On the Parametric Representation of Dynamic Geometry Constructions

Francisco Botana

Departamento de Matemática Aplicada I, Universidad de Vigo, Campus A Xunqueira, 36005 Pontevedra, Spain, fbotana@uvigo.es

Abstract. This paper describes an ongoing implementation of an open source library dealing with parametric representation of dynamic geometry constructions. We show how some current issues in standard dynamic geometry environments, such as computing envelopes of lines which are not primitive objects known by the geometric system, can be efficiently solved using the correspondence between geometry and algebra. We also propose enriching the *tool* (or *macro*) mechanism, available in some environments, with our parametric approach. Finally, some problems arising from the algebraic method considered are also studied.

Keywords: dynamic geometry, parametric representation, automatic discovery, derived curves, macroconstructions

1 Introduction

The development, in the late eighties, of The Geometer's Sketchpad [1] and Cabri [2] marked the birth of the dynamic geometry (DG) paradigm. Since then, a myriad of dynamic geometry environments have been released (too many to be listed here, see [3]) and, some of which have been primarily used in secondary mathematical education.

Almost simultaneously, promising results were obtained in automatic reasoning in elementary geometry driven by a new class of algebraic approaches, mainly the Groebner bases method [4] and the one due to Wu [5]. Intense theoretical research on both methods was conducted (see, for instance, [6–9]), and software packages, mostly using Wu's method, were made public [8, 10, 11]. A further development of the automatic theorem–proving method proposed by Kapur was described in [12]. It concluded aiming for a deeper interleaving of dynamic geometry and automated discovery paradigms. Such a wish was partialy fulfilled by linking The Geometer's Sketchpad with a Maple library for parametric description of constructions in [13] using Wu's method, and with Groebner bases in GDI [14, 15], a DG prototype using Mathematica and CoCoA [16] as back–end symbolic engines.

In this paper we continue pursuing the above stated goal. We describe a preliminary implementation of an open source library designed to extend current abilities of standard DG environments. Section 2 summarizes the state of the art in DG on deriving new curves and some of our previous findings on this subject.

A problem concerning the generation of the envelopes of families of general lines is posed, and the different answers provided by some well known DG systems are given. In Section 3 we introduce the open source library. The rationale for using free tools is discussed and we show how to use the library for solving some problems related to curve deriving. Some problems concerning mismatchs in the correspondence between algebra and geometry are also discussed. The paper concludes pointing out the next items in our work of enhancing the DG paradigm with more powerful symbolic tools.

2 Curve Derivation: State of the Art

An informal definition of the DG paradigm is that unconstrained parts of the constructions can be moved and, as they are, all other elements automatically self–adjust, preserving dependence relationships and constraints [17]. An immediate consequence of this behavior is that it allows us to visualize the path of an object that depends on another object while this one is dragged. If the dependent object is a point, in general its trace provides a locus, whereas if it has higher dimension, the path can be used to suggest related geometric elements, such as envelopes. Loci generation has been listed as one of the five properties needed by a geometry system to be considered dynamic [18].

2.1 Finding Loci and Envelopes in Dynamic Geometry Environments

Most dynamic geometry systems implement loci generation using what we have called an *interactive approach* [19]. The basic strategy is simple: in order to compute the locus of a point depending somehow on another, that lies on a predefined path, one just needs to sample this path and register the position of the locus point for each member of the sample. The list of these positions constitutes a subset of the locus. Usually, DG systems will join, using some ad–hoc heuristics, the points on contiguous positions, returning an object similar to the other basic ones in the construction. The main exception to this joining approach is Geometry Expert [10]. So, the loci obtained in the way sketched above are just a collection of screen pixels, the system has no algebraic information about them, and sometimes they behave aberrantly when plotting them due to continuity issues (see [15] for this problem).

This loci–computing strategy is also used for dealing with envelopes of families of lines in standard DG systems. The Geometer's Sketchpad and Geometry Expert suggest envelopes by plotting a collection of family lines, while Cabri and Cinderella [20] share this approach but are sometimes able to return them as a line in the case of simple families. It must be noted that Cabri claims that "for a locus, the algorithm produces its algebraic equation if its degree is no greater than 6. For loci whose points are of very different magnitudes, numerical errors appear very rapidly as the degree increases" [21]. Although this algorithm has not been made public by the vendor, it seems that it is very unstable, and the

returned results are frequently erroneous even for simple cases [22]. For instance, computing an astroid as the envelope of a moving segment with each end on one of a pair of perpendicular axes, Cabri gives different equations when constraining the segment to the upper and lower half planes, as shown in Fig. 1. Furthermore, the degree of the equations cannot be explained as a rounding error, but a wrong algorithmic approach to finding the real sextic.

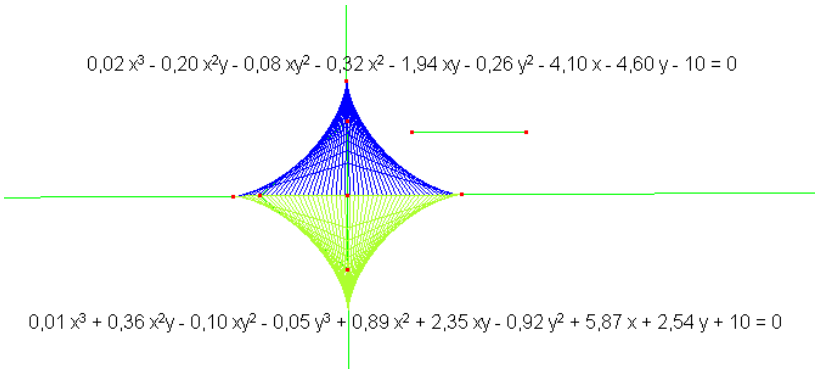

Fig. 1. Equations returned by Cabri for the upper and lower halves of an astroid.

2.2 A Symbolic Approach to Curve Derivation

The application of symbolic methods for loci generation, although restricted to algebraic curves, generalizes the class of obtainable loci, returns their algebraic expressions, and behaves in a uniform way for all construction instances. Since Recio and Vélez did not deal with loci in [12], we proposed in [23] a simple extension of their automated discovery proposal and showed that it can efficiently be implemented in a DG environment. This extension has also been added to JSXGraph [24], a library for interactive geometry, and its incorporation into GeoGebra [25] is currently under development [26].

A similar strategy was used in [27] for symbolic computation of envelopes and other derived curves. Since the implementation partially used proprietary software (Mathematica) and the DG prototype just worked under Windows, we decided to rewrite the algorithms as an open source library and to develop it as an add–on for standard DG environments.

2.3 Computing Envelopes of Geometric Loci

We describe in this subsection a problem currently unsolvable in DG systems. It deals with deriving objects from non basic objects. As said above, most systems are not able to compute the equations of loci or envelopes. Consider, for instance, an offset curve of a parabola, that is, the envelope of a family of equal radius

circles centered at a point lying on the parabola. While standard DG systems show this envelope by plotting a reduced list of such circles, GDI returns it as a simple line and also provides its equation. Figure 2 illustrates these offsets in Cinderella (left) and GDI (right).

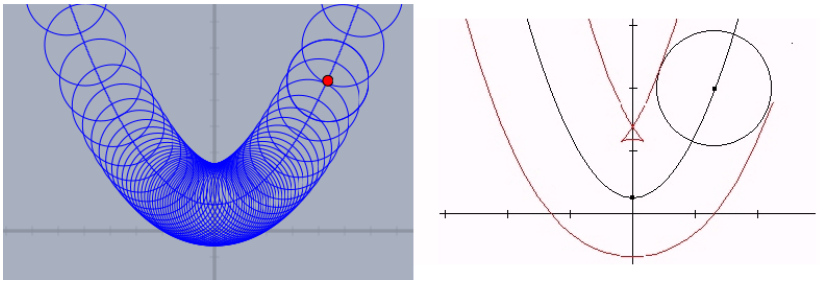

Fig. 2. The offset of a parabola as plotted by Cinderella (*left*) and GDI (*right*).

Nevertheless, plotting some family members for visualizing the envelope only works if the lines are basic objects in the environment. Replacing the circle moving along the parabola by another circle, obtained as a locus, no standard system, as far as we know, can compute the envelope. There is just a solution: moving the circle, with its trace activated. But note that not all systems can trace loci. For instance, current version of GeoGebra cannot trace them (although GeoGebra 4.0 will). Besides that, using the trace option is hard in order to get a descriptive picture of the envelope.

Consider an ellipse built as a locus (following the gardener's method) and let A, B be its foci, where A is a fixed point and B lies on a line. Figure 3 shows the envelope of these ellipses, when B moves along its parent line, in Cinderella (left) and GeoGebra 4.0 Beta (right).

In the above situation, GDI would also fail, since it could not return this ellipse as a locus. Note that the ellipse we are trying to build is the locus of all points X in the plane such that $\text{distance}(A, X) + \text{distance}(B, X)$ is a constant. Being B a semifree point, the algebraic answer should be, as GDI returns, the whole plane (or a bidimensional subset). And asking for the envelope of non linear elements is forbidden in GDI.

In order to overcome the above situation, we roughly proceed as follows. We compute the ellipse as a locus in a 4–dimensional space (where two variables are the locus ones, and the other pair comes from B), and project it over the space of the first pair of variables (see next Section for a more detailed description).

3 The Open Source Library

There are two approaches for extending DG systems with new symbolic, algebraic related, abilities. The first one consists of incorporating the algorithms in

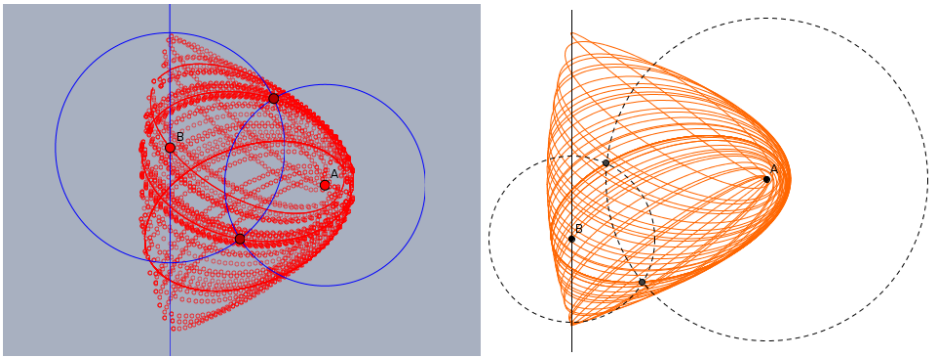

Fig. 3. The traces of a moving ellipse obtained as locus in Cinderella (*left*) and GeoGebra 4.0 Beta (*right*).

the heart of the system, while the second one uses preexistent software (mainly computer algebra systems, CAS) and connects them somehow with the DG environment. Apart from historical reasons concerning the places where Wu and Groebner methods first appeared, we have no doubt about the importance of the developers milieu when making such a decision. So, Chinese systems mainly use the first approach, while academic proposals coming from the occidental world use external CAS, since their cost is a minor point when distributing the system. Paradigmatic examples are Geometry Expert and Geometry Expressions [28]. Nevertheless, the global systemic crisis, with its present and upcoming cuts in educational and non–profit research budgets, the globalization of the information and some centrifugal tendencies in academy, help to explain a renewed interest in providing free access to DG systems and related tools. The growing use of GeoGebra and its probably settlement as the de facto standard in secondary mathematical education is a vivid example of it. Apart from its free character, the open source model followed by GeoGebra also helps explain its success, since it involves a bigger part of the educational community and reacts faster to user requirements and updates than other proprietary DG software does. Same reasons apply to a recent CAS, Sage [29], a free open source mathematics software system licensed under the GPL, whose declared mission is "*creating a viable free open source alternative to Magma, Maple, Mathematica and Matlab*". So, using Sage as development platform, we decided to rewrite from scratch our previous algorithms related to automatic discovery in geometry.

3.1 The Structure of the Library: Examples

Currently, there are about a dozen of basic geometric predicates, and some internal functions, needed for internal work or consistency checkings. The interested reader can download the library as a Sage worksheet or text from [30]. The constructive predicates are: `FreePoint`, `Line`, `Circle`, `MidPoint`, `PointOnObject`, `ParallelLine`, `PerpendicularLine`, `TangentLine`, `IntersectionObjectObject`,

`Locus`, `Locus2` and `Envelope`. When invoked, each predicate adds to a dictionary, called `Todo`, the corresponding geometric element together with some relevant information. The elementary action of adding a point, for instance, is performed through the function

```
def FreePoint(pnt,absc,orde):
    """Adds the point $pnt$ with coordinates $(absc,orde)$
    to the geometric construction."""
    Todo.update({pnt:{'coords':(absc,orde),
                      'parents':Set([]),
                      'type':'FreePoint',
                      'hist':['FreePoint',pnt,absc,orde],
                      'eq':Set([])}})
```

where partial indenting has been done for legilibility. So, the evaluation of the command `FreePoint('P',2,-1)` defines the point $P(2,-1)$, with type `'FreePoint'`, and without parents or equation, being the remaining keyword for future use.

Defining the midpoint of a pair of points is done through the function

```
def MidPoint(n,p,q):
    """Constructs the midpoint $n$ of points $p$ and $q$."""
    if n in Todo.keys():
        eq=Todo[n]['eq']
        parents=Todo[n]['parents']
        temp=Todo[n]['coords']
    else:
        eq=Set([])
        parents=Set([])
        temp=(BoVar.pop(),BoVar.pop())
    eq=eq.union(Set([temp[0]-1/2*(x(p)+x(q)),
                     temp[1]-1/2*(y(p)+y(q))]))
    parents=parents.union(Set([p,q]))
    Todo.update({n:{'coords':temp,
                    'type':'BoundedPoint',
                    'eq':eq,
                    'parents':parents,
                    'hist':['MidPoint',n,p,q]}})
```

where it should be noted that multiple definition for points is allowed. This function also introduces the second type for 0–dimensional objects, `BoundedPoint`. There are other two general types for objects in the library, `Line`, for 1–dimensional objects, and `Plane`, for any geometric object with dimension 2.

3.2 Finding Envelopes of Loci

Recalling the unsolved envelope problem in 2.3 we define the ellipse as follows:

```
FreePoint('A',4,0)
FreePoint('P1',0,0)
FreePoint('P2',0,1)
Line('y','P1','P2')
PointOnObject('B','y')
FreePoint('M',2,2)
FreePoint('N',2,7)
Line('MN','M','N')
PointOnObject('P','MN')
Circle('c1','A','M','P')
Circle('c2','B','N','P')
IntersectionObjectObject('X','c1','c2')
Locus2('loc','X','B','P')
```

Asking for the definition of the special object `loc` we get

```
{'type': 'Locus2',
 'tracer': 'X',
 'hist': ['Locus2', 'loc', 'X', 'B', 'P'],
 'parents': {'X', 'P'},
 'mover': 'P',
 'eq': {x1, 4*orde^2*x2^2 - 4*orde*x2^3 - 36*absc^2 - 100*orde^2
 + x2^4 - 32*absc*orde*x2 + 16*absc*x2^2 + 164*orde*x2 - 82*x2^2
 + 144*absc + 81},
 'implicit': False}
```

where the locus is the zero set of two polynomials in $\mathbb{Q}[absc, orde, x_1, x_2]$. The evaluation of `Envelope('env','loc','B')` just carries out the elementary computation for plane envelopes, returning

```
{'type': 'Line',
 'hist': ['Envelope', 'env', 'loc', 'B'],
 'parents': {'loc', 'B'},
 'mover': 'B',
 'eq': {absc^2*orde^4 + orde^6 - 16*absc^3*orde^2 - 24*absc*orde^4
 - 36*absc^4 + 74*absc^2*orde^2 - 2*orde^4 + 432*absc^3
 + 32*absc*orde^2 - 1647*absc^2 - 207*orde^2 + 1656*absc + 1296},
 'tracer': 'loc'}
```

Since the library provides the equation of the envelope, plotting it in a DG environment would be a simple matter. Figure 4 shows a plot of this equation. Nevertheless, special care must be taken here. Factoring the polynomial we get the shown parabolas, $-orde^2 + 18 * absc + 9 = 0$ and $orde^2 + 2 * absc - 9 = 0$, plus an extraneous factor $-absc^2 - orde^2 + 8 * absc - 16$, that is, the focus A. The problem of such extraneous factors deserves special consideration and its relation with the application of this library to DG environments will be discussed in a future note.

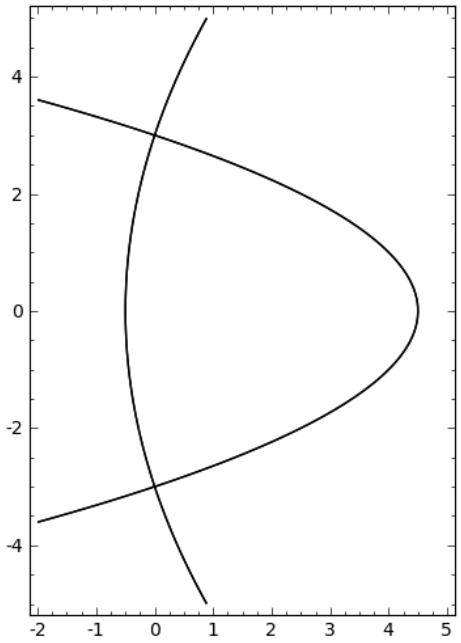

Fig. 4. A Sage plot of the moving ellipse envelope.

3.3 Macro Definitions

Even with the small set of library basic functions listed in 3.1, new curves can be easily derived following an approach that resembles the well known mechanism of macros or tools in standard DG environments. Consider computing the pedal curve of a line with respect to a given point. The following function solves the problem:

```
def Pedal(n,l,p):
    """Computes the pedal line $n$ of line $l$ with respect to
    point $p$."""
    TangentLine('tan',l,'ptemp')
    PerpendicularLine('perp',p,'tan')
    IntersectionObjectObject('x','tan','perp')
    Locus(n,'x','ptemp')
```

If the line `l` is the ellipse with foci in $(0,0)$ and $(4,0)$, and passing through $(9/2,1/2)$ (so having as equation $36x^2+100y^2-144x-81=0$), its pedal curve with respect to the point $(2,0)$, computed with the above function, is the quartic $4x^4+8x^2y^2+4y^4-32x^3-32xy^2+71x^2+23y^2-28x-36=0$.

The above protocol would be natural in any DG system, assuming the system can compute tangent lines. If using GeoGebra, the procedure can be easily defined as a tool, since all involved operations are defined. We propose enhancing

the definition of tools with our library. It must be noted that this connection will require fine tuning. For instance, GeoGebra (and many other DG systems) deal with straight lines, conics and greater degree curves as essentially different objects, while they are just `Line` objects in the library. Defining a parent class for 1–dimensional objects would solve this point.

Another illustration of this technique is the computation of catacaustics, the reflective case of caustics. These caustics envelope a family of reflected rays, so, given a fixed point p and a line l, the procedure

```
def Caustic(n,l,p):
    """Computes the catacaustic $n$ of line $l$ with radiant $p$."""
    TangentLine('tan',l,'ptemp')
    PerpendicularLine('perp','ptemp','tan')
    Symmetrical('q',p,'perp')
    Line('ref_ray','q','ptemp')
    Envelope(n,'ref_ray','ptemp')
```

returns the caustic.

3.4 Using the Library: Caveat Emptor!

Some considerations must be made about using the library. The first one is an advice for non expert users when mimicking the macro approach described above. Both procedures use some intermediate objects that share their names (`tan`, `perp`, `ptemp`,...) and, since accessing elements is done by name, a previously constructed element could be used. Multiple constraining is allowed for some elements (`BoundedPoint`, for example). So, a careless user can introduce undesired constraints for construction elements. The library code contains two alternative definitions for computing pedals and caustics, with appropriate names for intermediate objects and where the dictionary values of the final curves contain more specific information about them.

The second consideration involves the relation between varieties and ideals, so allowing a correspondence between geometry and algebra. The problem of extraneous factors, illustrated in Section 3.2, can be seen as one of an algorithmic nature. It is planned that new library versions will add alternative algorithms (different types of resultants) and heuristic strategies for dealing with it. Nevertheless, there is another source or imprecision, coming from the elimination approach taken in the library. Although algebraic elimination has been proved as succesfull when introducing automated deduction in DG systems, its findings must be critically examined in this environment. For instance, once computed the pedal in Section 3.3, one should note that the zero set of the polynomial wrongly includes the pedal point. So, we cannot rely on the pedal object for a posteriori computations. This behavior can be explained as follows: when we eliminate variables, we do not get just a projection, but its Zariski closure. That is, there can exist spurious points. As GDI warns, when finding a locus, "*The locus is (or is contained in)...*". There is an ongoing theoretical work on this subject. Future versions of the library will incorporate these new developments.

4 Conclusions and Future Work

We report a free open source implementation of a Sage based library for computations related to plane parametric geometric contructions. The library solves the problem of finding the algebraic description of objects in a virtual ruler and compass environment. Although it is restricted to a purely algebraic realm and it does not give a complete solution to the translation between geometry and algebra (if there is one!), it is an efficient solver for handy computations and can be easily integrated as an add–on to general dynamic geometry environments. Future work concerning library development involves automatic theorem proving and discovery.

Acknowledgements

The author was partially supported by research grant MTM2008-04699-C03-03/MTM from the Spanish MICINN.

References

1. Jackiw, N.: The Geometer's Sketchpad v 4.0. Key Curriculum Press, Berkeley (2002)
2. Laborde, J.M., Bellemain, F.: Cabri Geometry II. Texas Instruments, Dallas (1998)
3. List of interactive geometry software, `http://en.wikipedia.org/wiki/List_of_interactive_geometry_software`
4. Buchberger, B.: Groebner Bases: An Algorithmic Method in Polynomial Ideal Theory. In: Bose, N.K. (ed.) Multidimensional Systems Theory, pp. 184–231. Reidel, Dordrecht (1985)
5. Wu, W.T: Mechanical Theorem Proving in Geometries. Springer, Vienna (1994)
6. Kapur, D.: A Refutational Approach to Geometry Theorem Proving. Artif. Intell. 37, 61–94 (1988)
7. Kapur, D.: Using Groebner Bases to Reason about Geometry Problems. J. Symb. Comput. 2, 399–408 (1986)
8. Chou, S.C.: Mechanical Geometry Theorem Proving. Reidel, Dordrecht (1988)
9. Chou, S.C.: Proving Elementary Geometry Theorems Using Wu's Algorithm. In Bledsoe, W.W, Loveland, D.W. (eds.) Automated Theorem Proving: After 25 years. Contemporary Mathematics vol. 29, pp 243–286. AMS, Providence (1984)
10. Gao, X.S., Zhang, J.Z., Chou, S.C.: Geometry Expert. Nine Chapters, Taiwan (1998)
11. Wang, D.: GEOTHER: A Geometry Theorem Prover. In McRobbie, M.A., Slaney, J.K. (eds.) CADE–13. LNAI, vol 1104, pp. 166–170. Springer, Heidelberg (1996)
12. Recio, T., Vélez, M.P.: Automatic Discovery of Theorems in Elementary Geometry. J. Autom. Reasoning 23, 63–82 (1999)
13. Roanes–Lozano, E., Roanes–Macías, E., Villar, M.: A Bridge between Dynamic Geometry and Computer Algebra. Math. Comput. Model. 37(9–10), 1005–1028 (2003)
14. Botana, F., Valcarce, J.L.: A Dynamic–Symbolic Interface for Geometric Theorem Discovery. Comput. Educ. 38(1–3), 21–35 (2002)

15. Botana, F., Recio, T.: Towards Solving the Dynamic Geometry Bottleneck via a Symbolic Approach. In Hong, H., Wang, D. (eds.) ADG 2004. LNAI, vol. 3763, pp. 92–110. Springer, Heidelberg (2006)
16. Capani, A., Niesi, G., Robbiano, L.: CoCoA, a System for Doing Computations in Commutative Algebra, http://cocoa.dima.unige.it
17. King, J., Schattschneider, D.: Geometry Turned On. MAA, Washington (1997)
18. Gao, X.S.: Automated Geometry Diagram Construction and Engineering Geometry. In Gao, X.S. Wang, D., Yang, L. (eds.) ADG 1998. LNAI, vol. 1669, pp. 232–257. Springer, Heidelberg (1999)
19. Botana, F.: Interactive versus Symbolic Approaches to Plane Loci Generation in Dynamic Geometry Environments. In Sloot, P.M.A., Tan, C.J.K., Dongarra, J.J., Hoekstra, A.G. (eds.) ICCS 2002. LNCS, vol. 2657, pp. 801–810. Springer, Heidelberg (2003)
20. Richter–Gebert, J., Kortenkamp, U.: The Interactive Geometry Software Cinderella. Springer, Berlin (1999)
21. http://download.cabri.com/data/pdfs/manuals/cabri2plus140/Man_uk_PDF3.pdf
22. Botana, F., Abánades M., Escribano, J.: Computing Locus Equations for Standard Dynamic Geometry Environments. In Shi, Y., van Albada, G.D., Dongarra, J.J. (eds.) ICCS 2007. LNCS, vol. 4488, pp. 227–234. Springer, Heidelberg (2007)
23. Botana, F., Valcarce, J.L.: A Software Tool for the Investigation of Plane Loci. Math. Comput. Simul. 61(2), 139–152 (2003)
24. JSXGraph, http://jsxgraph.uni-bayreuth.de
25. GeoGebra, http://www.geogebra.at
26. GeoGebra Locus Line Equation, http://www.geogebra.org/trac/wiki/LocusLineEquation
27. Botana, F., Valcarce, J.L.: Automatic Determination of Envelopes and Other Derived Curves within a Graphic Environment. Math. Comput. Simul. 67(1–2), 3–13 (2004)
28. Geometry Expressions, http://www.geometryexpressions.com
29. Stein, W.A. et al.: Sage Mathematics Software (Version 4.6.0), The Sage Development Team, 2010, http://www.sagemath.org
30. Automatic Discovery Sage Library http://webs.uvigo.es/fbotana/AutDiscLib.{sws.txt}