Computing bisectors in a dynamic geometry environment

Journal:	*International Journal of Mathematical Education in Science and Technology*
Manuscript ID:	TMES-2011-0323.R1
Manuscript Type:	Classroom Notes
Keywords:	bisector, geometric loci, dynamic geometry
Note: The following files were submitted by the author for peer review, but cannot be converted to PDF. You must view these files (e.g. movies) online.	
Figure1.ggb Figure2.ggb Figure4.ggb Figure8.ggb	

International Journal of Mathematical Education in Science and Technology, Vol. 00, No. 00, 00 Month 200x, 1–10

Classroom note

Computing bisectors in a dynamic geometry environment

Francisco Botana*

(*Received 00 Month 200x; final version received 00 Month 200x*)

In this note an approach combining dynamic geometry and automated deduction techniques is used to study the bisectors between points and curves. Usual teacher constructions for bisectors are discussed, showing that inherent limitations in dynamic geometry software impede their thorough study. We show that the interactive sketching of bisectors and an automatic treatment of the algebraic problem involved can give a reasonable knowledge about them. Since some cases are currently out of computational scope, despite the simplicity of the bisector problem, we sketch an alternative method for dealing with them.

Keywords: bisector; geometric loci; dynamic geometry

1. Introduction

Most of Dynamic Geometry (DG) environments incorporate primitive predicates for computing the perpendicular bisector of two points or a segment, and the parabola defined by a point and its directrix. These cases are particular instances of general bisectors. The bisector of two geometric elements is the locus of a point that moves while remaining equidistant respect to those elements. Since its definition is so simple, it is not surprising that DG users, both teachers and students, frequently deal with bisector constructions. However, most of these constructions are limited to exhibit a graphical trace of the sought locus and there is no algebraic knowledge about them. The aim of this note is to perform a systematic discussion of the construction of bisectors between point and curves in a DG environment. Furthermore, some considerations about automatically obtaining their algebraic characterization are given. We also discuss the differences between answers coming from the DG environment and the symbolic ones.

Section 2 deals with bisectors between point and general lines. We study and propose dynamic constructions for bisectors, and a symbolic method for computing the equations of objects containing the bisector is recalled. Section 3 describes the computation of curve/curve bisectors. Since the approach taken here is based on envelopes, we show that a closed–form answer is returned just for low degree curves. A general symbolic answer for this case is currently out of computational scope, so we sketch a graphical method for computing such bisectors.

Throughout this paper we use GeoGebra as the DG environment which to illustrate our findings. Some of the proposed algorithms rely on the GeoGebra ability to compute a point in a curve which is the closest to a given point, a feature currently not present in other DG systems. Given the actual spread of GeoGebra, we think that this dependence is not a severe limitation and our approach could be easily replicated in other DG software.

*Corresponding author. Email: fbotana@uvigo.es

ISSN: 1045-1129 print/ISSN 1029-0346 online

DOI: 10.1080/1045112YYxxxxxxxx
http://www.informaworld.com

2. Point/curve bisectors

Given a point A and a curve c the construction of the bisector of A and c involves selecting a point B lying in c and tracing the intersection point of the perpendicular bisector of A and B and the normal line to c passing through B. The following algorithm lists the constructive steps for computing bisectors.

Algorithm 1:
Input: curve $c : f(x, y) = 0$, point A
Construct a point B on c
Construct the normal line to c passing through B
Construct the perpendicular bisector of A and B
Intersect both lines, giving a point C
Return: the locus of C when B moves along c

If c is a straight line, Algorithm 1 can be used instead the system–defined Parabola tool. The user will not visually appreciate any difference in the results shown in the Graphics window. Nevertheless, while using the Parabola tool the user obtains a system's primitive object (a parabola whose equation is known, and shown in the Algebra window), no algebraic knowledge is returned when using the above locus approach (see Figure 1. Note that some GeoGebra files illustrating the proposed approach are available as Supplementary Content).

Figure 1. The parabola with focus $A(1, 3)$ and directrix $c : y = 0$ obtained by using the standard Parabola tool (dotted line) and by tracing point C, which is equidistant from A and c.

Suppose that the user wants to compute the bisector of the obtained parabola and its focus. If the parabola was defined as a locus, there is not an easy way to draw its tangent through a point on it, so the general procedure for constructing the new bisector may fail. A user could place five points on the locus and ask for the conic passing through them, thus replacing the parabolic locus by a primitive Implicit Curve element. Nevertheless, the five points are numerically defined, and precision errors will lead to a parabola equation slightly different from the exact one. A simple test confirms they are different: Asking if a point in the locus is in the parabola, the system generally reports that the point does not lie on it. Even worse, we know that this locus is a conic, but for general curves obtained as loci this trick will not be applicable since the curve degree is unknown.

Let us recall a method for automatically computing the algebraic description of such a locus. It uses elimination techniques (see [1] for a detailed description) and is currently adopted by JSXGraph [2] and under consideration for inclusion in GeoGebra. For the sake of illustration, the construction of the parabola, when obtained as a locus, can be parametrically described as follows. Let $A(1, 3)$ be the focus and c the horizontal axis. The point $B(x_1, x_2)$ lies on the line $y = 0$, so $x_2 = 0$. The locus point $C(x_3, x_4)$ is constrained to lie on the line $x = x_1$ and on the perpendicular bisector of A and B (with equation $y - \frac{3}{2} = \frac{x_1-1}{3}(x - \frac{x_1+1}{2})$). Eliminating all symbolic coordinates but those of the locus point in the polynomial system

$$x_2 = 0$$
$$x_3 = x_1$$
$$x_4 - \frac{3}{2} = \frac{x_1 - 1}{3}(x_3 - \frac{x_1 + 1}{2}),$$

we get the expected parabola equation, after replacing C coordinates by generic variables,

$$6y - x^2 + 2x - 10 = 0.$$

Once we have this equation (thus being able to easily compute the normal line in any point on the curve), drawing the bisector of the parabola and its focus A is just reapplying Algorithm 1 with A and the parabola as inputs (Figure 2).

Figure 2. The bisector of A and the parabola $-x^2 + 2x + 6y = 10$.

When doing the GeoGebra construction we get an unknown curve as bisector. It is easy to check that parts of this locus do not satisfy the required equidistance condition. Since the locus is computed for each possible position of a point in the parabola, the returned locus is not the true bisector, but the set of equidistant points from the given point and each point on the parabola. In Figure 2, any point out of the locus loop does not lie on the true bisector, as can be easily checked by drawing the circle centred on it and passing through A. There are points in the parabola closer to A than the locus point, so we have to exclude it from the bisector. See, for instance, Figure 3: Any point in the parabola between D and E is closer to C' than A.

Figure 3. The points out of the bisector loop do not satisfy the condition of equidistance.

Using the above remark, we can trace the true bisector with the following

Algorithm 2:
Input: curve $c : f(x, y) = 0$, point A
Construct a point B on c
Construct the normal line to c passing through B
Construct the perpendicular bisector of A and B
Intersect both lines, giving a point L
Construct the locus *loc* of L when B moves along c
Construct the point D on c closest to L
If $D = B$ then construct the point E, $E = L$.
Return: the locus *truelocus* of E when B moves along c

It must be noted that the construction of D is numerical, so we are introducing some inexactness in the diagram. And testing the equality between points D and B, given their numerical representation, must be replaced by a closeness measure (in empiric tests, we have found that constraining their distance to be less than 0.1 is enough for most constructions).

Using this algorithm the bisector shown in Figure 2 is trimmed. Figure 4 shows the true and untrimmed bisectors of the parabola and a point, for two positions of the point.

Figure 4. The bisectors of the parabola and a point (top $A(4,4)$, bottom $A(6,4)$). Dotted lines are the untrimmed parts of the bisectors.

The computation of the closest point involves inequalities, so a strict algebraic approach to fully determine bisectors is not possible. Nevertheless, we can compute a variety where the bisector lies. Using the above sketched elimination process, we get that the bisector of the parabola and its focus lies on the zero set of the polynomial

$$81x^2 - 4y^3 + 81y^2 - 162x - 486y + 810,$$

thus being a cubic. In order to automate these computations, we have developed a web service at http://193.146.36.205:5467 that uses Sage [3] and a Sage Cell Server [4]. Using this server for bisector computations involves sending a task as follows:

```
load('/home/scs/AD2D4scs.sage')
ACurve('c',[-x^2+2*x+6*y-10])
FreePoint('P',1,3)
Bisector('b','P','c')
```

where the first line loads the appropriate library, the parabola c is defined via its equation, so it is the point, and a new Bisector object b is defined. The system's answer is a line stating the dimension of the bisector and a polynomial list giving the variety that contains the bisector. In this case, the variety is denoted by the list

$$[x^2 + y^2 - 2x - 6y + 10, 81x^2 - 4y^3 + 81y^2 - 162x - 486y + 810].$$

Note that besides the cubic, the elimination also returns a real point (the parabola focus) that also satisfies the algebraic conditions for the bisector, but not being part of the true bisector. The reason for to appear such a spurious solution lies in the elimination process. The calculi take place in the complex field, so for the complex points in the parabola $(1 \pm 3i, 0)$ we get the real locus point $(1, 3)$.

If this extra point shows that the algebraic knowledge must be carefully considered, the Algorithm 2 does not even assure getting the trace of the true bisector. One must remember that geometry and DG are not exactly the same thing. Trying to draw the bisector of the parabola and the parabola point $A(0, 5/3)$ no trace is returned for the true part of the bisector. Nevertheless, the server declares that the bisector lies on the variety

$$[9x - 3y + 5, 2187x^2 - 108y^3 - 486xy + 1503y^2 - 2142x - 6486y + 10735].$$

Again, it is easy to reject the second polynomial as a false part of the true bisector. The reader can check that a GeoGebra construction using Algorithm 1 returns a locus corresponding to this second polynomial, while no visible locus is shown if using Algorithm 2. So, what is the meaning of line $9x-3y+5 = 0$? A simple mental experiment concludes that there are points on the bisector, for instance, A itself. In fact, the bisector is a halfline with origin at A and normal to the parabola. A simple modification of Algorithm 2 could return a trace of the true bisector. Just replacing the first locus computation with the result of the symbolic computation, we have a certified object where the bisector lies.

Algorithm 3:
Input: curve $c : f(x, y) = 0$, point A

Compute, via the server, the untrimmed bisector of A and c
Define this object as an implicit curve d
Construct a point B on d
Construct the point D on c closest to B
If D and A are close, then construct the point E, $E = B$.
Return: the locus *truelocus* of E when B moves along d

This algorithm is more efficient than the previous one for showing true bisectors, since it can detect hidden parts of them. Nevertheless, using a tool external to GeoGebra, we loose interactivity. A smart combination of both algorithms 2 and 3 should be balanced when teaching, at least while this mixed approach is not fully integrated in GeoGebra.

The reason behind the limitation of the Algorithm 2 lies in its use of the Perpendicular Bisector primitive. When using this tool, an implicit restriction is assumed: the segment is well defined, or, in other words, its endpoints are not the same. GeoGebra and other software do not allow to define a perpendicular bisector when both segment endpoints coincide, declaring it as undefined if some operation causes the equality of points. In the bisector we are studying, this undefinition happens when the moving point in the parabola coincides with A, thus preventing the desired computation. Although it is generally meaningful to declare the perpendicular bisector in such degenerated cases as undefined, it should be noted that the bisector of a pair of coincident points is the whole plane, as it can be computed via the server. Sending

```
load('/home/scs/AD2D4scs.sage')
FreePoint('P',1,2)
Bisector('b','P','P')
```

the server returns a bidimensional object defined by the polynomials in [0], that is, all points in the plane are in the bisector. Thus, when computing the bisector of the parabola and a point on it, if the parabola moving point coincides with the given point, the intersection between the normal line and the bisector of the points reduces to the normal line, which is returned as a factor as said above. A quite similar situation occurs when computing the bisector of a straight line and a point lying on the line. If Algorithm 1 is used, no locus is returned, since there are undefined objects (the perpendicular line and the perpendicular bisector are always parallel). Nevertheless, there exists an elementary bisector, as is well known. Computing through the server the bisector of $y = x$ and the point $A(1, 1)$, we get the true bisector as the whole line $x + y - 2 = 0$, while if using the GeoGebra Parabola tool the answer will the double line $(x + y - 2)^2 = 0$.

As a final illustration of the method, we study the bisector of a point A and the cubic $x^3 - 5y^2 = 0$. Apart from knowing the polynomial support of the bisector there are no appreciable differences between using algorithms 2 and 3, when the point lies outside the cubic. Figure 5 shows the bisector for $A(3, 1)$ computed via Algorithm 2.

Figure 5. The bisector of $x^3 - 5y^2 = 0$ and $A(3, 1)$ (the dotted line is the untrimmed part).

The server computation

```
load('/home/scs/AD2D4scs.sage')
ACurve('c',[x^3-5*y^2])
FreePoint('A',3,1)
Bisector('b','A','c')
```

states that the variety containing the bisector is an octic curve, which factors to $3x+y-5$ and a long polynomial $540x^3y^4+\cdots+1445000$ with 30 terms. The linear factor is trimmed (it comes from a degeneracy when trying to compute the tangent to the cubic at its singular point) following the closeness heuristic, thus remaining the graphical trace of the bisector and the knowledge of its polynomial support. Things are different when A lies in the cubic. For $A(5,5)$, Algorithm 2 does not return anything, while Algorithm 3 correctly returns a line as the true bisector (again, this fact is due to a GeoGebra degenerated condition). For $A(0,0)$, we have the contrary situation. Algorithm 3 does not give any sensible information (the server computation returns [0], that is, it concludes that the bisector is contained in the plane!), while Algorithm 2 shows a non empty bisector. Nevertheless, this answer is incomplete, since the negative part of the horizontal axis is also part of the bisector. Figure 6 shows the bisector for $A(0,0)$.

Figure 6. The true bisector of $x^3-5y^2=0$ and $A(0,0)$.

Algorithm 3 fails for $A(0,0)$ returning the bisector just as contained in the plane, since it tries to use the normal line to the cubic at a singular point. Thus, the solution is the whole plane, as it is swept by the bisector of A and a moving point on the cubic. In turn, the horizontal ray that the pure GeoGebra approach ignores is due to the Perpendicular Bisector degeneration when the cubic moving point coincides with $A(0,0)$.

3. Curve/curve bisectors

Despite the subtleties involved in tracing the bisectors of a point and a curve, we did not find hardly any reference about this problem in DG forums. The situation is quite different when dealing with curve/curve bisectors. Circle/circle bisectors have been extensively discussed in this community. For instance, in [5] the participants discuss about the bisectors of two circles. Their constructions are able to draw part of the bisector of external circles, but they face severe issues concerning continuity and degeneracies. Indeed, these ad–hoc constructions can lead to misunderstandings when used for different circle positions, as a false statement in the cited page about the bisector of internally tangent circles. Another contributor [6] gives a more complete solution, and points to a standard proof supporting his statements although restricted to circle/circle bisectors.

We did not find any attempt to construct simpler curve/curve bisectors (e.g. line/circle ones), and, beyond circles, only the ellipse/ellipse bisector has been posed. Regarding this last problem, an interesting discussion (partly in Spanish) is carried out in [7]. Trying to draw these bisectors the contributors employ a closeness approach, and they offer some brute force methods for returning the bisector trace. There, a participant argues against a previous claim declaring this bisector as a conic, stating that "since it seems from the picture that there are inflexion points, we can think on a quartic". We feel that this commentary illustrates the convenience of combining the standard DG approaches with other tools, such as the one reported here.

Our treatment of curve/curve bisectors is inspired in the work in [8] and the automatic methods for computing envelopes described in [9]. It has been proved [8, p. 343] that the bisector of two curves c and d lies in the envelope of the family of bisectors of c and a moving point D in d. Although, as is well known, the envelope can contain spurious factors, our closeness heuristic for trimming the bisectors makes those aberrant factors a matter of lesser significance.

An algorithm to produce the trace of a curve/curve bisector is

Algorithm 1:
Input: a pair of curves $a : f(x, y) = 0, b : g(x, y) = 0$
Compute, via the server, the untrimmed bisector of a and b
Define this object as an implicit curve c
Construct a point C on c
Construct the point A (B) on a (b) closest to C
If the numbers $distance(A, C)$ and $distance(B, C)$ are close, then construct the point D, $D = C$.
Return: the locus *truelocus* of D when C moves along c

The server syntax for computing the untrimmed bisector of a pair of curves is

```
load('/home/scs/AD2D4scs.sage')
ACurve('a',[<polynomial in x and y>])
ACurve('b',[<polynomial in x and y>])
Bisector('bis','a','b')
```

Thus, besides obtaining the trace of the true bisector via the proposed algorithm, the server computation returns a polynomial containing the bisector. This algebraic knowledge is a prerrequisite for deriving new objects with such bisectors as arguments.

Figure 7 shows some bisectors involving lines and conics. From top to bottom, the figure shows the bisector of external (internal) circles with unequal radii, externally (internally) tangent circles, a straight line and a circle, ellipse, hyperbola and parabola. As the previous figures in this paper, the trimmed part of the bisector is shown as a dotted line.

Figure 7. Some curve/curve bisectors.

If a user tries to compute the bisector of two ellipses (or two general conics, even simple ones), he/she will find that the server is not able to find its polynomial support in a reasonable time. The computational cost of evaluating these envelopes is currently out of scope, given the actual algorithms and hardware. Although the problem of tracing the bisector of general plane curves can be numerically solved (see, for instance, a method coming from the community of computer aided geometry design in [10]), here we describe a simpler method based on the standard DG way to suggest envelopes.

As recalled above, the bisector of a pair of curves a and b lies on the envelope of the family of bisectors of a and a moving point on b. Using Algorithm 2 we can trace these bisectors in the DG environment, getting a picture that suggests the envelope. Analogously, tracing the bisector of b and a moving point on a, it is possible to sketch another envelope. Since the bisector of a, b lies in both envelopes, their common part will be the the sought bisector.

Figure 8. An ellipse/ellipse bisector as the common part of envelopes.

Figure 8 shows the bisector of two ellipses as suggested through this graphical procedure. Although this method can be seen as a brute force one, we think that it is more efficient than the ones used in [7]. Nevertheless, it can forget some parts of the true bisector, since it is based on Algorithm 2. For instance, tracing the bisector of two internally tangent ellipses, the ray with origin at the tangency point and with direction the normal line to the ellipses will not be a priori detected.

It is also possible to numerically compute points on the bisector of two general curves. The bisector of a curve and a point is a convex line [8, p. 337]. Thus, intersecting the bisector of a and a point B in b with the normal line to b through B (Figure 9), at most we get two points in the envelope. A closeness measure could decide which of these points lies on the bisector of the curves. Nevertheless, we cannot trace these points in GeoGebra, since it cannot currently compute intersections of a locus and other element. Thus, this approach can only be used to obtain approximations of points lying on the bisector.

Figure 9. A method to graphically determine points (see +) in the bisector.

4. Conclusion

We developed a mixed approach combining a standard DG environment and an algebraic study of bisectors of points and lines. Anecdotal evidence from teachers' constructions of simple bisectors is cited to highlight inherent limitations to this subject under the DG paradigm. We show that elimination techniques are needed for getting more detailed bisectors. Although the complete bisector solution falls out of the algebraic setting, we describe how a numerical approach to trim the bisectors, solved as geometric loci, can give a thorough access to an easy generation of bisectors.

The setup of a remote server for computing geometric loci enables accessibility of the required algebraic knowledge for getting the sough bisectors. Although a price for loosing interactivity must be paid when using the server, the algebraic description of loci is a necessary step for returning certified answers. An automatic connection between GeoGebra and the remote server is ongoing work. Since determining the algebraic description of some curve/curve bisectors is computationally out of scope, a graphical method has been proposed to obtain the trace of such bisectors.

Acknowledgement

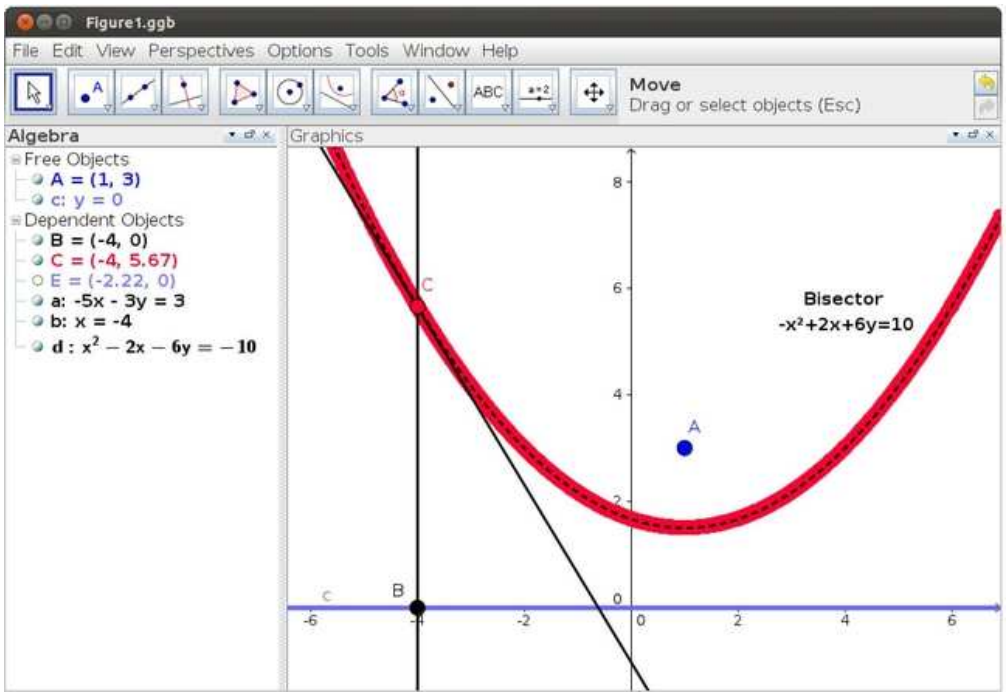
I thank the referees for all their valuable remarks and suggestions. This work has been supported by grants MTM2008–04699–C03–03/MTM and MTM2011-25816-C02-02 from the Spanish MINECO.

References

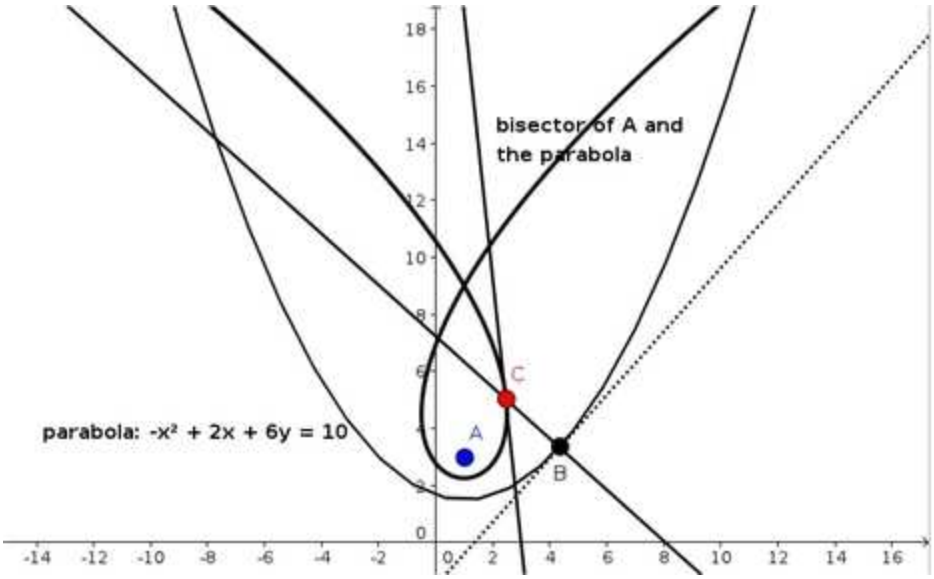
[1] F. Botana and J.L. Valcarce, *A software tool for the investigation of plane loci*, Math. Comput. Simulation 61 (2003), pp. 139–152.
[2] A. Wassermann, *JSXGraph (Dynamic Mathematics with JavaScript)*; software available at http://jsxgraph.uni-bayreuth.de.
[3] W.A. Stein, *Sage Mathematics Software*; software available at http://sagemath.org.
[4] J. Grout, *Sage Cell Server*; software available at https://github.com/sagemath/sagecell.
[5] *The locus of points equidistant to two circles*; available at http://www.geogebra.org/forum/viewtopic.php?f=22&t=23090.
[6] *locus of points equidistant from two circles*; available at http://www.geogebra.org/forum/viewtopic.php?f=2&t=23948.
[7] *Bisector of two ellypses explained*; available at http://www.geogebra.org/forum/viewtopic.php?f=11&t=21842.
[8] R.T. Farouki and J.K. Johnstone, *Computing point/curve and curve/curve bisectors*, in *Design and Application of Curves and Surfaces (The Mathematics of Surfaces V)*, ed. R.B. Fisher, Oxford Univ. Press, 1994, pp. 327–354.
[9] F. Botana and J.L. Valcarce, *Automatic determination of envelopes and other derived curves within a graphic environment*, Math. Comput. Simulation 67 (2004), pp. 3–13.

[10] S.L. Omirou and G.A. Demosthenous, *A new interpolation algorithm for tracing planar equidistant curves*, Robot Comput. Integrated Manuf. 22 (2006), pp. 306–314.

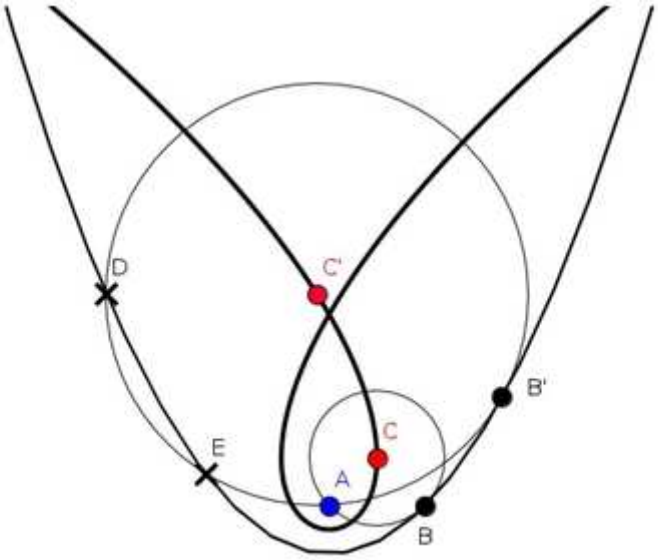
List of figures



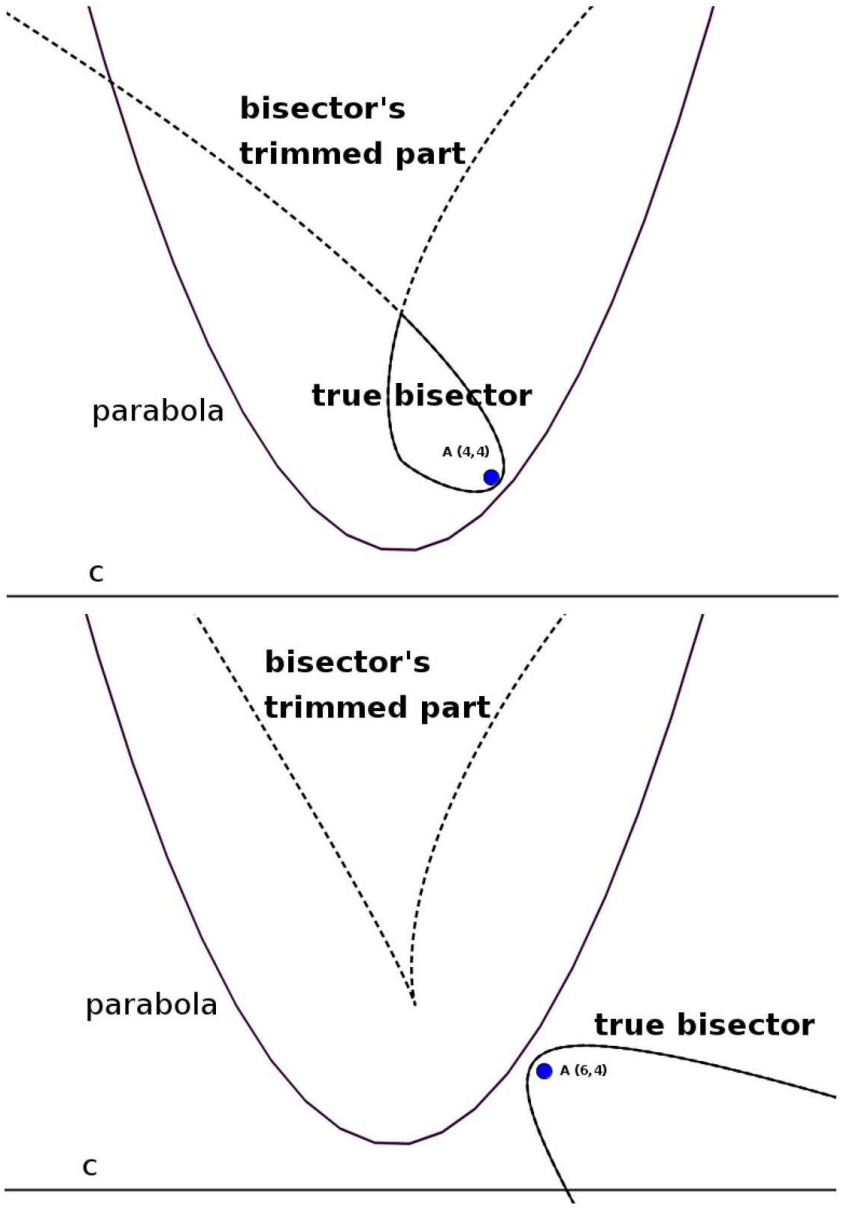
The parabola with focus $A(1,3)$ and directrix $c:y=0$ obtained by using the standard Parabola tool (dotted line) and by tracing point C, which is
equidistant from A and c.
54x37mm (300 x 300 DPI)



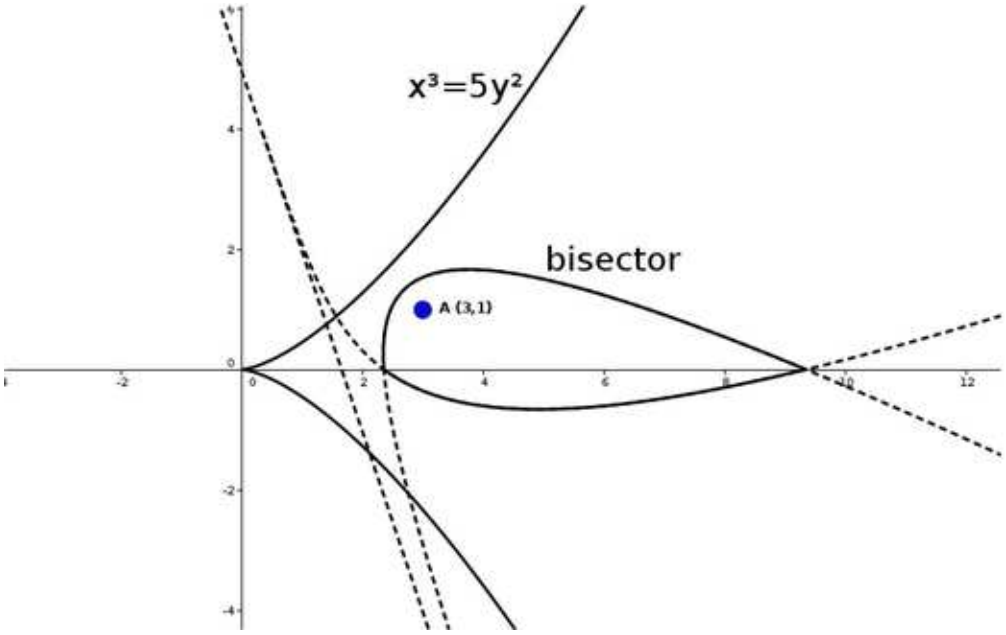
The bisector of A and the parabola $-x^2+2x+6y=10$.
39x24mm (300 x 300 DPI)



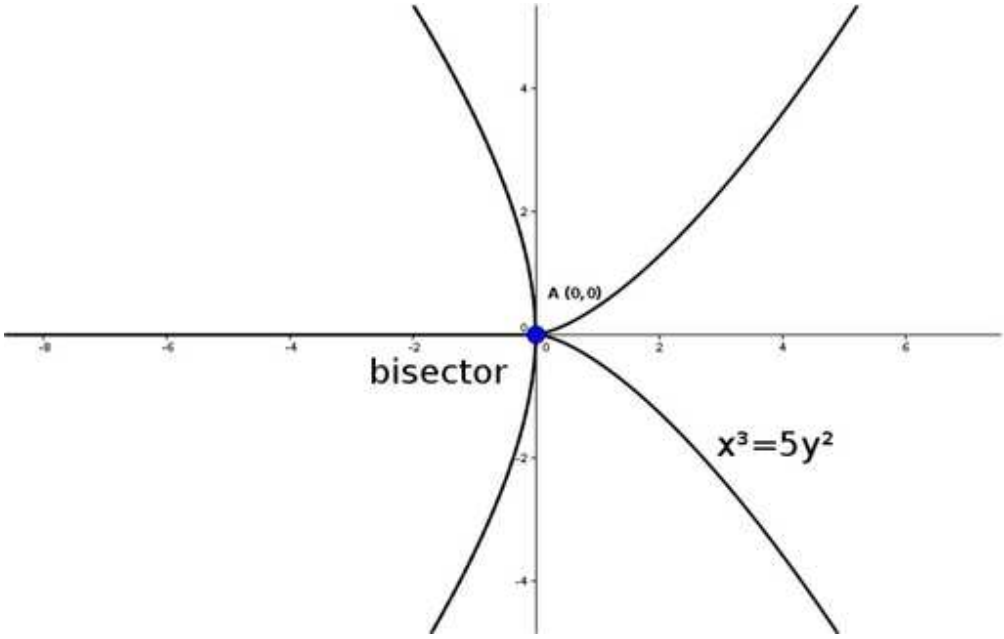
The points out of the bisector loop do not satisfy the condition of equidistance.
37x25mm (300 x 300 DPI)



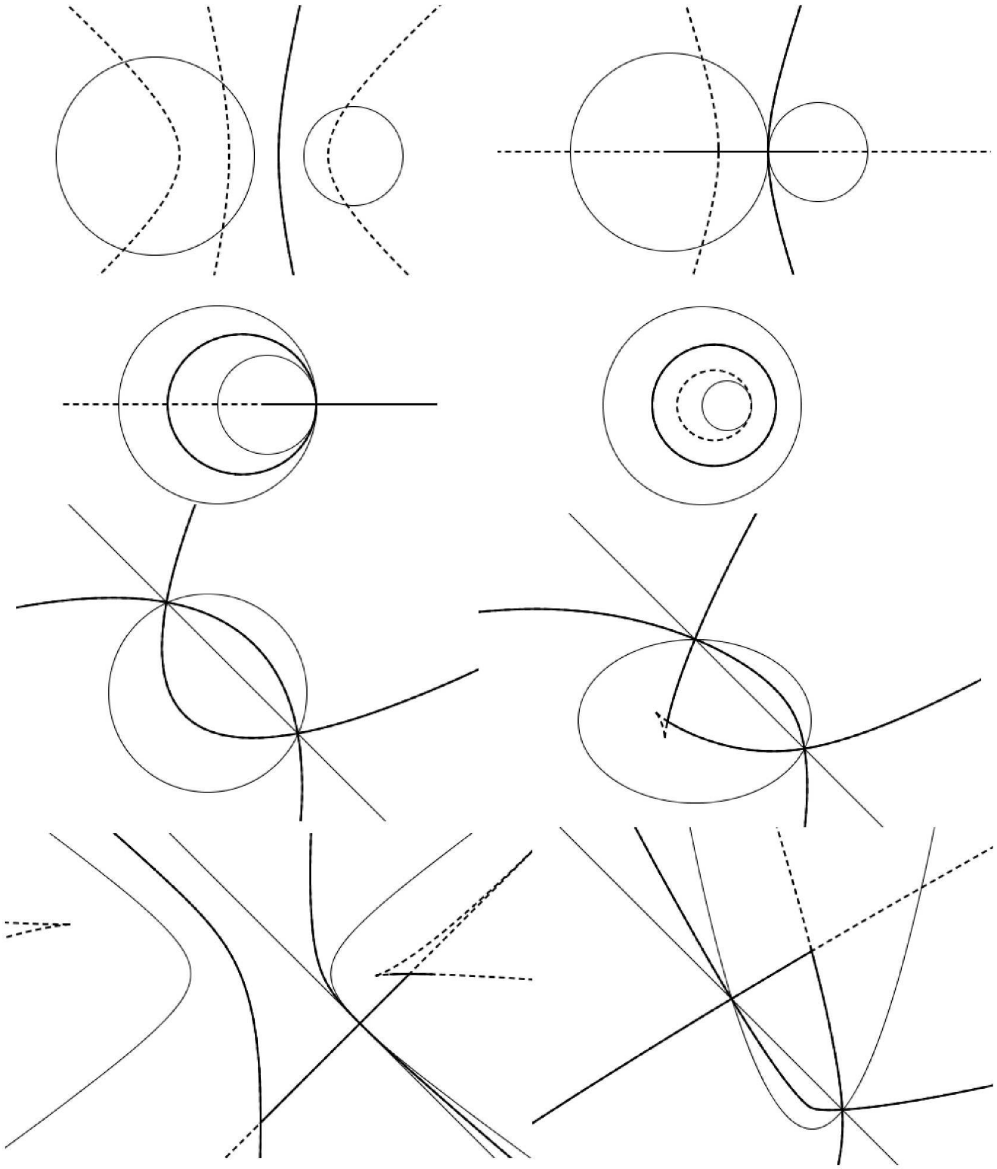
The bisectors of the parabola and a point (top $A(4,4)$, bottom $A(6,4)$). Dotted lines are the untrimmed parts of the bisectors.
94x135mm (300 x 300 DPI)



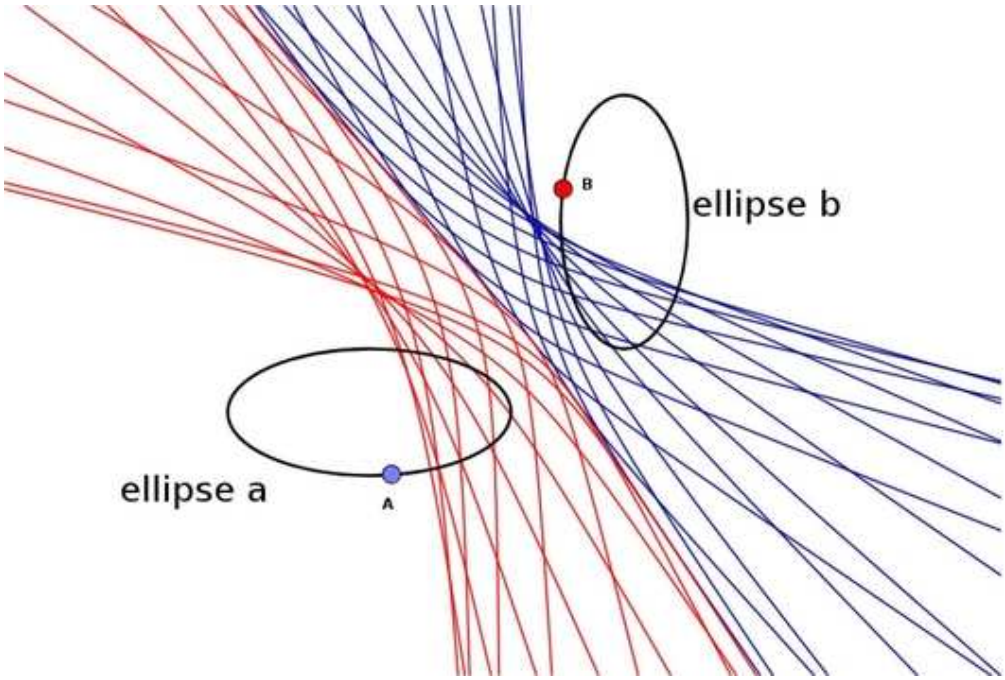
The bisector of $x^3-5y^2=0$ and $A(3,1)$ (the dotted line is the untrimmed part).
44x27mm (300 x 300 DPI)



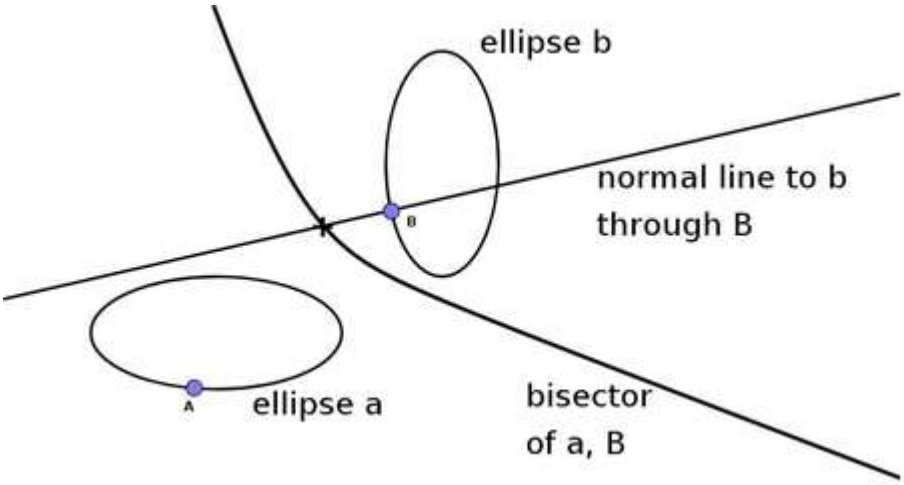
The true bisector of $x^3-5y^2=0$ and $A(0,0)$.
43x27mm (300 x 300 DPI)



Some curve/curve bisectors.
427x516mm (300 x 300 DPI)



An ellipse/ellipse bisector as the common part of envelopes.
44x30mm (300 x 300 DPI)



A method to graphically determine points (see $+$) in the bisector.
38x21mm (300 x 300 DPI)