd)y + (4ac - b^{2}) = 0$$

when the auxiliary conic  $C \equiv x^2 + y^2 = 1$ .

### Approach

Create a general conic, Q, and the auxiliary conic. Construct a point  $P_1(x_1, y_1)$ , assumed to be on Q. Construct the tangent line, L, at  $P_1$ . Take the reciprocal of L with respect to C, producing  $P_2$ . Show that  $P_2$  is on the postulated quadratic.

## Solution

Create a general quadratic.

```
In[1]: Clear[a, b, c, d, e, f];
q1 = Quadratic2D[a, b, c, d, e, f];
```

The point  $P_1(x_1, y_1)$  is a point on Q, and L is tangent to Q at  $P_1$ .

```
In[2]: Clear[x1, y1];
    p1 = Point2D[x1, y1];
    l1 = Line2D[p1, q1]
Out[2] Line2D[d+2ax1+by1, e+bx1+2cy1, 2f+dx1+ey1]
```

Find the auxiliary conic (a unit circle at the origin).

In[3]: c1 = Circle2D[{0, 0}, 1];

Define the reciprocal function.

```
In[4]: Reciprocal2D[
    Line2D[A1_, B1_, C1_],
    Circle2D[{0, 0}, 1]] :=
    Point2D[-A1/C1, -B1/C1];
```

Find the reciprocal of L.

 $In[5]: \mathbf{p2} = \mathbf{Reciprocal2D[11, c1]}$  $Out[5] \quad Point2D\left[\left\{-\frac{d+2 \, a \, x1 + b \, y1}{2 \, f + d \, x1 + e \, y1}, -\frac{e + b \, x1 + 2 \, c \, y1}{2 \, f + d \, x1 + e \, y1}\right\}\right]$ 

Find the reciprocal quadratic.

Construct a polynomial.

```
In[7]: eq1 = Polynomial2D[q2, Coordinates2D[p2]] // Together
```

```
\begin{array}{l} {\it Out\,[7]} & - \frac{1}{{{\left( {2\,\,f + d\,x1 + e\,y1} \right)}^2}}\left( {4\,\left( {c\,\,d^2\,f - b\,d\,e\,f + a\,e^2\,f + b^2\,f^2 - 4\,a\,c\,f^2 + c\,\,d^3\,x1 - b\,\,d^2\,e\,x1 + a\,d^2\,e\,x1 + a\,d\,e^2\,x1 + a\,d\,e^2\,x1 - 4\,a\,c\,d\,f\,x1 + a\,c\,\,d^2\,x1^2 - a\,b\,d\,e\,x1^2 + a^2\,e^2\,x1^2 + a\,b^2\,f\,x1^2 - 4\,a^2\,c\,f\,x1^2 + c\,\,d^2\,e\,y1 - b\,d\,e^2\,y1 + a\,e^3\,y1 + b^2\,e\,f\,y1 - 4\,a\,c\,e\,f\,y1 + b\,c\,d^2\,x1\,y1 - b^2\,d\,e\,x1\,y1 + a\,b\,e^2\,x1\,y1 + b^3\,f\,x1\,y1 - 4\,a\,b\,c\,f\,x1\,y1 + c^2\,d^2\,y1^2 - b\,c\,d\,e\,y1^2 + a\,c\,e^2\,y1^2 + b^2\,c\,f\,y1^2 - 4\,a\,c^2\,f\,y1^2 \right)} \right) \end{array}
```

Ignore the denominator and the constant (the numerator will be shown to be zero).

```
In[8]: eq2 = Numerator[eq1][[2]]
```

```
 \begin{array}{l} \textit{Out [8]} \quad c\ d^2\ f - b\ d\ e\ f + a\ e^2\ f + b^2\ f^2 - 4\ a\ c\ f^2 + c\ d^3\ x1 - b\ d^2\ e\ x1 + a\ d\ e^2\ x1 + b^2\ d\ f\ x1 - 4\ a\ c\ d\ f\ x1 - b^2\ d\ f\ x1 - b^2\ d\ f\ x1 - b^2\ d\ f\ x1 + a\ d^2\ y1 - b\ d\ e\ x1 + b^2\ d\ f\ x1 + b^2\ d\ f\ x1 - b^2\ d\ f\ x1 + b^2\ d\ x
```

Factor.

```
In[9]: eq3 = Factor[eq2]
```

 $\textit{Out[9]} \quad (c \ d^2 - b \ d \ e + a \ e^2 + b^2 \ f - 4 \ a \ c \ f) \quad (f + d \ x1 + a \ x1^2 + e \ y1 + b \ x1 \ y1 + c \ y1^2)$ 

One of the terms is zero, therefore the expression is zero.

```
In[10]: eq3 /.
    (f + d x1 + a x1^2 + e y1 + b x1 y1 +
        c y1^2) -> 0
Out[10] 0
```

## reflctpt.nb

#### **Reflection in a Point**

#### **Exploration**

A point P'(x', y') is said to be the reflection of a point P(x, y) in the point C(H, K) if C is the midpoint of the segment PP'. Using this definition show the following.

A. The transformation equations for a reflection in a point are

$$x' = 2H - x$$
 and  $x = 2H - x';$   
 $y' = 2K - y$  and  $y = 2K - y';$ 

B. The reflection of the line ax + by + c = 0 in the point (H, K) is

$$ax + by - (2aH + 2bK + c) = 0;$$

C. The reflection of the quadratic  $ax^2 + bxy + cy^2 + dx + ey + f = 0$  in the point (H, K) is

$$ax^{2} + bxy + cy^{2} - (4aH + 2bK + d)x - (2bH + 4cK + e)y - 4aH^{2} + 4bHK + 4cK^{2} + 2dH + 2eK + f = 0.$$

Also, verify that the reflection in a point transformation is equivalent to a rotation of  $\pi$  radians about the reflection point (H, K).

## Approach

Solve the midpoint relationship for the coordinates of the transformation. Substitute the reflected coordinates into the equation of a line to produce a reflected line. Substitute the reflected coordinates into the equation of a quadratic to produce the reflected quadratic. Apply the proposed rotation to show it is equivalent to the reflection.

#### Solution

(H, K) is the midpoint of PP'. Solve for (x, y) and (x', y'). This is the solution to proposition A.

```
In[1]: Clear[x, y, x1, y1, H, K];
        {{Solve[(x + x1) / 2 == H, x1],
        Solve[(y + y1) / 2 == K, y1]},
        {Solve[(x + x1) / 2 == H, x],
        Solve[(y + y1) / 2 == K, y]}
Out[1] {{{{x1 - 2H - x}}, {{y1 - 2K - y}}}, {{x - 2H - x1}}, {{y - 2K - y1}}}}
```

Reflect a line through a point. This is the solution to proposition B.

```
In[2]: Clear[a, b, c];
eq1 = a * x + b * y + c /. {x -> 2 H - x, y -> 2 K - y};
Map[Times[-1, #]&, Line2D[eq1, {x, y}]]
Out[2] Line2D[a, b, -c - 2 a H - 2 b K]
```

Reflect a quadratic through a point. This is the solution to proposition C.

```
In[3]: Clear[d, e, f];
eq2 = a * x^2 + b * x * y + c * y^2 + d * x + e * y + f /. {x -> 2 H - x, y -> 2 K - y};
Quadratic2D[eq2, {x, y}]
Out[3] Quadratic2D[a, b, c, -d - 4 a H - 2 b K, -e - 2 b H - 4 c K,
f + 2 d H + 4 a H<sup>2</sup> + 2 e K + 4 b H K + 4 c K<sup>2</sup>]
```

The reflection is the same as the specified rotation. This is the solution to the final proposition.

```
In[4]: Rotate2D[{x, y}, Pi, {H, K}]
Out[4] {2H-x, 2K-y}
```

## rtangcir.nb

### Angle Inscribed in a Semicircle

### **Exploration**

Show that an angle inscribed in a semicircle is a right angle.

## Approach

Find the parametric coordinates of the points that define the angle and use the Pythagorean Theorem to show they form a right angle.

### Solution

Create a circle at the origin.

In[1]: Clear[r]; C1 = Circle2D[{0, 0}, r];

Construct the points on the semicircle.  $P_1$  and  $P_3$  are the end points of the semicircle,  $P_2$  is the (right) angle vertex.

```
In[2]: Clear[t];
    P1 = C1[0];
    P2 = C1[t];
    P3 = C1[Pi];
```

Apply the Pythagorean Theorem. First compute  $a^2 + b^2$  and then show it is equal to  $c^2$  and independent of the parameter value of the vertex point (it turns out that it is a function of the circle's radius only).

```
In[3]: {Distance2D[P1, P2] ^2 + Distance2D[P2, P3] ^2,
Distance2D[P1, P3] ^2} // Simplify
Out[3] {4 r<sup>2</sup>, 4 r<sup>2</sup>}
```

# rttricir.nb

## Circle Inscribed in a Right Triangle

## **Exploration**

Show that if r is the radius of a circle inscribed in a right triangle with sides a and b and hypotenuse c, then r = (a + b - c)/2.

## Approach

Position the triangle so that the sides of length a and b align with the x- and y-axes and the vertex opposite the hypotenuse is at the origin. Create the circle inscribed in this triangle and examine its radius.

## Solution

The radius of the inscribed circle is found here.

```
In [1]: Clear[a, b];

r1 = Radius2D[

Circle2D[

Triangle2D[{a, 0}, {0, b}, {0, 0}],

Inscribed2D]] // Simplify

Out [1] \frac{\sqrt{a^2 + b^2 + \sqrt{a^2} \sqrt{b^2} - \sqrt{a^2} \sqrt{a^2 + b^2} - \sqrt{b^2} \sqrt{a^2 + b^2}}}{\sqrt{2}}
```

Simplify the expression for the radius.

```
In[2]: Clear[c]; 
r2 = r1 //. {
Sqrt[a^2] -> a,
Sqrt[b^2] -> b,
Sqrt[c^2] -> c,
a^2+b^2-> c^2 // Simplify
Out[2] <math>\frac{\sqrt{(a-c)(b-c)}}{\sqrt{2}}
```

Since  $r_2$  and r are clearly positive we can square each of them and compare the squared values for equality.

## In [3]: $r2^2 / Expand$ Out [3] $\frac{ab}{2} - \frac{ac}{2} - \frac{bc}{2} + \frac{c^2}{2}$

This is clearly the same as  $r^2$ .

 $In[4]: \mathbf{r} = (\mathbf{a} + \mathbf{b} - \mathbf{c}) / 2;$ Expand  $[\mathbf{r}^2] / \mathbf{b}^2 - \mathbf{c}^2 - \mathbf{a}^2 / \mathbf{c}^2$ Out [4]  $\frac{\mathbf{a} \mathbf{b}}{2} - \frac{\mathbf{a} \mathbf{c}}{2} - \frac{\mathbf{b} \mathbf{c}}{2} + \frac{\mathbf{c}^2}{2}$ 

## shoulder.nb

#### **Coordinates of Shoulder Point**

#### **Exploration**

Show that the coordinates of the shoulder point of a conic arc with control points  $P_0(x_0, y_0)$ ,  $P_A(x_A, y_A)$  and  $P_1(x_1, y_1)$  and projective discriminant  $\rho$  are given by

$$(x_M + \rho(x_A - x_M), y_M + \rho(y_A - y_M))$$

where  $P_M(x_M, y_M)$  is the midpoint of the conic arc's chord and has coordinates

$$x_M = \frac{x_0 + x_1}{2}$$
 and  $y_M = \frac{y_0 + y_1}{2}$ .

## Approach

Create the conic arc and construct the point at parameter t = 1/2.

## Solution

Create the conic arc.

```
In[1]: Clear[x0, y0, xA, yA, x1, y1, p];
cal = ConicArc2D[{x0, y0}, {xA, yA}, {x1, y1}, p];
```

Find the point at parameter t = 1/2.

```
In[2]: ptl = Point2D[cal[1/2]] // SimplifyOut[2] Point2D[\left\{\frac{1}{2} (x0 - px0 + x1 - px1 + 2pxA), \frac{1}{2} (y0 - py0 + y1 - py1 + 2pyA)\right\}]
```

This is the same as the specified point, when simplified.

```
In[3]: pt2 = Point2D[
    (x0 + x1) / 2 + p (xA - (x0 + x1) / 2),
    (y0 + y1) / 2 + p (yA - (y0 + y1) / 2)] // Simplify
Out[3] Point2D[{ 1/2 (x0 - p x0 + x1 - p x1 + 2 p xA), 1/2 (y0 - p y0 + y1 - p y1 + 2 p yA)}]
In[4]: SameQ[pt1, pt2]
Out[4] True
```

## stewart.nb

Stewart's Theorem

## **Exploration**



Show that for any  $\triangle ABC$  as shown in the figure the relationship between the lengths of the labeled line segments is given by

$$a^2m + b^2n = c(d^2 + mn).$$

## Approach

Without loss of generality, place the triangle in a convenient position and use the distance formula repeatedly to verify the relationship.

### Solution

Create points A, B, C and D in a convenient position.

```
In[1]: Clear[c, m, x, y];
    ptA = Point2D[{0, 0}];
    ptB = Point2D[{c, 0}];
    ptC = Point2D[{x, y}];
    ptD = Point2D[{m, 0}];
```

Compute the distances between the points.

```
In[2]: a = Distance2D[ptB, ptC];
b = Distance2D[ptA, ptC];
d = Distance2D[ptC, ptD];
```

Verify that the relationship is an identity.

```
In[3]: a^2*m+b^2*n-c*(d^2+m*n) /. n->c-m // Expand
Out[3] 0
```

# tancir1.nb

## Circle Tangent to Circle, Given Center

#### **Exploration**

Show that the radii of the two circles centered at  $(h_1, k_1)$  and tangent to the circle

$$(x - h_2)^2 + (y - k_2)^2 = r_2^2$$

are given by

$$r = |d \pm r_2|$$

where

$$d = \sqrt{(h_1 - h_2)^2 + (k_1 - k_2)^2}.$$

This formula is a special case of TangentCircles2D[ $\{pt \mid ln \mid cir\}$ , point].

## Approach

Fix the center point using the equations  $h = h_1$  and  $k = k_1$ . The circles are tangent if

$$(d - (r_2 - r))^2 (d - (r_2 + r))^2 = 0$$

where  $d = \sqrt{(h_1 - h_2)^2 + (k_1 - k_2)^2}$ . Solve the three equations for r.

## Solution

Solve the three equations.

Assuming d > 0 and  $r_2 > 0$ : (1)  $r = -d - r_2$  is always negative, hence invalid; (2)  $r = d - r_2$  is positive if  $d > r_2$ , i.e.  $(h_1, k_1)$  is outside circle  $c_2$ ; (3)  $r = -d + r_2$  is positive if  $d < r_2$ , i.e.  $(h_1, k_1)$  is inside circle  $c_2$ ; and (4)  $r = d + r_2$  is always positive and valid.

# tancir2.nb

### Circle Tangent to Circle, Center on Circle, Radius

### Exploration

Show that the centers (h, k) of the two circles passing through the point  $(x_1, y_1)$  with center on the circle  $x^2 + y^2 = 1$  and radius r = 1 are given by

$$(h,k) = \left(\frac{x_1}{2} \pm \frac{y_1\sqrt{4-d_1^2}}{2d_1}, \frac{y_1}{2} \mp \frac{x_1\sqrt{4-d_1^2}}{2d_1}\right)$$

where  $d_1 = \sqrt{x_1^2 + y_1^2}$ . This is a special case of TangentCircles2D[{*obj*},  $ln \mid cir, r$ ], where the object is a point.

## Approach

The radius is given, r = 1, so the center point (h, k) needs to be found. The equation  $(x_1 - h)^2 + (y_1 - k)^2 = 1$  is formed noting that the given point is on the circle. The equation  $h^2 + k^2 = 1$  is formed noting that the center is on this circle. Solve two equations in two unknowns.

## Solution

Solve the two equations.

```
\begin{split} & \text{In[1]: Clear[h, k, x1, y1, d1];} \\ & \text{ansl = Solve[{(x1 - h)^2 + (y1 - k)^2 = 1, h^2 + k^2 = 1}, \\ & \{h, k\}] \text{ //. } \{ \\ & \text{x1^2 + y1^2 -> d1^2} \text{ // FullSimplify} \end{split} \\ & \text{Out[1] } \{ \{h \rightarrow \frac{d1^4 - y1 \left(x1^2 y1 + y1^3 + \sqrt{-x1^2 \left(-4 + x1^2 + y1^2\right) \left(x1^2 + y1^2\right)}\right)}{2 d1^2 x1}, \\ & k \rightarrow \frac{x1^2 y1 + y1^3 + \sqrt{-x1^2 \left(-4 + x1^2 + y1^2\right) \left(x1^2 + y1^2\right)}}{2 d1^2} \}, \\ & \{h \rightarrow \frac{d1^4 - y1 \left(x1^2 y1 + y1^3 - \sqrt{-x1^2 \left(-4 + x1^2 + y1^2\right) \left(x1^2 + y1^2\right)}\right)}{2 d1^2 x1}, \\ & k \rightarrow \frac{x1^2 y1 + y1^3 - \sqrt{-x1^2 \left(-4 + x1^2 + y1^2\right) \left(x1^2 + y1^2\right)}}{2 d1^2 x1}, \\ & k \rightarrow \frac{x1^2 y1 + y1^3 - \sqrt{-x1^2 \left(-4 + x1^2 + y1^2\right) \left(x1^2 + y1^2\right)}}{2 d1^2} \} \} \end{split}
```

Simplify. Without loss of generality, assume all the point coordinates are positive.

```
\begin{split} & \ln[2]: \ \text{Clear[E1];} \\ & \text{ans2 = ans1 //. } \{ \\ & \text{x1^2*y1+y1^3 -> y1*d1^2,} \\ & \text{x1^2+y1^2 -> d1^2,} \\ & \text{sqrt[d1^2*E1_] -> d1*\text{sqrt[E1],} \\ & \text{sqrt[x1^2*E1_] -> x1*\text{sqrt[E1]} // Fullsimplify} \\ \\ & \text{Out[2]} \ \left\{ \left\{ h \to \frac{d1^3 - \sqrt{4 - d1^2} x1y1 - d1y1^2}{2 d1 x1}, \ k \to \frac{\sqrt{4 - d1^2} x1 + d1y1}{2 d1} \right\}, \\ & \left\{ h \to \frac{d1^3 + \sqrt{4 - d1^2} x1y1 - d1y1^2}{2 d1 x1}, \ k \to \frac{-\sqrt{4 - d1^2} x1 + d1y1}{2 d1} \right\} \right\} \\ & \text{In[3]: ans3 = ans2 //. } \{ \\ & \text{d1^3 - d1*y1^2 -> d1*(d1^2 - y1^2), \\ & \text{d1^2 - y1^2 -> x1^2} \} \\ & \text{Out[3]} \ \left\{ \left\{ h \to \frac{d1 x1^2 - \sqrt{4 - d1^2} x1y1}{2 d1 x1}, \ k \to \frac{\sqrt{4 - d1^2} x1 + d1y1}{2 d1} \right\}, \\ & \left\{ h \to \frac{d1 x1^2 - \sqrt{4 - d1^2} x1y1}{2 d1 x1}, \ k \to \frac{-\sqrt{4 - d1^2} x1 + d1y1}{2 d1} \right\} \right\} \\ & \text{In[4]: ans4 = Map[Apart, ans3, 3]} \end{split}
```

```
\begin{array}{l} \textit{Out}\,[4] & \Big\{ \Big\{ h \rightarrow \frac{x1}{2} - \frac{\sqrt{4-d1^2} \ y1}{2 \ d1} \ , \ k \rightarrow \frac{\sqrt{4-d1^2} \ x1}{2 \ d1} + \frac{y1}{2} \Big\} \,, \\ & \Big\{ h \rightarrow \frac{x1}{2} + \frac{\sqrt{4-d1^2} \ y1}{2 \ d1} \ , \ k \rightarrow - \frac{\sqrt{4-d1^2} \ x1}{2 \ d1} + \frac{y1}{2} \Big\} \Big\} \end{array}
```

# tancir3.nb

### Circle Tangent to Two Lines, Radius

## **Exploration**

Show that the centers (h, k) of the four circles tangent to the perpendicular lines

$$A_1x + B_1y = 0$$
 and  $-B_1x + A_1y = 0$ 

with radius r = 1 are given by

$$(A_1 - B_1, A_1 + B_1),$$
  
 $(A_1 + B_1, -A_1 + B_1),$   
 $(-A_1 - B_1, A_1 - B_1),$   
 $(-A_1 + B_1, -A_1 - B_1).$ 

Assume that the two lines are normalized,  $A_1^2 + B_1^2 = 1$ .

## Approach

A circle  $(x - h)^2 + (y - k)^2 = r^2$  tangent to a line Ax + By + C = 0 implies that

$$(A^2 + B^2) r^2 = (Ah + Bk + C)^2$$

giving two equations. The fixed radius r = 1 is a third equation. Solve three equations in three unknowns.

## Solution

Solve the three equations.

```
\begin{split} & \text{In}[1]: \ \text{Clear}[\textbf{r},\textbf{h},\textbf{k},\textbf{A1},\textbf{B1}]; \\ & \text{ansl} = \text{Solve}[\{\textbf{r}^2 = = (\ \textbf{A1} + \textbf{h} + \textbf{B1} + \textbf{k})^2, \\ & \textbf{r}^2 = = (-\textbf{B1} + \textbf{h} + \textbf{A1} + \textbf{k})^2, \\ & \textbf{r} = = 1\}, \\ & \textbf{(h, k, r)}] \\ & \text{Out}[1] \ \left\{ \left\{ \textbf{h} \to -\frac{-\textbf{A1} - \textbf{B1}}{\textbf{A1}^2 + \textbf{B1}^2}, \ \textbf{k} \to -\frac{\textbf{A1} - \textbf{B1}}{\textbf{A1}^2 + \textbf{B1}^2}, \ \textbf{r} \to 1 \right\}, \ \left\{ \textbf{h} \to -\frac{\textbf{A1} - \textbf{B1}}{\textbf{A1}^2 + \textbf{B1}^2}, \ \textbf{k} \to -\frac{\textbf{A1} + \textbf{B1}}{\textbf{A1}^2 + \textbf{B1}^2}, \ \textbf{r} \to 1 \right\}, \\ & \left\{ \textbf{h} \to -\frac{-\textbf{A1} + \textbf{B1}}{\textbf{A1}^2 + \textbf{B1}^2}, \ \textbf{k} \to -\frac{-\textbf{A1} - \textbf{B1}}{\textbf{A1}^2 + \textbf{B1}^2}, \ \textbf{r} \to 1 \right\}, \ \left\{ \textbf{h} \to -\frac{\textbf{A1} + \textbf{B1}}{\textbf{A1}^2 + \textbf{B1}^2}, \ \textbf{k} \to -\frac{-\textbf{A1} + \textbf{B1}}{\textbf{A1}^2 + \textbf{B1}^2}, \ \textbf{r} \to 1 \right\}, \end{split}
```

#### Simplify.

In[2]: ans2 = ans1 //. A1^2 + B1^2 -> 1

```
 \begin{array}{ll} \textit{Out[2]} & \left\{ \left\{ h \rightarrow \texttt{A1} + \texttt{B1}, \ k \rightarrow -\texttt{A1} + \texttt{B1}, \ r \rightarrow 1 \right\}, \ \left\{ h \rightarrow -\texttt{A1} + \texttt{B1}, \ k \rightarrow -\texttt{A1} - \texttt{B1}, \ r \rightarrow 1 \right\}, \\ & \left\{ h \rightarrow \texttt{A1} - \texttt{B1}, \ k \rightarrow \texttt{A1} + \texttt{B1}, \ r \rightarrow 1 \right\}, \ \left\{ h \rightarrow -\texttt{A1} - \texttt{B1}, \ k \rightarrow \texttt{A1} - \texttt{B1}, \ r \rightarrow 1 \right\} \right\} \end{array}
```

# tancir4.nb

### Circle Through Two Points, Center on Circle

### **Exploration**

Show that the radii of the two circles passing through the points (0, a) and (0, -a) with centers on the circle  $x^2 + y^2 = r_2^2$  are both given by

$$r = \sqrt{a^2 + r_2^2}.$$

This is a special case of TangentCircles2D[ $\{obj_1, obj_2\}$ , *line* | *circle*] where the two objects are points.

## Approach

Two equations can be formed using the fact that points (0, a) and (0, -a) are on the circle. A third equation can be formed since the center is on a given circle. Solve three equations in three unknowns.

## Solution

Solve three equations in three unknowns. The solutions with negative radii are invalid and discarded.

$$\begin{split} & \text{In[1]: Clear[h, k, r];} \\ & \text{ans1 = Solve[{ (0 - h) ^2 + (a - k) ^2 == r^2, \\ & (0 - h) ^2 + (-a - k) ^2 == r^2, \\ & h^2 + k^2 == r2^2 2 \}, \\ & \text{ { } } h, k, r \}] \\ & \text{Out[1] } \left\{ \left\{ r \to -\sqrt{c^2 + r2^2 - 2 c x + x^2 + y^2}, \ h \to -r2, \ k \to 0 \right\}, \\ & \left\{ r \to \sqrt{c^2 + r2^2 - 2 c x + x^2 + y^2}, \ h \to r2, \ k \to 0 \right\}, \\ & \left\{ r \to \sqrt{c^2 + r2^2 - 2 c x + x^2 + y^2}, \ h \to -r2, \ k \to 0 \right\}, \\ & \left\{ r \to \sqrt{c^2 + r2^2 - 2 c x + x^2 + y^2}, \ h \to -r2, \ k \to 0 \right\}, \\ & \left\{ r \to \sqrt{c^2 + r2^2 - 2 c x + x^2 + y^2}, \ h \to r2, \ k \to 0 \right\}, \end{split}$$

## tancir5.nb

#### Circle Tangent to Three Lines

#### **Exploration**

Show that the radii of the four circles tangent to the lines x = 0, y = 0 and Ax + By + C = 0, are given by

$$r = \left| \frac{C}{1 \pm A \pm C} \right|$$

taking all four combinations of signs and assuming that the lines are normalized. This is a special case of TangentCircles2D[{ $obj_1$ ,  $obj_2$ ,  $obj_3$ }] where all three of the objects are lines.

## Approach

A line ax + by + c = 0 is tangent to a circle  $(x - h)^2 + (y - k)^2 = r^2$  if the equation

$$(a^2 + b^2) r^2 = (ah + bk + c)^2$$

holds. Form three equations in three unknowns from this equation and solve.

## Solution

Solve three equations in three unknowns.

Extract the value of r.

In[2]: ans2 = Map[(r /. #)&, ans1]  $Out[2] \left\{ -\frac{C1}{-1+A1-B1}, \frac{C1}{-1+A1-B1}, -\frac{C1}{1+A1-B1}, \frac{C1}{1+A1-B1}, -\frac{C1}{-1+A1+B1}, \frac{C1}{-1+A1+B1}, -\frac{C1}{-1+A1+B1}, -\frac{C1}{-1+A1+B1+B1}, -\frac{C1}{-1+A1+B1+B1}, -\frac{C1}{-1+A1+B1+$ 

Put all the negative signs in the denominator.

$$\begin{array}{l} In \left\{3\right\}: \mbox{ Clear[E1];} \\ \mbox{ ans3 = ans2 //. Times[-1, Power[E1_, -1], C1] :>} \\ \mbox{ Times[Power[Expand[-E1], -1], C1]} \\ Out \left\{3\right\} \left\{ \begin{array}{c} C1 \\ \hline 1-A1+B1 \end{array}, \begin{array}{c} C1 \\ \hline -1+A1-B1 \end{array}, \begin{array}{c} C1 \\ \hline -1-A1-B1 \end{array}, \begin{array}{c} C1 \\ \hline 1+A1-B1 \end{array}, \begin{array}{c} C1 \\ \hline 1+A1-B1 \end{array}, \begin{array}{c} C1 \\ \hline 1+A1-B1 \end{array}, \end{array} \right. \end{array} \right.$$

Change all the minus signs to positive.

```
In[4]: ans4 = ans3 //. Times[Power[Plus[-1, El_], -1], Cl] :> 
Times[Power[Plus[1 + El], -1], -Cl] 
Out[4] { Cl 
 - Al - Bl 
 - Cl 
 - Cl 
 - Al - Bl 
 - Cl 
 - Al - Bl 
 - Cl 
 - Al - Bl 
 - Cl 
 - Al - Bl 
 - Cl 
 - Cl
```

Take the absolute value and return only the unique terms.

```
In[5]: Union[Abs[ans4]]
```

 $\textit{Out[5]} \hspace{0.1in} \left\{ \texttt{Abs} \left[ \hspace{0.1in} \frac{\texttt{Cl}}{\texttt{1}-\texttt{Al}-\texttt{Bl}} \right], \hspace{0.1in} \texttt{Abs} \left[ \hspace{0.1in} \frac{\texttt{Cl}}{\texttt{1}+\texttt{Al}-\texttt{Bl}} \right], \hspace{0.1in} \texttt{Abs} \left[ \hspace{0.1in} \frac{\texttt{Cl}}{\texttt{1}-\texttt{Al}+\texttt{Bl}} \right], \hspace{0.1in} \texttt{Abs} \left[ \hspace{0.1in} \frac{\texttt{Cl}}{\texttt{1}+\texttt{Al}+\texttt{Bl}} \right] \right\}$ 

## tancirpt.nb

#### Tangency Point on a Circle

#### **Exploration**

Show that if a line Ax + By + C = 0 is tangent to a circle  $(x - h)^2 + (y - k)^2 = r^2$  then the coordinates of the point of tangency are

$$\left(h - \frac{Ar^2}{Ah + Bk + C}, k - \frac{Br^2}{Ah + Bk + C}\right).$$

## Approach

The pole (point) of the line is the point of tangency.

## Solution

Create the line, circle and pole point. This result was computed using *Mathematica* Version 3.0.1. Version 4.0 computes a different result that is algebraically equivalent. Both versions produce the same final step.

Simplify to the desired form.

```
\label{eq:linear} \begin{split} & \text{In[2]: } \textbf{Map[Apart, p1]} \\ & \text{Out[2] Point2D} \Big[ \big\{ h - \frac{\text{Al}\,r^2}{\text{Cl} + \text{Al}\,h + \text{Bl}\,k} \,, \, k - \frac{\text{Bl}\,r^2}{\text{Cl} + \text{Al}\,h + \text{Bl}\,k} \big\} \Big] \end{split}
```

## tetra.nb

#### Area of a Tetrahedron's Base

## **Exploration**

A tetrahedron is a three-dimensional geometric object bounded by four triangular faces. Given a tetrahedron with vertices O(0,0,0), A(a,0,0), B(0,b,0) and C(0,0,c) show that the areas of the triangular faces are related by the equation

$$(A_{ABC})^{2} = (A_{AOB})^{2} + (A_{AOC})^{2} + (A_{BOC})^{2}$$

where  $A_{xyz}$  is the area of the triangle whose vertices are x, y and z. Note the similarity to the Pythagorean Theorem for right triangles.

#### Approach

Compute the area of  $\triangle ABC$  using Heron's formula and compare it to the areas of the other triangles.

#### Solution

Compute the semi-perimeter, s, of  $\triangle ABC$ .

```
In[1]: Clear[AB, AC, BC];
    s = (AB + AC + BC) / 2
Out[1]    1/2 (AB + AC + BC)
```

Compute the areas of  $\triangle ABC$  using Heron's formula. Replace the lengths of each side by expressions in a, b and c, the coordinates on the axes.

```
In[2]: Clear[a, b, c];
A1 = Expand[s (s - AB) (s - AC) (s - BC)] //. {
AB^2 -> a^2 + b^2, AB^4 -> (a^2 + b^2)^2, AC^2 -> a^2 + c^2, AC^4 -> (a^2 + c^2)^2, BC^2 -> b^2 + c^2, BC^4 -> (b^2 + c^2)^2, BC^2 -> b^2 + c^2, BC^4 -> (b^2 + c^2)^2 + \frac{1}{16} (a^2 + b^2)^2 + \frac{1}{8} (a^2 + b^2) (a^2 + c^2) - \frac{1}{16} (a^2 + c^2)^2 + \frac{1}{8} (a^2 + b^2) + \frac{1}{8} (a^2 + c^2) (b^2 + c^2) - \frac{1}{16} (b^2 + c^2)^2
```

Replace certain expressions with the areas of the triangles involved.

# tncirtri.nb

#### Circles Tangent to an Isosceles Triangle

#### **Exploration**

A circle is inscribed in an isosceles triangle with sides a, a and 2b in length. A second, smaller circle is inscribed tangent to the first circle and to the equal sides of the triangle. Show that the radius of the second circle is

$$r = b\sqrt{\frac{(a-b)^3}{(a+b)^3}}$$

Assume that a > b.

## Approach

Construct an isosceles triangle whose sides are the given lengths. Construct the circle inscribed in the triangle. The point of tangency between the first and second circle is at the parameter  $\theta = \pi/2$  on the first circle. Construct a second triangle from the equal-length sides and a line tangent to the first circle at the tangency point. The second circle can then be inscribed inside the second triangle. Find and simplify the radius of the second inscribed circle.

### Solution

Construct the isosceles triangle with the proper side lengths.

```
In[1]: Clear[a, b];
T1 = Triangle2D[{a, a, 2b}] // FullSimplify
Out[1] Triangle2D[{0, 0}, {2b, 0}, {b, \sqrt{(a-b) (a+b)}}]
```

Construct the first inscribed circle and simplify the result.

```
In[2]: Clear[E1, E2];
C1 = (Circle2D[T1, Inscribed2D] // FullSimplify) //.
\{Sqrt[a^2] \rightarrow a, Sqrt[b^2] \rightarrow b,
Sqrt[E1\_*b^2/E2\_] \rightarrow b*Sqrt[E1/E2]\} // FullSimplify
Out[2] Circle2D[\{b, \frac{b\sqrt{a^2-b^2}}{a+b}\}, b\sqrt{-1 + \frac{2a}{a+b}}]
```

Construct the point of tangency between the first circle and the second.

```
In[3]: P1 = Point2D[C1[Pi/2]] // FullSimplifyOut[3] Point2D[\{b, b\left(\frac{\sqrt{(a-b)}(a+b)}{a+b} + \sqrt{-1 + \frac{2a}{a+b}}\right)\}]
```

Construct the second triangle. The results are complicated, so we define and use some simplification rules that are applied to the result.

```
In[4]: rules1 = {
    -1+2a/(a+b) -> (a-b)/(a+b),
    Sqrt[(a-b) (a+b)]/(a+b) -> Sqrt[(a-b)]/Sqrt[(a+b)],
    Sqrt[(a-b)/(a+b)] -> Sqrt[a-b]/Sqrt[a+b],
    1/Sqrt[a^2-b^2] -> 1/(Sqrt[a-b]*Sqrt[a+b]),
    Sqrt[a^2-b^2] -> Sqrt[a-b]*Sqrt[a+b];
    L1 = Line2D[Segment2D[T1, 2, 3]] //FullSimplify;
    L2 = Line2D[Segment2D[T1, 1, 3]] //FullSimplify;
    T2 = Triangle2D[L1, L2, Line2D[P1, 0]];
    T2 = (T2 //. rules1 // Simplify) //. rules1
Out[4] Triangle2D[{b, \sqrta-b \sqrta+b}}, {2ab \sqrta-b b \sqrta-b b \sqrta-b s \s
```

Construct the circle inscribed in the second triangle and find the radius. The results are complicated, so we define and use some simplification rules that are applied to the result.

```
In[5]: rules2 = \{ Sqrt[a^2*(a-b)^2/(a+b)^2] \rightarrow a (a-b) / (a+b), \\Sqrt[(a-b)^2*b^2/(a+b)^2] \rightarrow b (a-b) / (a+b), \\Sqrt[(a-b)^3*b^2/(a+b)^3] \rightarrow b*Sqrt[(a-b)^3/(a+b)^3]\}; \\R2 = ((Radius2D[C2 = Circle2D[T2, Inscribed2D]] // Simplify) //. \\rules2 // Simplify) //. rules2 \\Out[5] b \sqrt{\frac{(a-b)^3}{(a+b)^3}}
```

Mathematica Version 3.0.1 produces the desired result in the previous step. Version 4.0 needs the following additional step to produce the desired result. This step doesn't change the expression generated by Version 3.0.1.

```
In [6]: R2 //. {

Sqrt[E1_*b^2/(E2_)^3] -> b*Sqrt[E1/E2^3],

Sqrt[E1_] :> Sqrt[Factor[E1]]}

Out[6] b\sqrt{\frac{(a-b)^3}{(a+b)^3}}
```

## Discussion

This is the plot of a numerical example with a = 10 and b = 3.5.

```
In[7]: Sketch2D[{T1, C1, P1, T2, C2} /. {a -> 10, b -> 3.5}];
```



This is another example with a = 10 and b = 6.

In[8]: Sketch2D[{T1, C1, P1, T2, C2} /. {a -> 10, b -> 6}];



# tnlncir.nb

## Construction of Two Related Circles

#### **Exploration**

Prove that if OP and OQ are the tangent lines from (0,0) to the circle

$$x^2 + y^2 + 2gx + 2fy + c = 0$$

then the equation of the circle OPQ is

$$x^2 + y^2 + gx + fy = 0.$$

## Approach

Create the circle from the given quadratic and construct the polar (line) of the origin with respect to the circle. Intersect the polar with the circle to find P and Q. Construct a circle through O, P and Q and find its equation.

### Solution

Create the origin point and the circle from the given equation.

```
In[1]: Clear[g, f, c];
    P0 = Point2D[0, 0];
    C1 = Circle2D[Quadratic2D[1, 0, 1, 2g, 2f, c]] // Simplify
Out[1] Circle2D[{-g, -f}, \sqrt{-c+f^2+g^2}]
```

Construct the polar line.

```
In[2]: L1 = Line2D[P0, C1] // Simplify
Out[2] Line2D[g, f, c]
```

Find the intersection points.

In[3]: pts = Points2D[L1, C1] // FullSimplify

$$\begin{aligned} & \text{Out[3]} \; \left\{ \text{Point2D} \Big[ \Big\{ -\frac{c \; g}{f^2 + g^2} - \frac{f \; \sqrt{c - \frac{c^2}{f^2 + g^2}}}{\sqrt{f^2 + g^2}} \; , \; -\frac{c \; f}{f^2 + g^2} + \frac{g \; \sqrt{c - \frac{c^2}{f^2 + g^2}}}{\sqrt{f^2 + g^2}} \Big\} \Big] \; , \\ & \text{Point2D} \Big[ \Big\{ -\frac{c \; g}{f^2 + g^2} + \frac{f \; \sqrt{c - \frac{c^2}{f^2 + g^2}}}{\sqrt{f^2 + g^2}} \; , \; -\frac{c \; f}{f^2 + g^2} - \frac{g \; \sqrt{c - \frac{c^2}{f^2 + g^2}}}{\sqrt{f^2 + g^2}} \Big\} \Big] \Big\} \end{aligned}$$

Construct the circle through the three points.

In[4]: C2 = Circle2D[P0, pts[[1]], pts[[2]]] // FullSimplify $Out[4] Circle2D\left[\left\{-\frac{g}{2}, -\frac{f}{2}\right\}, \frac{1}{2}\sqrt{f^2 + g^2}\right]$ 

Convert the circle to an equation.

```
In[5]: Clear[x, y];
Equation2D[Quadratic2D[C2] // Simplify, {x, y}]
Out[5] g x + x<sup>2</sup> + f y + y<sup>2</sup> == 0
```

### Discussion

Construct the circle related to  $x^2 + y^2 - 6x - 4y + 12 = 0$ .

```
In[6]: P0 = Point2D[0, 0];
C1 = Circle2D[Quadratic2D[x<sup>2</sup> + y<sup>2</sup> - 6 x - 4 y + 12 == 0, {x, y}]];
L1 = TangentLines2D[P0, C1];
P1 = Point2D[First[L1], C1];
P2 = Point2D[Last[L1], C1];
C2 = Circle2D[Quadratic2D[x<sup>2</sup> + y<sup>2</sup> - 3 x - 2 * y == 0, {x, y}]];
Sketch2D[{P0, C1, L1, P1, P2, C2}];
```



## triallen.nb

### Triangle Altitude Length

#### **Exploration**

Show that the length, L, of a triangle's altitude (from vertex  $V_3$  to side  $s_1$ ) is given by

$$L^{2} = s_{2}s_{3}\left(1 - \frac{s_{1}^{2}}{(s_{2} + s_{3})^{2}}\right)$$

where  $s_1$ ,  $s_2$  and  $s_3$  are the lengths of the triangle's sides.

### Approach

Construct a triangle in a convenient, yet sufficiently general position. Then construct the triangle's altitude. Show that the length of the altitude is given by the expression. Since the length of each triangle side,  $s_n$ , is positive,  $\sqrt{s_n^2} = s_n$ .

## Solution

Construct a triangle with sides  $s_1$ ,  $s_2$  and  $s_3$ . By default, the triangle's first vertex is located at the origin.

The length of the altitude is the distance from the triangle's third vertex to the x-axis. This result was computed using *Mathematica* Version 3.0.1. Version 4.0 computes a slightly different result that is algebraically equivalent. 810
# trialt.nb

## Altitude of a Triangle

### **Exploration**

The *altitude* from vertex A of  $\triangle ABC$  is a line segment from A perpendicular to side BC (or the extension of BC). Show that the equation of the line containing the altitude from A is

$$(x_3 - x_2) x + (y_3 - y_2) y - x_1(x_3 - x_2) - y_1(y_3 - y_2) = 0$$

where the coordinates of the vertices are  $A(x_1, y_1)$ ,  $B(x_2, y_2)$  and  $C(x_3, y_3)$ .

## Approach

Construct the altitude and show that the line containing it is the line given.

## Solution

Construct the line BC.

```
In[1]: Clear[x2, y2, x3, y3];
BC = Line2D[{x2, y2}, {x3, y3}];
```

Construct the altitude from A perpendicular to BC.

```
In[2]: Clear[x1, y1];
    alt = Line2D[Point2D[x1, y1], BC]
Out[2] Line2D[-x2+x3, -y2+y3, -x1 (-x2+x3) +y1 (y2-y3)]
```

Convert the line to an equation.

```
In[3]: Clear[x, y];
    Equation2D[alt, {x, y}]
Out[3] x (-x2+x3) -x1 (-x2+x3) +y1 (y2-y3) +y (-y2+y3) == 0
```

### Discussion

This defines a new function that constructs all the lines underlying the altitudes of a triangle.

```
In[4]: Altitudes2D[Triangle2D[{x1_, y1_}, {x2_, y2_}, {x3_, y3_}]] :=
        {Altitude$2D[{x1, y1}, {x2, y2}, {x3, y3}],
        Altitude$2D[{x2, y2}, {x3, y3}, {x1, y1}],
        Altitude$2D[{x3, y3}, {x1, y1}, {x2, y2}]};
        Altitude$2D[{x1_, y1_}, {x2_, y2_}], {x3_, y3_}] :=
        Line2D[x3 - x2, y3 - y2, -x1 (x3 - x2) - y1 (y3 - y2)];
```

This is the plot of a numerical example.

```
In[5]: T1 = Triangle2D[{-1, -2}, {-2, 3}, {4, 0}];
    Sketch2D[{T1, Altitudes2D[T1],
        Map[Point2D, List @@ T1]}];
```



## triarea.nb

## Area of Triangle Configurations

#### **Exploration**



For the triangle illustrated in the figure, show that the area,  $A_1$ , associated with the AAS (angle-angle-side) configuration whose parameters are  $a_1$ ,  $a_2$  and  $s_1$  is given by

$$A_1 = \frac{s_1^2 \sin(a_2) \sin(a_1 + a_2)}{2 \sin(a_1)}$$

Show that the area,  $A_2$ , associated with the ASA (angle-side-angle) configuration whose parameters are  $a_1$ ,  $s_3$  and  $a_2$  is given by

$$A_2 = \frac{s_3^2 \sin(a_1) \sin(a_2)}{2 \sin(a_1 + a_2)}$$

Show that the area,  $A_3$ , associated with the SAS (side-angle-side) configuration whose parameters are  $s_1$ ,  $a_2$  and  $s_3$  is given by

$$A_3 = \frac{s_1 s_3 \sin(a_2)}{2}.$$

## Approach

Construct the triangle associated with each configuration, and then compute the expression for the area and simplify.

#### Solution

Construct the triangle for the AAS configuration.

```
In[1]: Clear[a1, a2, a3, s1, s2, s3];
T1 = (Triangle2D[{{s1, Null, Null}, {a1, a2, Null}}] // FullSimplify) //.
Sqrt[s1^2 * Sin[a2]^2] -> s1 * Sin[a2]
Out[1] Triangle2D[{0, 0}, {s1 Csc[a1] Sin[a1 + a2], 0},
{s1 Cot[a1] Sin[a2], s1 Sin[a2]}]
```

Compute the area for the AAS configuration and simplify. The sine function is introduced to prevent simplification back to the cosecant form.

```
In[2]: A1 = Area2D[T1] //. Csc[a1] -> 1/sine[a1]
Out[2] \frac{s1^{2} Sin[a2] Sin[a1 + a2]}{2 sine[a1]}
```

Construct the triangle for the ASA configuration.

```
In[3]: Clear[a1, a2, a3, s1, s2, s3];
T2 = (Triangle2D[{{Null, Null, s3}, {a1, a2, Null}}] // FullSimplify) //.
Sqrt[s3^2*Csc[a1+a2]^2*Sin[a1]^2*Sin[a2]^2] ->
s3*Csc[a1+a2]*Sin[a1]*Sin[s2]
Out[3] Triangle2D[{0, 0}, {s3, 0},
{s3 Cos[a1] Csc[a1+a2] Sin[a2], s3 Csc[a1+a2] Sin[a1] Sin[s2]}]
```

Compute the area for the ASA configuration and simplify. The sine function is transformed to lower case to prevent simplification back to the cosecant form.

Construct the triangle for the SAS configuration.

```
In[5]: Clear[a1, a2, a3, s1, s2, s3];
T3 = (Triangle2D[{{s1, Null, s3}, {Null, a2, Null}}] // FullSimplify) //.
Sqrt[a1^2 + Sin[a2]^2] -> s1 + Sin[a2]
Out[5] Triangle2D[{0, 0}, {s3, 0}, {s3 - s1 Cos[a2], \sqrt{s1^2 Sin[a2]^2}}]
```

Compute the area for the SAS configuration and simplify.

```
In[6]: A3 = Area2D[T3] //. Sqrt[s1^2*Sin[a2]^2] -> s1*Sin[a2]
Out[6] 1/2 s1 s3 Sin[a2]
```

# triarIns.nb

## Area of Triangle Bounded by Lines

## **Exploration**

Show that the area of the triangle bounded by the lines

$$y = m_1 x + c_1$$
,  $y = m_2 x + c_2$  and  $x = 0$ 

is given by

$$A = \frac{(c_1 - c_2)^2}{2\sqrt{(m_1 - m_2)^2}}.$$

## Approach

Create the triangle and compute the area.

## Solution

Create the triangle.

Get the vertex points of the triangle.

```
In[2]: \{p1, p2, p3\} = Map[Point2D[t1, #]\&, \{1, 2, 3\}]Out[2] \{Point2D[\{\frac{c1-c2}{-m1+m2}, \frac{-c2 m1+c1 m2}{-m1+m2}\}], Point2D[\{0, c1\}], Point2D[\{0, c2\}]\}
```

Compute the area of the triangle using Heron's formula.

```
In[3]: Clear[s];
    a = Distance2D[p1, p2];
    b = Distance2D[p2, p3];
    c = Distance2D[p3, p1];
    s = (a + b + c) / 2;
    Al = Sqrt[s (s - a) (s - b) (s - c)] // FullSimplify
Out[3] \frac{1}{2} \sqrt{\frac{(c1 - c2)^4}{(m1 - m2)^2}}
```

Since  $(c_1 - c_2)^2$  is positive, the formula simplifies to the desired result.

```
In[4]: A1 /. Sqrt[E1_^4/E2_^2] ->
E1^2/Sqrt[E2^2]
Out[4] (c1-c2)<sup>2</sup>
2 \sqrt[m1-m2)<sup>2</sup>
```

# tricent.nb

## Centroid of a Triangle

## Exploration



Show that the centroid of a triangle, as illustrated in the figure, is on a line at a distance  $\overline{y} = b/3$  from the base of the triangle.

## Approach

Place the triangle in a convenient position as shown in the figure. Create equations for the moments of infinitesimal areas on either side of the centroid line. Use integration to find the ordinate of the line.

## Solution

Create lines for two sides of the triangle whose vertices are (0,0), (d,0) and (a,b).

```
In[1]: Clear[a, b, d];
L1 = Line2D[{a, b}, {0, 0}];
L2 = Line2D[{a, b}, {d, 0}];
```

Construct a horizontal line at a general coordinate, y, which is the height of the centroid.

```
In[2]: Clear[y];
L = Line2D[Point2D[0, y], 0];
```

The width of the triangle, W, is the difference between the abscissa of the intersection points of the sides and the horizontal line.

The moments of inertia on each side of the horizontal line must be equal.

```
 \begin{array}{l} In\left[4\right]: \mbox{ Clear[yB];} \\ & \mbox{ Solve[Integrate[W \star (yB - y), \{y, 0, yB\}] ==} \\ & \mbox{ Integrate[W \star (y - yB), \{y, yB, b\}], yB]} \\ \\ Out\left[4\right] \ \left\{\left\{yB \rightarrow \frac{b}{3}\right\}\right\} \end{array}
```

#### Discussion

Notice that the centroid only depends on the height of the triangle, b. The centroid's height does not depend on the horizontal location of the apex point, a, nor on the width of the base, d.

## tricev.nb

## Triangle Cevian Lengths

## **Exploration**

Prove that the length of the altitude,  $h_1$ , from vertex  $V_1$  of a triangle to the opposite side of the triangle (whose length is  $s_1$ ) is given by

$$h_1 = \frac{\sqrt{PS}}{2s_1}$$

where  $S = s_1 + s_2 + s_3$ ,  $P = (-s_1 + s_2 + s_3)(s_1 - s_2 + s_3)(s_1 + s_2 - s_3)$  and  $s_1$ ,  $s_2$  and  $s_3$  are the lengths of the triangle's sides. Prove that the length of the median,  $m_1$ , from vertex  $V_1$  is given by

$$m_1 = \frac{1}{2}\sqrt{-s_1^2 + 2\left(s_2^2 + s_3^2\right)}.$$

Prove that the length of the angle bisector,  $b_1$ , from vertex  $V_1$  is given by

$$b_1 = \frac{\sqrt{Ss_2s_3(-s_1+s_2+s_3)}}{s_2+s_3}.$$

Also show that the formulas for the lengths of the cevians from vertices  $V_2$  and  $V_3$  can be found by cyclic permutation of the subscripts given in the formulas above.

## Approach

Construct a triangle with the given side lengths. Construct the associated cevians (altitude, median and angle bisector) and compute and simplify the expressions for their lengths.

#### Solution

Construct a triangle with the given side lengths and simplify.

```
In[1]: Clear[s, s1, s2, s3, S, P, e1];
T = (Triangle2D[{s1, s2, s3}] // FullSimplify) /.
Sqrt[-e1_/s_Symbol^2] -> Sqrt[-e1]/s
Out[1] Triangle2D[{0, 0}, {s3, 0},
{ (-s1<sup>2</sup> + s2<sup>2</sup> + s3<sup>2</sup> / 2 s3 , (-s2 - s3) (s1 - s2 + s3) (s1 + s2 + s3) / 2 s3 }]
```

The *altitude* of a triangle is the cevian perpendicular to the opposite side. These functions return the length of the altitude (the *height*) for each vertex, 1, 2 or 3. The perpendicular is found by projecting the vertex on the line containing the opposite side.

Compute the length of each altitude (the height) using the functions defined above and simplify.

```
In[4]: (Map[Height2D[T, #]&, \{1, 2, 3\}] // FullSimplify) //. 
 {Sqrt[-e1_/s_Symbol^2] -> Sqrt[-e1]/s, 
 Sqrt[e1_/s_Symbol^2] -> Sqrt[e1]/s, 
 s1 + s2 + s3 -> S, 
 (s1 - s2 - s3) (s1 + s2 - s3) (s1 - s2 + s3) -> -P, 
 (-s1 + s2 - s3) (s1 + s2 - s3) (-s1 + s2 + s3) -> -P, 
 (s1 + s2 - s3) (s1 - s2 + s3) (-s1 + s2 + s3) -> P} 
Out[4] {<math>\frac{\sqrt{PS}}{2s1}, \frac{\sqrt{PS}}{2s2}, \frac{\sqrt{PS}}{2s3}}
```

The *median* of a triangle is the cevian connecting a vertex to the midpoint of the opposite side. These functions return the length of the median for each vertex, 1, 2 or 3.

Compute the length of each median using the functions defined above and simplify.

In[7]: Map[Median2D[T, #]&, {1, 2, 3}] // FullSimplify

 $\textit{Out[7]} \quad \left\{ \frac{1}{2} \, \sqrt{-\texttt{s1}^2 + 2 \, (\texttt{s2}^2 + \texttt{s3}^2)} \right. , \ \frac{1}{2} \, \sqrt{2 \, \texttt{s1}^2 - \texttt{s2}^2 + 2 \, \texttt{s3}^2} \, , \ \frac{1}{2} \, \sqrt{2 \, (\texttt{s1}^2 + \texttt{s2}^2) - \texttt{s3}^2} \, \right\}$ 

The *angle bisector* of a triangle is the cevian bisecting the angle of the vertex. These functions return the length of the angle bisector for each vertex, 1, 2 or 3. Note that the angle bisector must pass through the center of the inscribed circle.

Compute the length of each angle bisector using the functions defined above and simplify.

```
In[10]: (((Map[Bisector2D[T, #]&, {1, 2, 3}] // FullSimplify) //.
        {Sqrt[s_Symbol^2] -> s} // FullSimplify) /.
        (1 - s3^2/(s1 + s2)^2) :> Factor[(s1 + s2)^2 - s3^2] / (s1 + s2)^2) //.
        {Sqrt[e1_/(s1_Symbol + s2_Symbol)^2] :> Sqrt[e1] / (s1 + s2),
        (s1 + s2 + s3) -> S}
```

```
\textit{Out[10]} \; \left\{ \frac{\sqrt{S\,s2\,s3\,(-s1+s2+s3)}}{s2+s3} \; , \; \frac{\sqrt{S\,s1\,s3\,(s1-s2+s3)}}{s1+s3} \; , \; \frac{\sqrt{S\,s1\,s2\,(s1+s2-s3)}}{s1+s2} \right\}
```

## triconn.nb

### **Concurrent Triangle Altitudes**

#### **Exploration**

Show that the three altitudes of any  $\triangle ABC$  are concurrent in a single point (x, y) whose coordinates are given by

$$x = \frac{x'}{D}$$
 and  $y = \frac{y'}{D}$ 

where

$$\begin{aligned} x' &= -(y_1 - y_2) \left( x_1 x_2 + y_3^2 \right) + (y_1 - y_3) \left( x_1 x_3 + y_2^2 \right) - (y_2 - y_3) \left( x_2 x_3 + y_1^2 \right) \\ y' &= +(x_1 - x_2) \left( y_1 y_2 + x_3^2 \right) - (x_1 - x_3) \left( y_1 y_3 + x_2^2 \right) + (x_2 - x_3) \left( y_2 y_3 + x_1^2 \right) \end{aligned}$$

and

$$D = \left| \begin{array}{ccc} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{array} \right|$$

and the coordinates of the vertices are  $A(x_1, y_1)$ ,  $B(x_2, y_2)$  and  $C(x_3, y_3)$ . This point is called the *orthocenter* of the triangle.

### Approach

From the exploration trialt.nb the *altitude* from vertex A of  $\triangle ABC$  to side BC (or the extension of BC) is the line

$$(x_3 - x_2) x + (y_3 - y_2) y - x_1(x_3 - x_2) - y_1(y_3 - y_2) = 0$$

where the coordinates of the vertices are  $A(x_1, y_1)$ ,  $B(x_2, y_2)$  and  $C(x_3, y_3)$ . Use this formula to show that the three altitudes are concurrent and intersect one pair of the altitudes to find the coordinates of the orthocenter.

#### Solution

Define a function for constructing the altitude line of a triangle (the altitude from the first vertex).

```
In[1]: Altitude2D[Triangle2D[{x1_, y1_}, {x2_, y2_}, {x3_, y3_}]] :=
Line2D[x3 - x2, y3 - y2, -x1 * (x3 - x2) - y1 * (y3 - y2)]
```

Define a function for constructing the orthocenter.

```
In[2]: Orthocenter2D[Triangle2D[{x1_, y1_}, {x2_, y2_}, {x3_, y3_}]] :=
Module[{X1, Y1, D1},
Point2D[X1/D1, Y1/D1] //. {
X1 -> - (y1 - y2) (x1 * x2 + y3^2)
+ (y1 - y3) (x1 * x3 + y2^2)
- (y2 - y3) (x2 * x3 + y1^2),
Y1 -> (x1 - x2) (y1 * y2 + x3^2)
- (x1 - x3) (y1 * y3 + x2^2)
+ (x2 - x3) (y2 * y3 + x1^2),
D1 -> Det[{x1, y1, 1}, {x2, y2, 1}, {x3, y3, 1}]];
```

Create the three vertex points.

```
In[3]: Clear[x1, y1, x2, y2, x3, y3];
A1 = Point2D[x1, y1];
B1 = Point2D[x2, y2];
C1 = Point2D[x3, y3];
```

Construct the three altitudes.

```
In[4]: lnA = Altitude2D[Triangle2D[A1, B1, C1]];
lnB = Altitude2D[Triangle2D[B1, C1, A1]];
lnC = Altitude2D[Triangle2D[C1, A1, B1]];
```

Show that they are concurrent by showing that the determinant of their coefficients is zero.

```
In[5]: Det[{List @@ lnA,
      List @@ lnB,
      List @@ lnC}]
Out[5] 0
```

Find the coordinates of the orthocenter by intersecting a pair of altitudes.

```
In[6]: p1 = Point2D[lnA, lnB]
```

```
\begin{array}{l} \textit{Out[6] Point2D} \Big[ & \\ & \left\{ \frac{(-x2\ (x1-x3)\ -y2\ (y1-y3)\ )\ (-y2+y3)\ -\ (y1-y3)\ (-x1\ (-x2+x3)\ -y1\ (-y2+y3)\ )}{(-x2+x3)\ (y1-y3)\ -\ (x1-x3)\ (-y2+y3)\ )} \right. \\ & \left. ,\ (-(-x2+x3)\ (-x2\ (x1-x3)\ -y2\ (y1-y3)\ )+ \\ & \left. (x1-x3)\ (-x1\ (-x2+x3)\ -y1\ (-y2+y3)\ )\right) \right. \\ & \left. ((-x2+x3)\ (y1-y3)\ -\ (x1-x3)\ (-y2+y3)\ )\right) \right. \\ & \left. ((-x2+x3)\ (y1-y3)\ -\ (x1-x3)\ (-y2+y3)\ )\right\} \Big] \end{array}
```

Compute the orthocenter using the formula.

```
\begin{split} &In[7]: \ \mathbf{p2} = \mathbf{Orthocenter2D}[\mathbf{Triangle2D}[\mathbf{A1}, \mathbf{B1}, \mathbf{C1}]] \\ &Out[7] \ \mathsf{Point2D}\Big[\Big\{\frac{(x1\,x3+y2^2)\ (y1-y3)\ -(x2\,x3+y1^2)\ (y2-y3)\ -(y1-y2)\ (x1\,x2+y3^2)}{-x2\,y1+x3\,y1+x1\,y2-x3\,y2-x1\,y3+x2\,y3}, \\ & \frac{(x1-x2)\ (x3^2+y1\,y2)\ -(x1-x3)\ (x2^2+y1\,y3)\ +(x2-x3)\ (x1^2+y2\,y3)}{-x2\,y1+x3\,y1+x1\,y2-x3\,y2-x1\,y3+x2\,y3}\Big\}\Big] \end{split}
```

The x- and y-coordinates of the intersection point and the orthocenter are identical.

## Discussion

This is the plot of a numerical example.

```
In[9]: Sketch2D[{Triangle2D[A1, B1, C1], lnA, lnB, lnC, p2} //.
{x1 -> -1, y1 -> -2, x2 -> 4, y2 -> 0, x3 -> -2, y3 -> 3}];
```

-2

4

0

2 4

# tridist.nb

## Hypotenuse Midpoint Distance

### **Exploration**

Prove that the midpoint of the hypotenuse of a right triangle is equidistant from the vertices.

## Approach

Without loss of generality, create a triangle in a convenient position with the right angle vertex at the origin and the other two vertices at (a, 0) and (0, b). Create the midpoint of the hypotenuse and then examine the distance from the midpoint to each of the vertices.

## Solution

Create the points defining the triangle's vertices.

In[1]: Clear[a, b]; p1 = Point2D[0, 0]; p2 = Point2D[a, 0]; p3 = Point2D[0, b];

Construct the midpoint of the hypotenuse.

 $In[2]: \mathbf{P} = Point2D[p2, p3]$  $Out[2] Point2D[\left\{\frac{a}{2}, \frac{b}{2}\right\}]$ 

The distances from the midpoint to the vertices are equal by inspection.

 $In[3]: Map[Distance2D[P, #]&, \{p1, p2, p3\}]$ Out[3]  $\left\{\sqrt{\frac{a^2}{4} + \frac{b^2}{4}}, \sqrt{\frac{a^2}{4} + \frac{b^2}{4}}, \sqrt{\frac{a^2}{4} + \frac{b^2}{4}}\right\}$ 

## Discussion

This is the plot of a numerical example.

# trieuler.nb

## Euler's Triangle Formula

### **Exploration**

If T is a triangle, and P and r are the center and radius of the circle inscribed in T, and Q and R are the center and radius of the circle circumscribing T, show that

$$d^2 = R^2 - 2rR$$

where d is the distance from P to Q.

## Approach

Construct the required geometry using a triangle in a special, but sufficiently general, position. Show the equation is true by showing that the difference of the left and right side of the equation is identically zero.

## Solution

Construct the required geometry and find symbolic expressions for d, r and R.

```
In[1]: Clear[a, b];
P1 = Point2D[0, 0];
P2 = Point2D[1, 0];
P3 = Point2D[a, b];
T = Triangle2D[P1, P2, P3];
Ci = Circle2D[T, Inscribed2D];
Cc = Circle2D[T, Circumscribed2D];
{P, r} = {Point2D[Ci], Radius2D[Ci]};
{Q, R} = {Point2D[Cc], Radius2D[Cc]};
d = Distance2D[P, Q];
```

In this step we take a slight diversion to show that  $b^2 + (a - a^2 - b^2)^2 = A^2 B^2$ . We will use this substitution in a subsequent step. Variables A and B are the lengths of the sides of the triangle, that is  $a^2 + b^2 = A^2$  and  $(1 - a)^2 + b^2 = B^2$ .

```
In[2]: Clear[A, B];
Factor[b<sup>2</sup>+ (a - a<sup>2</sup> - b<sup>2</sup>)<sup>2</sup>] //. {
    1 - 2*a + a<sup>2</sup> -> (a - 1)<sup>2</sup>,
    (-1 + a)<sup>2</sup> + b<sup>2</sup> -> B<sup>2</sup>,
    a<sup>2</sup> + b<sup>2</sup> -> A<sup>2</sup>}
Out[2] A<sup>2</sup> B<sup>2</sup>
```

The expression  $d^2 - (R^2 - 2rR)$  should be zero if the equation  $d^2 = R^2 - 2rR$  is true, so we will apply a series of simplifications to show that the expression is identically zero. Notice throughout expressions of the form  $\sqrt{Z^2} = Z$  whenever Z is known to be positive.

$$In \{3\}: el = d^{2} - (R^{2} - 2 * r * R) //. \{a^{2} + b^{2} - > A^{2},l - 2a + a^{2} + b^{2} - > (1 - a)^{2} + b^{2},(1 - a)^{2} + b^{2} - > B^{2},Sqrt[A^{2}] - > A,Sqrt[B^{2}] - > B,b^{2} + (a - a^{2} - b^{2})^{2} - > A^{2} * B^{2} \}$$
  
Out [3]  $-\frac{A^{2}B^{2}}{4b^{2}} + (-\frac{1}{2} + \frac{a + A}{1 + A + B})^{2} + (\frac{a - a^{2} - b^{2}}{2b} + \frac{b}{1 + A + B})^{2} + (\sqrt{2}\sqrt{\frac{A^{2}B^{2}}{b^{2}}}\sqrt{\frac{(-1 + \frac{1}{2}(1 + A + B))(-A + \frac{1}{2}(1 + A + B))(-B + \frac{1}{2}(1 + A + B))}{1 + A + B}}$ 

Make substitutions to remove some of the radicals. S is the semi-perimeter of the triangle, S = (1 + A + B)/2.

$$In[4]: Clear[S];$$

$$e2 = e1 //. \{$$

$$1 + A + B -> 2 * S,$$

$$Sqrt[A^2 * B^2 / b^2] -> A * B / b\}$$

$$Out[4] - \frac{A^2 B^2}{4 b^2} + \left(-\frac{1}{2} + \frac{a + A}{2 S}\right)^2 + \left(\frac{a - a^2 - b^2}{2 b} + \frac{b}{2 S}\right)^2 + \frac{A B \sqrt{\frac{(-1 + S) (-A + S) (-B + S)}{S}}}{b}$$

This is the crucial substitution. Using Heron's formula for the area of a triangle,

Area = 
$$\sqrt{S(S-A)(S-B)(S-C)}$$

(C = 1 in this case) and the standard formula for area,

Area = base × height = 
$$1 \times b/2 = b/2$$
,

we can eliminate the remaining radical.

```
In[5]: e3 = e2 //. \{
Sqrt[(-1+S) (-A+S) (-B+S)/S] \rightarrow Areal/S,
Areal \rightarrow b/2\}
Out[5] - \frac{A^2 B^2}{4 b^2} + \left(-\frac{1}{2} + \frac{a+A}{2 S}\right)^2 + \left(\frac{a-a^2-b^2}{2 b} + \frac{b}{2 S}\right)^2 + \frac{A B}{2 S}
```

If the expression is a fraction, we don't care what value the denominator is, so long as the numerator is zero.

```
In[6]: e4 = Numerator[Together[e3]]
Out[6] a^{2}b^{2} + 2 a A b^{2} + A^{2}b^{2} + b^{4} - 2 a^{2}b^{2}S - 2 A b^{2}S - 2 b^{4}S + 2 A b^{2}B S + a^{2}S^{2} - 2 a^{3}S^{2} + a^{4}S^{2} + b^{2}S^{2} - 2 a b^{2}S^{2} + 2 a^{2}b^{2}S^{2} + b^{4}S^{2} - A^{2}B^{2}S^{2}
```

Repeated expansions and substitutions confirm that the expression is zero and that the original equation is an identity.

Out[7] 0

# trirad.nb

## Triangle Radii

### **Exploration**

Prove that the radius, r, of a circle inscribed in a triangle is given by

$$r = \frac{1}{2}\sqrt{\frac{P}{S}}$$

where  $S = s_1 + s_2 + s_3$ ,  $P = (-s_1 + s_2 + s_3)(s_1 - s_2 + s_3)(s_1 + s_2 - s_3)$  and  $s_1$ ,  $s_2$  and  $s_3$  are the lengths of the triangle's sides. Furthermore, prove that the radius, R, of the circle circumscribing the triangle is given by

$$R = \frac{s_1 s_2 s_3}{\sqrt{PS}}.$$

#### Approach

Construct a triangle with the given side lengths. Construct the associated inscribed and circumscribed circles and examine the radius of these circles.

### Solution

Construct the triangle with the given side lengths.

```
In[1]: Clear[s1, s2, s3, S, P, e1, e2];
T = (Triangle2D[{s1, s2, s3}] // FullSimplify) //.
        {Sqrt[-e1_/s_Symbol^2] :> Sqrt[-e1] / s}
Out[1] Triangle2D[{0, 0}, {s3, 0},
        { (-s1<sup>2</sup> + s2<sup>2</sup> + s3<sup>2</sup> / 2 s3 , 0), (s1 + s2 + s3) (s1 + s2 + s3) (s1 + s2 + s3) / 2 s3 }]
```

Construct the inscribed circle and compute its radius. Simplify the resulting expression using appropriate substitutions.

```
In[2]: ((Radius2D[Circle2D[T, Inscribed2D]] // FullSimplify) //. 
 {Sqrt[s_Symbol^2] -> s} /. 
 Sqrt[-e1_/e2_] :> Sqrt[e1/Factor[-e2]]) //. 
 {(s1 - s2 - s3) (s1 + s2 - s3) (s1 - s2 + s3) -> -P, 
 s1 + s2 + s3 -> S} 
Out[2] <math>\frac{\sqrt{\frac{P}{S}}}{2}
```

Construct the circumscribed circle and compute its radius. Simplify the resulting expression using appropriate substitutions.

# trisides.nb

## Triangle Side Lengths from Altitudes

## **Exploration**

Prove that the lengths of a triangle's sides whose altitudes are of lengths  $h_1$ ,  $h_2$  and  $h_3$  are given by

$$s_1 = \frac{2h_1H_1}{H}, \ s_2 = \frac{2h_2H_2}{H} \text{ and } s_3 = \frac{2h_3H_3}{H}$$

where  $H_1 = h_2 h_3$ ,  $H_2 = h_1 h_3$  and  $H_3 = h_1 h_2$ , and

$$H = \sqrt{(H_1 + H_2 - H_3)(H_1 - H_2 + H_3)(-H_1 + H_2 + H_3)(H_1 + H_2 + H_3)}.$$

## Approach

Construct a triangle with the formulas given for the side lengths and show that the altitude lengths are  $h_1$ ,  $h_2$  and  $h_3$ .

## Solution

Construct the triangle with the given side lengths.

$$In [1]: Clear [h1, h2, h3, H1, H2, H3, H];$$
  
T = Triangle2D[2 \* {h1 \* H1^2, h2 \* H2^2, h3 \* H3^2} / H] // FullSimplify  
Out [1] Triangle2D[{0, 0}, { $\frac{2 h3 H3^2}{H}, 0$ }, { $\frac{-h1^2 H1^4 + h2^2 H2^4 + h3^2 H3^4}{H h3 H3^2}$ ,  
 $\sqrt{-\frac{h1^4 H1^8 + (h2^2 H2^4 - h3^2 H3^4)^2 - 2 h1^2 H1^4 (h2^2 H2^4 + h3^2 H3^4)}{H^2 h3^2 H3^4}}$ }]

Compute the lengths of each altitude (squared), which is the distance from the vertex to the opposite side.

```
In [2]: alts1 = \{ 
Distance2D[Point2D[T, 1], Line2D[T, 2, 3]]^2, 
Distance2D[Point2D[T, 2], Line2D[T, 1, 3]]^2, 
Distance2D[Point2D[T, 3], Line2D[T, 1, 2]]^2 // Fullsimplify 
Out [2] <math>\{-\frac{h1^4 H1^8 + (h2^2 H2^4 - h3^2 H3^4)^2 - 2 h1^2 H1^4 (h2^2 H2^4 + h3^2 H3^4)}{H^2 h1^2 H1^4}, -\frac{h1^4 H1^8 + (h2^2 H2^4 - h3^2 H3^4)^2 - 2 h1^2 H1^4 (h2^2 H2^4 + h3^2 H3^4)}{H^2 h2^2 H2^4}, -\frac{h1^4 H1^8 + (h2^2 H2^4 - h3^2 H3^4)^2 - 2 h1^2 H1^4 (h2^2 H2^4 + h3^2 H3^4)}{H^2 h3^2 H3^4} \}
```

A few substitutions verify that the altitude lengths (squared) are the expected values.

In[3]: alts2 = alts1 //. {H1 -> h2 \*h3, H2 -> h1 \*h3, H3 -> h1 \*h2} // FullSimplify // Factor  $\textit{Out[3]} \ \left\{ -\frac{1}{H^2} \ (\texttt{h1}^2 \ (\texttt{h1} \ \texttt{h2} - \texttt{h1} \ \texttt{h3} - \texttt{h2} \ \texttt{h3}) \right.$  $(h1 \ h2 \ + \ h1 \ h3 \ - \ h2 \ h3) \ (h1 \ h2 \ - \ h1 \ h3 \ + \ h2 \ h3) \ (h1 \ h2 \ + \ h1 \ h3 \ + \ h2 \ h3) \ ) \ ,$  $-\frac{1}{H^2}(h2^2(h1 h2 - h1 h3 - h2 h3))$ (h1 h2 + h1 h3 - h2 h3) (h1 h2 - h1 h3 + h2 h3) (h1 h2 + h1 h3 + h2 h3)),  $-\frac{1}{H^{2}}\left(h3^{2} \ (-h1 \ h2 \ -h1 \ h3 \ +h2 \ h3) \ (h1 \ h2 \ -h1 \ h3 \ +h2 \ h3) \ (-h1 \ h2 \ +h1 \ h3 \ +h2 \ h3)$ (h1 h2 + h1 h3 + h2 h3))In[4]: alts3 = alts2 //. {h2 \* h3 -> H1, h1 \* h3 -> H2, h1 \* h2 -> H3}  $\textit{Out[4]} \ \left\{ -\frac{\texttt{h1}^2 \ (-\texttt{H1} - \texttt{H2} + \texttt{H3}) \ (\texttt{H1} - \texttt{H2} + \texttt{H3}) \ (-\texttt{H1} + \texttt{H2} + \texttt{H3}) \ (\texttt{H1} + \texttt{H2} + \texttt{H3}) \right\}$ H<sup>2</sup>  $-\frac{h2^2 (-H1 - H2 + H3) (H1 - H2 + H3) (-H1 + H2 + H3) (H1 + H2 + H3)}{-H1 + H2 + H3}$ H2  $-\frac{h3^{2} (H1 - H2 - H3) (H1 + H2 - H3) (H1 - H2 + H3) (H1 + H2 + H3)}{(H1 - H2 + H3) (H1 - H2 + H3)}$ H<sup>2</sup> In[5]: alts4 = alts3 /. { (H1 - H2 - H3) (H1 + H2 - H3) (H1 - H2 + H3)  $(H1 + H2 + H3) -> -H^{2},$ (-H1 - H2 + H3) (H1 - H2 + H3) (-H1 + H2 + H3) (H1 + H2 + H3)  $-> -H^{2}$ 

```
Out[5] \{h1^2, h2^2, h3^2\}
```

Part IX Epilogue

## Installation Instructions

To make full use of the files provided on the CD-ROM, two software applications need to be installed on your computer: Adobe's Acrobat Reader and Wolfram Research's Mathematica. Acrobat Reader is used to view and print the PDF (Portable Document Format) files on the CD-ROM. The PDF files contain typeset text reproducing all the material in the book *Exploring Analytic Geometry with Mathematica*. Mathematica is used to view the notebook files and execute the Descarta2D packages. If Mathematica is not available, Wolfram Research's MathReader application may be used to view the notebook files, although the Descarta2D packages cannot be interactively executed using MathReader.

#### Installing Acrobat Reader and Viewing PDF Files

Acrobat Reader is a licensed product of Adobe Systems Incorporated. It is available as a free download from Adobe's web site, www.adobe.com. For Windows systems, a version of Acrobat Reader is provided on the CD-ROM and may be installed by double-clicking the ar302.exe file icon in the AcrobatReader folder. For other computer systems, you should download the appropriate files from Adobe's web site and follow the installation instructions provided. Acrobat Reader may already be installed on your computer system since PDF files are commonly used as a format for typeset files downloaded from the World Wide Web.

The entire typeset text of this book is stored on the CD-ROM in the Book folder. Doubleclicking any PDF file in the Book folder will cause Acrobat Reader to open the file and will allow viewing or printing of the typeset text and illustrations. The PDF files can be read directly off the CD-ROM using Acrobat Reader, or they can be copied to any convenient location on your computer's hard disk drive.

#### Installing Mathematica and Viewing Notebook Files

This book is designed around *Mathematica*, a product developed and licensed by Wolfram Research Incorporated. To gain maximum benefit from the book and the files provided on the CD-ROM, *Mathematica* should be installed on your computer. Information about licensing *Mathematica* is available at Wolfram's web site, www.wolfram.com. Instructions for installing

Mathematica are provided with Mathematica itself. Mathematica should be installed before installing the Descarta2D files provided on the CD-ROM.

If Mathematica is not installed on your computer system, you can still view the contents of the notebook files on the CD-ROM using Wolfram's MathReader application (notebook files have the extension .nb). MathReader is also a licensed product of Wolfram Research Incorporated. It is available as a free download from Wolfram's web site, www.wolfram.com. For Windows systems, a version of MathReader is provided on the CD-ROM and may be installed by double-clicking the Setupex.exe file icon in the MathReader folder on the CD-ROM. If Mathematica is installed on your computer system, do not install the MathReader software. Mathematica provides all the capabilities of MathReader.

By using Mathematica or MathReader you can view any notebook file directly off the CD-ROM. However, it is recommended that you install the files in the folder Descarta2D as described in the next section prior to viewing them. Generally, you will not be able to follow the hyperlinks in the notebook files unless they are installed on your computer's hard disk drive as described in the next section.

#### Installing the Descarta2D Files

When Mathematica or MathReader is installed on your computer system, a directory structure is created providing a standard location for installing applications such as Descarta2D. On a Windows system the standard Mathematica installation creates a directory structure whose path name is

```
c:\Program Files\Wolfram Research\Mathematica\3.0\AddOns\Applications\
```

Mathematica and MathReader will search this directory when trying to locate packages (.m files) and notebook files (.nb files). In order to install *Descarta2D* and related documentation so that *Mathematica* can find these files, copy the **Descarta2D** folder and all its contents from the CD-ROM into the **Applications** folder of the directory path named above.

If you plan to use *Descarta2D* on a different operating system, refer to the installation instructions for your *Mathematica* system to determine the name of the proper directory path for add-on applications. This information is also provided in Wolfram's *Mathematica* book. For example, the high-level directory name for *Mathematica* on a Macintosh is Mathematica 3.0. On a Unix or NeXT system the high-level directory name is /usr/local/Mathematica3.0.

#### Mathematica Help Browser

The interactive Front End program that serves as the user interface for *Mathematica* provides a Help Browser for accessing *Mathematica* documentation (in fact, the entire text of Stephen Wolfram's *Mathematica* book can be accessed using the Help Browser). The Help Browser is activated by clicking the Help>Help... item on the Front End's menu bar. The *Descarta2D* documentation and this entire book can also be linked into the *Mathematica* Help Browser. This is accomplished by clicking the Help>Rebuild Help Index... item on the Front End's menu bar after the Descarta2D folder has been copied into the proper folder. After the help index is rebuilt, the Descarta2D documentation and the notebooks representing this book can be accessed by clicking the Add-ons radio button on the Help Browser dialog. The high-level category name that opens access to the Descarta2D categories is Descarta2D.

## **Package Loading**

In order to initialize the Descarta2D software in any Mathematica session enter the command <<Descarta2D<sup>'</sup>. This command will load the initialization file for Descarta2D and will enable Mathematica to find and load any other Descarta2D package as it is needed.

## Bibliography

- Bowyer, Adrian and John Woodwark, A Programmer's Geometry, First Edition, Butterworths, London, UK, 1983.
- [2] Bowyer, Adrian and John Woodwark, Introduction to Computing with Geometry, First Edition, Information Geometers Ltd., Winchester, UK, 1993.
- [3] Copland, Sr., Arthur H., Geometry, Algebra, and Trigonometry by Vector Methods, First Edition, The Macmillan Company, New York, 1962.
- [4] Dörrie, Heinrich, 100 Great Problems of Elementary Mathematics, Second Edition, Dover Publications, Inc., New York, 1965.
- [5] Eisenhart, Luther Pfahler, Coordinate Geometry, Dover Edition, Dover Publications, Inc., New York, 1960.
- [6] Gasson, Peter C., Geometry of Spatial Forms, Revised Edition, Ellis Horwood Limited, Chichester, West Sussex, England, 1983.
- [7] Gellert W., H. Küster, M. Hellwich, H. Kästner (editors), The VNR Concise Encyclopedia of Mathematics, First Edition, Van Nostrand Reinhold Company, New York, 1977.
- [8] Gray, Alfred, Modern Differential Geometry of Curves and Surfaces, First Edition, CRC Press, Boca Raton, Florida, 1993.
- [9] Gullberg, Jan, Mathematics From the Birth of Numbers, First Edition, W. W. Norton & Company, New York, 1997.
- [10] Itô, Kiyosi (editor), Encyclopedic Dictionary of Mathematics, Second Edition, The MIT Press, Cambridge, Massachusetts, 1987.
- [11] Lee, Eugene T. Y., Gerald E. Farin (editor), Geometric Modeling: Algorithms and New Trends, First Edition, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1987.
- [12] Lehmann, Charles H., Analytic Geometry, Third Printing, John Wiley & Sons, Inc., New York, 1947.

- [13] Meserve, Bruce E., Fundamental Concepts of Geometry, First Edition, Dover Publications, Inc., New York, 1983.
- [14] Mortenson, Michael E., Geometric Modeling, First Edition, John Wiley & Sons, New York, 1985.
- [15] Oakley, C. O., An Outline of Analytic Geometry, First Edition, Barnes & Nobel, Inc., New York, 1949.
- [16] Ogilvy, C. Stanley, Excursions in Geometry, First Edition, Oxford University Press, New York, 1969.
- [17] Peck, William G., An Treatise on Analytical Geometry, First Edition, A. S. Barnes & Company, New York and Chicago, 1873.
- [18] Pedoe, Dan, Geometry: A Comprehensive Course, Dover Edition, Dover Publications, Inc., New York, 1988.
- [19] Salmon, George, A Treatise on Conic Sections, Sixth Edition, Chelsea Publishing Company, New York, date unknown.
- [20] Selby, Samuel M. (editor), Standard Mathematical Tables, Nineteenth Edition, The Chemical Rubber Co., Cleveland, Ohio, 1971.
- [21] Smith, Charles, An Elementary Treatise on Conic Sections by the Methods of Coordinate Geometry, New Edition, Macmillan and Co., Limited, London, 1927.
- [22] Smith, Edward S. and Meyer Salkover and Howard K. Justice, Analytic Geometry, Fifth Edition, John Wiley & Sons, Inc., London, 1943.
- [23] Smith, Percey F. and Arthur S. Gale and John H. Neelley, New Analytic Geometry, Revised Edition, Ginn and Company, Boston, 1928.
- [24] Thomas, Jr., George B., Calculus and Analytic Geometry, Alternate Edition, Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1972.
- [25] Underwood, R. S. and Fred W. Sparks, Analytic Geometry, First Edition, Houghton Mifflin Company, Boston, 1948.
- [26] Weisstein, Eric W., CRC Concise Encyclopedia of Mathematics, First Edition, CRC Press LLC, Boca Raton, Florida, 1999.
- [27] Wells, David, The Penguin Dictionary of Curious and Interesting Geometry, First Edition, Penguin Books, London, England, 1991.
- [28] Wolfram, Stephen, Mathematica, A System for Doing Mathematics by Computer, Second Edition, Addison-Wesley Publishing Company, Inc., Redwood City, California, 1991.

## Index

[INDEX USAGE: see footnote <sup>1</sup>] altitude triangle, 128-130 angle arc end angle, 105 arc start angle, 105 bulge factor arc, 109 triangle, 117 two lines, 55 angle bisector triangle, 128, 130 two lines, 73 Angle2D Arc2D, 389 Ellipse2D, 423example, 53, 56, 117, 138 help, 341 Hyperbola2D, 448 Line2D, 460 Parabola2D, 481 Triangle2D, 547 usage, 457 Apex2D ConicArc2D, 419 help, 341 usage, 415

<sup>1</sup>INDEX USAGE: The keyword index is set up with main and sub-entries. If a keyword cannot be found as a main entry, one should try to find it as a sub-entry of some more general term. Page numbers in normal times font refer to sections in the main body of the book. Page numbers in sans serif font refer to entries in the Descarta2D Command Browser. Page numbers in slanted font are references in the Descarta2D Packages.

arc

arc length, 232 bulge factor, 107 center point, 105 centroid, 115 complement, 108 definition, 105 description, 328 end angle, 105 end point, 105 midpoint, 115 overview, 15 parametric equations, 111, 112 radius, 105 ray points, 113 reflection, 108 sector, 106 sector area, 241 segment area, 242 semicircle, 105 span, 105, 232 start angle, 105 start point, 105 three-point, 110 arc length approximate, 236 arc, 232 conic arc, 236 definition, 229 ellipse, 234 general curve, 233 hyperbola, 234 overview, 20 parabola, 234, 236

summary of functions, 236 Arc2D Angle2D, 389 ArcLength2D, 396 Area2D, 399 Bulge2D, 390 Circle2D, 391 Complement2D, 391 construction, 392, 393 description, 328 evaluation, 388 example, 106, 108, 110-113 graphics, 388 help, 342 IsValid2D, 389 Point2D, 391 PrimaryAngleRange2D, 390 Radius2D, 390 Reflect2D, 390 representation, 387 Rotate2D, 390 Scale2D, 391 Span2D, 395 Translate2D, 391 usage, 387 validation, 388, 389 ArcLength2D Arc2D, 396 Circle2D, 396 ConicArc2D, 396 Ellipse2D, 397 example, 229, 230, 232-235 help, 342 Hyperbola2D, 397 Line2D, 397 Parabola2D, 398 Segment2D, 397 usage, 395 area arc sector, 241 arc segment, 242 circle, 240, 251 conic arc, 248, 250, 251

definition, 237 ellipse, 242 ellipse sector, 244 ellipse segment, 243 Heron's formula, 238, 249 hyperbola, 250 hyperbola sector, 245 hyperbola segment, 245 overview, 20 parabola segment, 246 polygon, 231 rectangle, 237 square, 237 summary of functions, 249 triangle, 237, 238, 249-251 Area2D Arc2D, 399 Circle2D, 400ConicArc2D, 400, 401 Ellipse2D, 401 example, 240, 242, 243, 248 help, 342 Triangle2D, 403usage, 399 AskCurveLength2D command, 513 help, 343 usage, 511 asymptote curve, 47 hyperbola, 159 Asymptotes2D help, 343 Hyperbola2D, 413usage, 411 biarc carrier circles, 311 configuration parameters, 311 defining constants, 313 definition, 311 entry direction, 114

exit direction, 114

knot circle, 314, 316

846
knot point, 314 knot point, incenter, 322 number of solutions, 313 radii ratio, 313 radius sign constant, 311 bulge factor arc, 107 associated angles, 109 complement arc, 108 reflected arc, 108 semicircle, 107 Bulge2D Arc2D, 390 help, 343 usage, 387 CD-ROM organization, 4 center arc, 105 circle, 85 ellipse, 146 hyperbola, 159 centroid arc, 115 triangle, 120, 129 Centroid2D example, 121 help, 343 usage, 545cevian triangle, 128, 130 ChopImaginary2D computation, 477 help, 343 usage, 477 chord parameters, 235 circle Apollonius, 102 area, 240, 251 biarc carrier, 311 biarc knot circle, 314, 316 Carlyle, 103

Castillon, 103 center, 85 circumference, 231 circumscribed, 122 coaxial, 96 coincident, 87 concentric, 87 definition, 85 description, 329 from diameter, 89 from quadratic, 178 general equation, 88 inscribed, 123, 128 overview, 14 parametric equations, 99 pencil, 96 polar equation, 101 radical axis, 97, 102, 103 radical center, 98, 101 radius, 85 rational parameterization, 100 reciprocal, 310 standard equation, 85 tangent, 283, 285-288, 290, 291 tangent line, 255, 264, 280 three points, 90, 102Circle2D Arc2D, 391 ArcLength2D, 396Area2D, 400 Circumference2D, 396 construction, 409, 410, 509 description, 329 evaluation, 406 example, 85, 88, 90, 96, 99, 100, 106, 122, 123 graphics, 406, 497 help, 343 IsValid2D, 406 Parameters2D, 455 Pencil2D, 486 Point2D, 408 Quadratic2D, 405, 409

radical axis, 408 Radius2D, 407 Reflect2D, 407 representation, 405 Rotate2D, 407 Scale2D, 407 SectorArea2D, 400 SegmentArea2D, 400 Translate2D, 408 Triangle2D, 553 usage, 405 validation, 406 circumference circle, 231 ellipse, 234 Circumference2D Circle2D, 396 Ellipse2D, 397 example, 232 help, 344 usage, 395 Circumscribed2D example, 122 help, 344 usage, 545 coaxial circles, 96 coincident circle, 87 line, 54 collinear point, 36, 38, 58 complement arc, 108 Complement2D Arc2D, 391 help, 345 usage, 387 concentric circle, 87 concurrent line, 74 configuration parameters

biarc, 311 conic center point, 184, 192 classification, 184 conic arc, 197 construction, 185 description, 330 intersection points, 189 MedialEquations2D, 473 pencil, 189 polar equation, 192 tangent, 293, 296, 298, 301 tangent line, 266, 271, 273 tangent line segment, 272 translate, 191 vertex equation, 186 conic arc arc length, 236 area, 248, 250 center point, 200 conic, 197 defining points, 193 definition, 193 equation, 194 parametric equations, 198, 200 projective discriminant, 193, 196 rho, p. 193, 196 shoulder point, 200 ConicArc2D Apex2D, 419 ArcLength2D, 396 Area2D, 400, 401 description, 330 evaluation, 416 example, 194, 198, 199 graphics, 416 help, 345 IsValid2D, 417 Loci2D, 419 Point2D, 419 Quadratic2D, 416 Reflect2D, 418 representation, 415

Rho2D, 417 Rotate2D, 418Scale2D, 418 Span2D, 396 Translate2D, 418 usage, 415 validation, 416, 417 conjugate hyperbola, 164 conjugate axis hyperbola, 160 Conjugate2D example, 165 help, 345 Hyperbola2D, 450usage, 445coordinates rectangular, 28 Coordinates2D help, 345 Point2D, 490 usage, 489 coords XCoordinate2D, 491 YCoordinate2D, 491Cramer's Rule three equations, 49 two equations, 49 curve approximated by polygon, 231 asymptotes, 47 definition, 46 extent, 47 intercepts, 47 symmetry, 47 CurveLength2D help, 345 option, 512usage, 511 CurveLimits2D command, 513 help, 346 usage, 511

Directrices2D Ellipse2D, 413 example, 140, 151 help, 346 Hyperbola2D, 413Parabola2D, 413 usage, 411 directrix ellipse, 145 hyperbola, 159 parabola, 135 distance between points, 30 line, 82 parallel lines, 81 point to circle, 95 point to line, 68 point to quadratic, 281 polar coordinates, 38 Distance2D example, 32, 69, 95 help, 346 point to circle, 407 point to line, 460two coordinates, 491 two points, 491 usage, 489 eccentricity ellipse, 145 hyperbola, 159, 173 parabola, 135 Eccentricity2D Ellipse2D, 412 example, 140, 165 help, 346 Hyperbola2D, 412Parabola2D, 412usage, 411 ellipse apoapsis, 157 arc length, 234 area, 242 center, 146

circumference, 234 construction, 151, 153 definition, 145 description, 331 directrix, 145 eccentricity, 145 focal chord, 146 focal chord length, 156 focus, 145 from quadratic, 180 general equation, 147 latus rectum, 146 major axis, 146 minor axis, 146 overview, 18 parametric equations, 155 periapsis, 157 polar equation, 157 rational equations, 155 sector area, 244 segment area, 243 similar, 157 standard equation, 147 tangent line, 274, 281 vertex equation, 187 vertices, 146 Ellipse2D Angle2D, 423 ArcLength2D, 397 Area2D, 401 Circumference2D, 397 construction, 425, 426 description, 331 Directrices2D, 413 Eccentricity2D, 412 equation, 422evaluation, 422 example, 146, 152–155 FocalChords2D, 414Foci2D, 412 graphics, 422 help, 346 IsValid2D, 423

Line2D, 425 Line2D, polar, 425 Parameters2D, 455 Point2D, 424 Point2D, pole, 425 Reflect2D, 424representation, 421 Rotate2D, 424 Scale2D, 424 SectorArea2D, 401 SegmentArea2D, 401 SemiMajorAxis2D, 423 SemiMinorAxis2D, 423 Translate2D, 424 usage, 421 validation, 422, 423 Vertices2D, 412entry direction biarc, 114 equation circle, 85, 88 conic arc, 194 conic, vertex, 186 definition, 41 graph, 41 hyperbola, 161, 166 locus, 46, 47 overview, 10 parabola, 135, 136, 139 parametric, 47 polar, 47, 49 rectangular, 47 reflect, 224 Reflect2D, 540root, 41 rotate, 220 Rotate2D, 541 scale, 222 Scale2D, 542solution, 41 solving, 42 TangentEquation2D, 532 translate, 218

Translate2D, 543Equation2D example, 41, 76 help, 347 Line2D, 428 Quadratic2D, 428 usage, 427 exit direction biarc, 114 Exploration apollon.nb, 102, 557 arccent.nb, 115, 559 arcentry.nb, 114, 561 arcexit.nb, 114, 563 archimed.nb, 289, 565 arcmidpt.nb, 115, 567 caarclen.nb, 236, 569 caarea1.nb, 250, 571 caarea2.nb, 251, 573 cacenter.nb, 200, 575 cacircle.nb, 199, 577 camedian.nb, 200, 579 caparam.nb, 200, 581 carlyle.nb, 103, 583 castill.nb, 103, 585 catnln.nb, 200, 589 center.nb, 192, 591 chdlen.nb, 101, 593 cir3pts.nb, 102, 595 circarea.nb, 251, 597 cirptmid.nb, 102, 599 cramer2.nb, 49, 601 cramer3.nb, 49, 603 deter.nb, 48, 605 elfocdir.nb, 157, 607 elimlin.nb, 191, 609 elimxy1.nb, 190, 611 elimxy2.nb, 191, 613 elimxy3.nb, 191, 615 elldist.nb, 157, 617 ellfd.nb, 157, 619 ellips2a.nb, 156, 623 elllen.nb, 156, 625

ellrad.nb, 157, 627 ellsim.nb, 157, 629 ellslp.nb, 281, 631 eqarea.nb, 251, 633 eyeball.nb, 280, 637 gergonne.nb, 129, 639 heron.nb, 249, 641 hyp2a.nb, 173, 643 hyp4pts.nb, 310, 645 hyparea.nb, 250, 647 hypeccen.nb, 173, 651 hypfd.nb, 173, 653 hypinv.nb, 173, 657 hyplen.nb, 173, 659 hypslp.nb, 281, 661 hyptrig.nb, 173, 663 intrsct.nb, 81, 665 inverse.nb, 227, 667 johnson.nb, 101, 671 knotin.nb, 322, 675 lndet.nb, 82, 677 lndist.nb, 82, 679 lnlndist.nb, 82, 681 lnquad.nb, 280, 685 lnsdst.nb, 81, 687 lnsegint.nb, 83, 689 lnsegpt.nb, 82, 691 lnsperp.nb, 82, 693 lntancir.nb, 280, 695 lntancon.nb, 281, 697 mdcircir.nb, 213, 699 mdlncir.nb, 213, 703 mdlnln.nb, 213, 705 mdptcir.nb, 212, 707 mdptln.nb, 212, 711 mdptpt.nb, 212, 713 mdtype.nb, 214, 715 monge.nb, 281, 717 narclen.nb, 236, 719 normal.nb, 281, 721 pb3pts.nb, 143, 723 pb4pts.nb, 310, 725 pbang.nb, 144, 727

pbarch.nb, 143, 729 pbarclen.nb, 236, 731 pbdet.nb, 143, 733 pbfocchd.nb, 142, 735 pbslp.nb, 280, 737 pbtancir.nb, 144, 739 pbtnlns.nb, 280, 743 polarcir.nb, 101, 745 polarcol.nb, 38, 747 polarcon.nb, 192, 749 polardis.nb, 38, 751 polarell.nb, 157, 753 polareqn.nb, 49, 755polarhyp.nb, 173, 757 polarpb.nb, 144, 759 polarung.nb, 38, 761 pquad.nb, 192, 763 ptscol.nb, 37, 765 radaxis.nb, 102, 767 radcntr.nb, 101, 769 raratio.nb, 103, 771 reccir.nb, 310, 773 recptln.nb, 310, 775 recquad.nb, 310, 777 reflctpt.nb, 226, 779 rtangcir.nb, 101, 781 rttricir.nb, 128, 783 shoulder.nb, 200, 785 stewart.nb, 38, 787 tancir1.nb, 290, 789 tancir2.nb, 290, 791 tancir3.nb, 290, 793 tancir4.nb, 291, 795 tancir5.nb, 291, 797 tancirpt.nb, 281, 799 tetra.nb, 251, 801 tncirtri.nb, 291, 803 tnlncir.nb, 102, 807 triallen.nb, 129, 809 trialt.nb, 129, 811 triarea.nb, 249, 813 triarlns.nb, 250, 815

tricent.nb, 129, 817

tricev.nb, 130, 819 triconn.nb, 129, 823 tridist.nb, 38, 827 trieuler.nb, 128, 829 trirad.nb, 130, 833 trisides.nb, 130, 835 focal chord ellipse, 146 hyperbola, 159 parabola, 135 FocalChords2D Ellipse2D, 414 help, 347 Hyperbola2D, 414 Parabola2D, 414 usage, 411 FocalLength2D help, 347 Parabola2D, 481 usage, 479 Foci2D Ellipse2D, 412 example, 140, 151 help, 347 Hyperbola2D, 412 Parabola2D, 412 usage, 411 focus ellipse, 145 hyperbola, 159 parabola, 135 FullSimplify Line2D, 460 Quadratic2D, 499 function definition, 39 graph, 46 multiple-valued, 39 periodic, 39 real-valued, 39 Gergonne Point of a triangle, 129

Heron's formula triangle area, 238, 249 horizontal line, 61 hyperbola arc length, 234 area, 250 asymptote, 159 center, 159 conjugate, 164 conjugate axis, 160 construction, 167–169, 310 definition, 159 description, 332 directrix, 159 eccentricity, 159, 173 equilateral, 166, 310 focal chord, 159 focal chord length, 173 focus, 159 from quadratic, 180 general equation, 161 latus rectum, 159 overview, 19 parametric equations, 170, 173 polar equation, 173 rational equations, 172 rectangular, 166 sector area, 245 segment area, 245 standard equation, 161, 166 tangent line, 277, 281 transverse axis, 159vertex equation, 187 vertices, 159 Hyperbola2D Angle2D, 448ArcLength2D, 397 Asymptotes2D, 413 Conjugate2D, 450construction, 450, 451 description, 332 Directrices 2D, 413

Eccentricity2D, 412 evaluation, 446 example, 160, 164, 168-171 FocalChords2D, 414Foci2D, 412 graphics, 446 help, 347 IsValid2D, 447 Line2D, 449 Line2D, polar, 450 Parameters2D, 456Point2D, 449Point2D, pole, 449 Quadratic2D, 446Reflect2D, 448 representation, 445 Rotate2D, 448 Scale2D, 449 SectorArea2D, 402 SegmentArea2D, 402 SemiConjugateAxis2D, 448 SemiTransverseAxis2D, 448 Translate2D, 449 usage, 445validation, 447 Vertices2D, 413 inclination line, 53 Inscribed2D example, 123 help, 348 usage, 545installation Acrobat Reader, 839 Descarta2D, 840 Help Browser, 840 Mathematica, 839 MathReader, 839 intersection line and circle, 91 line and conic, 189 two circles, 92 two conics, 189

two line segments, 82, 83 two lines, 69, 81 inversion transformation, 227 Is2D definition, 472 help, 348 usage, 471 IsApproximate2D help, 348 query, 431 usage, 429 IsCoincident2D circles, 439 coordinates, 439 example, 87 help, 348 lines, 439 list of objects, 440 points, 439 quadratics, 439 usage, 437 IsCollinear2D example, 37 help, 349 list of points, 440 points, 440 usage, 437 IsComplex2D help, 349 query, 431 usage, 429 IsConcentric2D circles, 440 example, 87 help, 349 list of circles, 440 usage, 437 IsConcurrent2D example, 75 help, 349 lines, 441 list of lines, 441

usage, 437 IsDisplay2D default, 512 help, 349 usage, 511 IsNegative2D help, 350 query, 434 usage, 429 IsNumeric2D help, 350query, 432 usage, 429 IsObject2D usage, 471 IsOn2D example, 52, 86, 256 help, 350 point on circle, 441 point on line, 441 point on quadratic, 441 usage, 437 IsParallel2D example, 55 help, 350 lines, 442 list of lines, 442usage, 437 IsPerpendicular2D example, 55 help, 351 lines, 442 list of lines, 443 usage, 437IsReal2D help, 351 query, 433 usage, 429 IsScalar2D help, 351 query, 433 usage, 429 IsScalarPair2D

help, 351 query, 434 usage, 429 IsTangent2D help, 351 line and circle, 443 line and quadratic, 443 two circles, 443 usage, 437 IsTinyImaginary2D help, 352 query, 434 usage, 429 IsTripleParallel2D help, 352 lines, 442list of lines, 442 usage, 437 IsValid2D Arc2D, 389 Circle2D, 406ConicArc2D, 417 default, 472 Ellipse2D, 423help, 352 Hyperbola2D, 447 Line2D, 459 Parabola2D. 481 Point2D, 490 Quadratic2D, 498 Segment2D, 506 Triangle2D, 546 usage, 471 IsZero2D help, 352 query, 432, 435 usage, 429 IsZeroOrNegative2D help, 353 query, 435, 436 usage, 429 knot circle biarc, 314, 316

center, 317 knot point biarc, 314, 318 latus rectum ellipse, 146 hyperbola, 159 parabola, 135 length chord, intersecting circles, 101 line, 229 line segment, 230 Length2D example, 230 help, 353 Segment2D, 507 usage, 505 line angle, 55 angle bisectors, 73 coincident, 54 concurrent, 74 definition, 51 description, 333 determinant form, 82 distance, 82 from quadratic, 176, 177 horizontal, 61 inclination, 53 intercept form, 64 length, 229 normal form, 65 offset from a line, 68 overview, 12 parallel, 54, 60 parametric equations, 78 pencil, 75 perpendicular, 54, 60, 72, 82 perpendicular form, 65 point-slope form, 58 quadratic normal, 280 reciprocal, 310 slope, 53 slope-intercept form, 62

two-point form, 56 vertical, 61 line segment definition, 51 description, 335 end point, 51 length, 230 overview, 13 parametric equations, 80 slope, 53 start point, 51 Line2D Angle2D, 460 ArcLength2D, 397 construction, 462, 463, 508 description, 333 Ellipse2D, 425 equation, 458Equation2D, 428 evaluation, 458 example, 40, 57, 59, 60, 62, 64, 66, 68, 72, 76, 77, 79, 98, 119, 140, 267, 270FullSimplify, 460 graphics, 458 help, 353 Hyperbola2D, 449IsValid2D, 459 normalize, 461 offset, 462 Parabola2D, 483Parallel2D, 463Pencil2D, 485 Perpendicular2D, 463polar (circle), 408Polynomial2D, 428 radical axis, 408 Reflect2D, 461 representation, 458 Rotate2D, 461 Scale2D, 461 Simplify, 460

slope, 462

Slope2D, 460 Translate2D, 461 Triangle2D, 552 usage, 457 validation, 459 Line2D, polar Ellipse2D, 425 Hyperbola2D, 450Parabola2D, 483 Quadratic2D, 463Loci2D ConicArc2D, 419 construction, 465, 468 example, 88, 139, 150, 151, 165, 167, 176-180, 182, 183, 188, 195, 297 help, 354 usage, 465 locus equation, 46, 47 major axis ellipse, 146 MakePrimitives2D command, 513 help, 355 usage, 511 MaxSeconds2D help, 355 option, 516usage, 515 medial curve circle-circle, 210, 213 definition, 201 line-circle, 207, 213 line-line, 206, 213 point-circle, 204, 212 point-line, 202, 212 point-point, 201, 212 MedialEquations2D conic, 473 help, 355 usage, 473 MedialLoci2D circle-circle, 475

construction, 474 example, 73, 202-205, 207-209, 211 help, 355 line-circle, 475 line-line, 475 point-circle, 474 point-line, 474 point-point, 474 usage, 473 median triangle, 120, 128, 130 midpoint arc, 115 Point2D, 493, 508 minor axis ellipse, 146 Monge's Theorem tangent lines, 281 names general conventions, 326 normal quadratic, 281 normalize Line2D, 461 Quadratic2D, 500 numbers complex, 27 conjugate complex, 27 imaginary, 27 integers, 27 rational, 27 real, 27 ObjectNames2D definition, 472 help, 355offset Line2D, 462 Point2D, 493 orthocenter triangle, 129

package

D2DArc2D, 387 D2DArcLength2D, 395 D2DArea2D, 399 D2DCircle2D, 405D2DConic2D, 411 D2DConicArc2D, 415 D2DEllipse2D, 421 D2DEquations2D, 427 D2DExpressions2D, 429 D2DGeometry2D, 437D2DHyperbola2D, 445D2DIntersect2D, 453 D2DLine2D, 457 D2DLoci2D, 465 D2DMaster2D, 469 D2DMedial2D, 473D2DNumbers2D, 477 D2DParabola2D, 479D2DPencil2D, 485D2DPoint2D, 489D2DQuadratic2D, 497 D2DSegment2D, 505 D2DSketch2D, 511 D2DSolve2D, 515D2DTangentCircles2D, 519 D2DTangentConics2D, 523 D2DTangentLines2D, 531 D2DTangentPoints2D, 537 D2DTransform2D, 539 D2DTriangle2D, 545 parabola arc length, 234, 236 construction, 140, 143, 310 definition, 135 description, 334 directrix, 135 eccentricity, 135 focal chord, 135 focal chord length, 142 focus, 135 from quadratic, 178, 179 general equation, 135 latus rectum, 135

overview, 17 parametric equations, 141 polar equation, 144 segment area, 246 standard equation, 136, 139 tangent line, 273, 280 vertex, 135 vertex equation, 186 Parabola2D Angle2D, 481 ArcLength2D, 398 construction, 483 description, 334 Directrices2D, 413Eccentricity2D, 412equation, 480evaluation, 480 example, 137, 138, 141, 142 FocalChords2D, 414 FocalLength2D, 481 Foci2D, 412 graphics, 480 help, 355 IsValid2D, 481 Line2D, 483 Line2D, polar, 483 Parameters2D, 456 Point2D, 482 Point2D, pole, 482 Reflect2D, 481 representation, 479 Rotate2D, 482 Scale2D, 482 SegmentArea2D, 402 Translate2D, 482 usage, 479 validation, 480, 481 Vertices2D, 413 parallel line, 54, 60 tangent line, 261, 279 Paralle12D example, 60, 262, 279

help, 356 Line2D, 463 usage, 457 parameters chord, 235 Parameters2D Circle2D, 455Ellipse2D, 455 example, 235 help, 356 Hyperbola2D, 456Parabola2D, 456usage, 453 parametric equations arc, 111, 112 circle, 99 conic arc, 198, 200 definition, 47 ellipse, 155 hyperbola, 170 line, 78 line segment, 80 parabola, 141 quadratic, 192 pencil circle, 96 conic, 189 line, 75 quadratic, 294 Pencil2D Circle2D, 486example, 76, 77, 96 help, 356 Line2D, 485Quadratic2D, 486, 487 usage, 485 perimeter polygon, 231 triangle, 230 Perimeter2D example, 230 help, 356 Triangle2D, 398

usage, 395 periodic function, 39 perpendicular line, 54, 60, 82 tangent line, 262, 279 Perpendicular2D example, 60, 72, 262, 279 help, 356 Line2D, 463usage, 457pi,  $\pi$ definition, 232 point arc center, 105 arc centroid, 115 arc end point, 105 arc start point, 105 biarc knot point, 314 center, conic, 192 center, quadratic, 192 circle, radical center, 98, 101 collinear, 36, 38, 58 conic center, 184 division point, 33 Gergonne point, 129 knot circle center, 317 midpoint, 33 midpoint of arc, 115 offset, 35 offset along line, 67 orthocenter, 129 overview, 9 polar coordinates, 38 projected on line, 70 reciprocal, 310 shoulder point, 200 tangency, 259, 281 Point2D Arc2D, 391 Circle2D, 408ConicArc2D, 419 construction, 492–494

Coordinates2D, 490 Ellipse2D, 424 example, 30, 33, 35, 36, 67, 70, 71, 117, 121, 151, 185, 197, 263, 268, 270, 274, 275, 277 graphics, 490 help, 356 Hyperbola2D, 449IsValid2D, 490 midpoint, 493, 508 offset, 493 Parabola2D, 482point of division, 493 quadratic center, 494 Quadratic2D, 491 Reflect2D, 492 representation, 490 Rotate2D, 492Scale2D, 492 translate, 492 Triangle2D, 551, 552 usage, 489 validation, 490 XCoordinate2D, 491 YCoordinate2D, 491 Point2D, pole Ellipse2D, 425 Hyperbola2D, 449 Parabola2D, 482Quadratic2D, 494 Points2D circle/circle, 454 curve/curve, 454 example, 92, 94, 190 help, 358 line/circle, 454 line/line, 453usage, 453 polar Circle2D, 408polar (line) definition, 269 polar equation

circle, 101 conic, 192 ellipse, 157 hyperbola, 173 parabola, 144 pole definition, 267 polygon approximating a curve, 231 area, 231 perimeter, 231 polynomial definition, 39 linear, 39 quadratic, 39 Polynomial2D example, 40, 260 help, 358 Line2D, 428 Quadratic2D, 428 usage, 427 PrimaryAngle2D computation, 478 help, 358 usage, 477 PrimaryAngleRange2D Arc2D, 390 computation, 478 help, 358 usage, 477 projective discriminant conic arc, 193, 196 quadratic center point, 184, 192 circle, 178 classification, 184 description, 335 distance to a point, 281 ellipse, 180 hyperbola, 180 linear polynomial, 176 lines, 176, 177 normal, 280, 281

parabola, 178, 179 parametric equations, 192 pencil, 294 reciprocal, 310 rotation, 182, 190, 191 standard conic, 175 tangent, 293, 296, 298, 301 translation, 181 Quadratic2D Circle2D, 405, 409 ConicArc2D, 416 construction, 500-502description, 335 Equation2D, 428 example, 40, 88, 91, 186, 195, 260, 297 FullSimplify, 499 help, 358 IsValid2D, 498 Line2D, polar, 463 normalize, 500 Pencil2D, 486, 487 Point2D, 491 Point2D, pole, 494 Polynomial2D, 428 Reflect2D, 498representation, 497 Rotate2D, 499 Scale2D, 499Simplify, 499 Translate2D, 499 usage, 497 validation, 498 vertex equation, 502query IsApproximate2D, 431 IsComplex2D, 431 IsNegative2D, 434 IsNumeric2D, 432 IsReal2D, 433 IsScalar2D, 433 IsScalarPair2D, 434 IsTinyImaginary2D, 434 IsZero2D, 432, 435

IsZeroOrNegative2D, 435, 436 query, object Is2D, 336 IsDisplay2D, 336 IsValid2D, 336 ObjectNames2D, 336 radical axis circle, 97, 102 radical center circle, 98, 101 radius arc, 105 biarc, radii ratio, 313 circle, 85 Radius2D Arc2D, 390 Circle2D, 407help, 359 usage, 405 rational equations ellipse, 155 hyperbola, 172 rational parameterization circle, 100 ray points arc, 113 reciprocal circle, 310 line, 310 point, 310 quadratic, 310 rectangle area, 237 rectangular coordinates abscissa, 28 ordinate, 28 origin, 28 quadrants, 28 reflect definition, 224 equation, 224 in a point, 226 Reflect2D

Arc2D, 390 Circle2D, 407ConicArc2D, 418 coordinates, 540 Ellipse2D, 424 equation, 540 example, 226 help, 360 Hyperbola2D, 448 Line2D, 461 list of objects, 540 Parabola2D, 481 Point2D, 492 Quadratic2D, 498 Segment2D, 507 Triangle2D, 550 usage, 539 ReflectAngle2D command, 540 help, 360 usage, 539 rho,  $\rho$ conic arc, 193, 196 Rho2D ConicArc2D, 417 help, 360 usage, 415 rotate definition, 219 equation, 220 rotate list of objects, 541 Rotate2D about origin, 541 Arc2D, 390 Circle2D, 407ConicArc2D, 418 coordinates, 541 Ellipse2D, 424 equation, 541 example, 222 help, 360 Hyperbola2D, 448

Line2D, 461 Parabola2D, 482Point2D, 492 Quadratic2D, 499 Segment2D, 507 Triangle2D, 551 usage, 539 scale definition, 222 equation, 222 Scale2D Arc2D, 391 Circle2D, 407 ConicArc2D, 418 coordinates, 542 Ellipse2D, 424 equation, 542example, 223 from origin, 541 help, 361 Hyperbola2D, 449Line2D, 461 list of objects, 542 Parabola2D, 482 Point2D, 492 Quadratic2D, 499 Segment2D, 507 Triangle2D, 551 usage, 539 sector arc, 106 arc, area of, 241 ellipse, area of, 244 hyperbola, area of, 245, 250 SectorArea2D Circle2D, 400 Ellipse2D, 401 example, 242, 245, 246 help, 361 Hyperbola2D, 402 usage, 399 segment arc, area of, 242

ellipse, area of, 243 hyperbola, area of, 245, 250 parabola, area of, 246 Segment2D ArcLength2D, 397 construction, 508 description, 335 evaluation, 505 example, 80, 119 graphics, 506 help, 361 IsValid2D, 506 Length2D, 507 Reflect2D, 507 representation, 505 Rotate2D, 507 Scale2D, 507 Slope2D, 507 Translate2D, 508 Triangle2D, 552usage, 505 validation, 506 SegmentArea2D Circle2D, 400Ellipse2D, 401 example, 244, 246, 247 help, 361 Hyperbola2D, 402Parabola2D, 402usage, 399 semicircle definition, 105 inscribed angle, 101 SemiConjugateAxis2D help, 362 Hyperbola2D, 448 usage, 445SemiMajorAxis2D Ellipse2D, 423 example, 151 help, 362 usage, 421 SemiMinorAxis2D

Ellipse2D, 423 example, 151 help, 362 usage, 421 SemiTransverseAxis2D help, 362 Hyperbola2D, 448 usage, 445SetDisplay2D command, 512 help, 362 usage, 511 Simplify Line2D, 460 Quadratic2D, 499 SimplifyCoefficients2D function, 427 help, 362 usage, 427Sketch2D command, 513 example, 7, 30, 51 help, 362 usage, 511slope line, 53 line segment, 53 Line2D, 462 Slope2D example, 53 help, 363 Line2D, 460 Segment2D, 507 usage, 457Solve2D command, 516 help, 363 usage, 515 SolveTriangle2D example, 126, 127 help, 363 usage, 545span

arc, 105, 232 Span2D Arc2D, 395 ConicArc2D, 396 example, 233 help, 363 usage, 395 square area, 237 Stewart's Theorem triangle, 38 tangent Archimedes' circles, 289 circle, 283, 285-288, 290, 291 conic, 293, 296, 298, 301 overview, 21 quadratic, 293, 296, 298, 301 reciprocal polars, 306 tangent line circle, 255, 280 conic, 266, 273 contact points, 259 definition, 255 ellipse, 274, 281 eyeball theorem, 280 hyperbola, 277, 281 line segment length, 260 Monge's Theorem, 281 parabola, 273, 280 parallel, 261, 279 perpendicular, 262, 279 polar, 269 pole, 267 two circles, 264 two conics, 271 tangent line segment two conics, 272 TangentCircles2D construction, 521, 522 example, 283, 285-288 help, 363 usage, 519 TangentConics2D

construction, 526 example, 297, 299, 301, 303, 305, 306, 308, 309 help, 364 usage, 523 TangentEquation2D equation, 532 example, 269 help, 364 usage, 531 TangentLines2D construction, 532-534example, 256, 259, 262, 265, 272, 274, 275, 277, 279  $\mathrm{help},\,364$ usage, 531 TangentPoints2D construction, 537, 538 example, 259 help, 365 usage, 537 TangentQuadratics2D construction, 526 example, 299 help, 365 usage, 523 TangentSegments2D construction, 534example, 273 help, 365 usage, 531 transformation definition, 217 inversion, 227 reflect, 224 rotate, 219 scale, 222 translate, 217 translate definition, 217 equation, 218 Translate2D Arc2D, 391

Circle2D, 408ConicArc2D, 418 coordinates, 542 Ellipse2D, 424 equation, 543 example, 217, 219 help, 365 Hyperbola2D, 449Line2D, 461 list of objects, 543 Parabola2D, 482Point2D, 492 Quadratic2D, 499 Segment2D, 508 Triangle2D, 551usage, 539 transverse axis hyperbola, 159 triangle altitude, 128-130 angle bisector, 128, 130 area, 237, 238, 249-251 centroid, 120, 129 cevian, 128, 130 circumscribed circle, 122, 130 definition, 117 description, 336 equilateral, 117 Euler's formula, 128 Gergonne point, 129 hypotenuse, 117 inscribed circle, 123, 128, 130 isosceles, 117 Law of Cosines, 126 Law of Sines, 126 median, 120, 128, 130 orthocenter, 129 overview, 16 perimeter, 230 right, 117 scalene, 117 solve, 124Stewart's Theorem, 38

vertex, 117 vertex angle, 117 Triangle2D Angle2D, 547Area2D, 403 Circle2D, 553construction, 548, 553, 554 description, 336 example, 117, 119, 126 graphics, 546  $\mathrm{help},\,365$ IsValid2D, 546 Line2D, 552 Perimeter2D, 398 Point2D, 551, 552 Reflect2D, 550 representation, 546 Rotate2D, 551 Scale2D, 551Segment2D, 552 Translate2D, 551 usage, 545validation, 546 variable definition, 39 dependent, 39 independent, 39 vertex parabola, 135 triangle, 117 vertex equation ellipse, 187 hyperbola, 187 parabola, 186 Quadratic2D, 502vertical line, 61 vertices ellipse, 146 hyperbola, 159 Vertices2D Ellipse2D, 412example, 151

## help, 366 Hyperbola2D, 413Parabola2D, 413 usage, 411 XCoordinate2D coords, 491 help, 366 Point2D, 491usage, 489 YCoordinate2D coords, 491 help, 366 Point2D, 491 usage, 489