Interfacing Free Computer Algebra Systems and C with KETCindy

Setsuo Takato [1] and José A Vallejo [2]

[1] Toho University
2-2-1 Miyama, Funabashi, Chiba, Japan
takato@phar.toho-u.ac.jp

[2] Universidad Autónoma de San Luis Potosí
Lat Av Salvador Nava s/n, San Luis Potosí, México
jvallejo@fc.uaslp.mx

Abstract. *LaTeX is the* de facto *standard for scientific communication, particularly in Mathematics or Physics. We have developed KETpic and KETCindy to make it easier to produce, embed and organize technical graphics in a LaTeX document. KETCindy involves an interaction between KETpic and Cinderella, a Dynamic Geometry Software (DGS). The present paper illustrates its basic usage along with a detailed treatment of its capabilities for interfacing with free Computer Algebra Systems (CASs) and the programming language C.*

1 Introduction

Many researchers and educators at different levels (ranging from high schools to universities) use LaTeX to write research papers or to produce teaching materials. Unquestionably, LaTeX is the most powerful tool to typeset complex mathematical formulas, and is the *de facto* standard for scientific communication, particularly in the fields of Mathematics or Physics. However, it presents some challenges for the non-expert when it comes to producing, embedding and arranging graphics. This is something notorious in the case of teaching materials. The needs of a teacher are typically different from those of a researcher, and sometimes the kind of graphics required to illustrate a classroom explanation are quite difficult to get using standard LaTeX packages, but very easy to obtain using, say, a DGS. Hence we developed KETCindy, which is a macro package capable of connecting to mathematical software such as Scilab or **R** to produce graphical elements and LaTeX style files to arrange components such as figures, sentences and symbols, at any place on the page of a LaTeX document. The package KETCindy is based on a previous one called KETpic. It is comparatively easier to produce LaTeX graphics with KETpic than with packages like TikZ (although there is always a subjective component in such statements), but a severe weakness of KETpic was the absence of a GUI (Graphical User Interface). To remedy this, attention was turned to Cinderella, a DGS developed by Gebert and Kortenkamp. After a fruitful collaboration with Kortenkamp, it was possible to connect KETpic and Cinderella, and the first version of KETCindy was released in September, 2014. Currently, Cinderella works as a GUI for KETCindy[1][2].

The first part of this paper is intended as an introduction of KETCindy, including the installation process and its basic usage. Then we move on to more advanced applications reflecting the recently added capabilities for calling some free CASs and the programming language C from KETCindy. We offer some samples related to two topics at a university course level: The calculus of variations (the brachistochrone) and dynamical systems (Poincaré sections for Hamiltonian systems).

2 How to use KETCindy

We show here how to use KETCindy to produce graphics that can be inserted in a LaTeX document, such a a research paper or some handout to be delivered at the classroom. We assume that the user's system has already installed a complete LaTeX distribution, and a `pdf` viewer. Other software required by KETCindy is listed below.

1. Cinderella, available at `https://www.cinderella.de`.
2. Scilab, available at `http://www.scilab.org`.
3. Maxima (optional), available at `http://maxima.sourceforge.net`. Although optional, we will make heavy use of this CAS in Section 3.

Additional programs such as **R**, Fricas, Risa/Asir, Meshlab and gcc can also interact with KETCindy. Download and install them if necessary.

2.1 Setting up and running KETCindy

The complete KETCindy package can be downloaded in `zip` format from `http://ketpic.com/?lang=english`, by clicking on the `Dropbox - KetInstall` menu (in the current version of the web page, it is located on the left pane[1]). Once downloaded, it can be extracted in any folder. The only requirement is that nested folders in the distribution be kept nested in the same order (as some commands use relative paths). In MS Windows, a common place to download and extract KETCindy is under a folder with that name in `Users/login-name`; in that case, if your user name contains blank spaces, please place the distribution in a folder called `ketcindy` directly under `C:` to avoid problems. We will assume in what follows that the root folder of the distribution is called `ketcindy`.

1. Open `ketoutset.txt` in the root folder `ketcindy` with a text editor. Notice that LF is used as newline character in this text file. Adjust the paths there as needed, particularly

```
PathT="(check the path of your TeX distribution)";
PathS="(check the version of scilab)";
Pathpdf="(check the name of your pdf viewer)";
PathM="(check the version of Maxima)";
```

[1] A direct link to the whole distribution is `https://www.dropbox.com/sh/kzt2bgaz07n7dr0/AABZRvOrqqCp5Tn1JZYpnvSQa?dl=0`

2. Open `dirhead.txt` in the root folder and adjust the path
   `Dirhead="(the path of your ketcindy)";`
3. Double-click `template.cdy` in the root folder, Cinderella will be launched. If it does not start, refer to `HowtoInstall` in the folder `InstallforMac(Win)` inside `KetInstall`.
4. In the Cinderella window, from the top menu, select `Scripting > Reveal Plugin Folder`, then the folder containing the Cinderella plugins will open. Copy into it the files `dirhead.txt` and `KetCindyPlugin.jar` that can be found into the folder `ketcindy > ketjava`.
5. Close the plugins window and temporarily quit `template.cdy` without saving. The previous steps have the only purpose of making the KETCindy plugins available to Cinderella. Now they can be executed.
6. Reload `template.cdy`. A triangle surrounded by a white frame will appear on the screen, provided the KETCindy libraries have been successfully loaded.

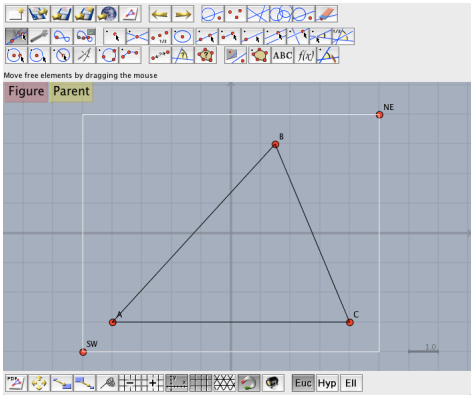


Figure 1: Cinderella window showing `template.cdy`.

By pressing the `Figure` button located at the upper left, a `pdf` file produced by KETCindy will pop up. Objects on the screen can be modified at will and the corresponding `pdf` will be updated. For example, the triangle on the right of figure 2 was obtained by displacing point B to the left of its original position.

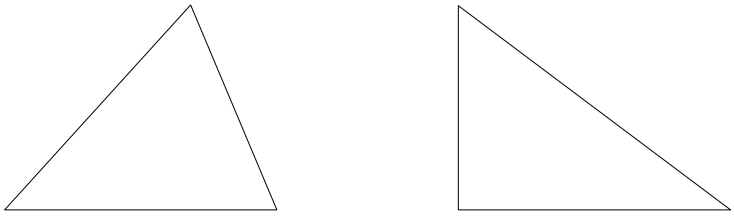


Figure 2: Two `pdf` graphics.

Actually, the graphics data are codes of TPIC specials for ordinary LaTeX, or pict2e commands for pdfLaTeX, written in the text file `template.tex`. Thus,

this file can be used as any other just by copying it into the working directory and using, for instance, the command `\input{template.tex}` to include it in the current document. Notice that, for this to work, you should copy the style files `ketpic.sty` and `ketlayer.sty` into the sub-folder `ketpicstyle` of your working directory.

2.2 Producing graphics

Cinderella comes with its own scripting language, CindyScript, which is easy to use and distinguishes Cinderella from other DGS. Actually, KETCindy is a macro package for CindyScript, which means that the user should be able to write her own scripts (composed of KETCindy commands) on the Cinderella script editor to produce a LaTeX graphics file.

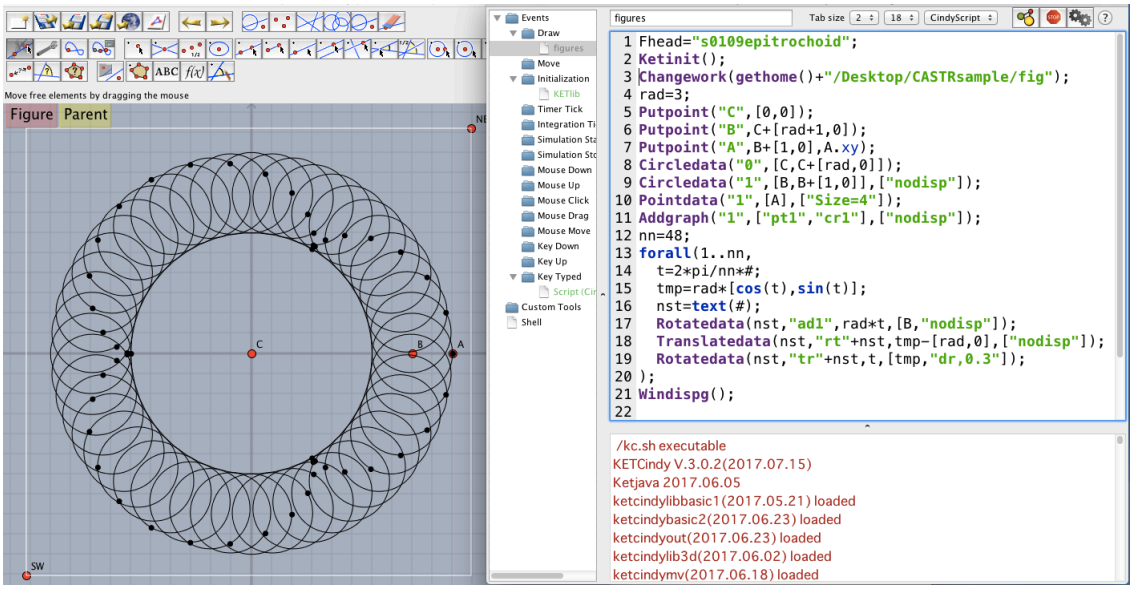


Figure 3: Script Editor and Console

There is a help system explaining the syntax of most commands. For example, to know how to use `Putpoint` just write `Help("Putpoint")` anywhere in the script editor and execute the query by pressing the gears symbol at the upper right corner of the editor (this serves to execute *any* script). The following brief explanation appears:

```
Putpoint("A",[1,2],[1,A.y]);
     put a point at the specified position
Putpoint3d("A",[2,1,3]);
Putpoint3d(["A",[2,1,3]]);
Putpoint3d(["A",[2,1,3]],"fix");
Putpoint3d(["A",[2,1,3]],["fix"]);
```

Remark: Points on the Cinderella screen are called 'geometric points', can be created by using a button easily spotted in the toolbar, and can move freely. Here, we are

using `Putpoint` to create a geometric point with given coordinates in the script editor. This procedure is often used to create a fixed (i.e., non-movable) point.

Once a script has been written and successfully executed, by pressing the `Figure` button as usual, the LaTeX graphics file is obtained.

2.3 Advanced examples

As a good DGS, Cinderella has an abundance of commands and tools to draw geometric figures; however, it is not so feature-rich in producing the graph of a function, or an analytically defined figure. We have added commands to do such tasks in KETCindy, and in this subsection we offer a couple of applications in advanced mathematics.

2.3.1 Solution curve of a differential equation

The command `Deqplot` is used to draw a solution curve of a given differential equation. Its syntax is: `Deqplot(name, deq, range, initial value, options);`
Here, each one of `deq` and `range` is given as a string, and the derivative $x' = \frac{dx}{dt}$ is written as x‘ because Scilab uses a single quote for a string delimiter. In the following example, the geometric points G, G, L lie on segments AB, EF, HK, respectively, and are used to change the initial values or coefficients of the equation.

```
Ketinit();
Deqplot("1","y‘‘=-L.x*y‘-G.x*y",
  "t=[0,XMAX]",0,[C.y,0],["Num=200"]);
  // Num is the number of partitions
  // the defaut is "Num=50"
Expr(M,"e","\dfrac{d^2 x}{dt^2}+"
  +L.x+"\dfrac{dx}{dt}+"+G.x+"x=0");
  // "e" means east
Expr(C,"w","x_0="+C.y);
  // "w" means west
Windispg();
```

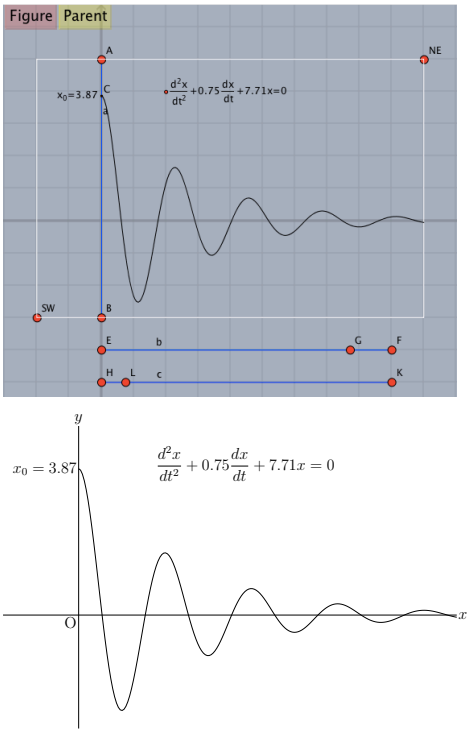


Figure 4: Solution curve and the resulting `pdf`.

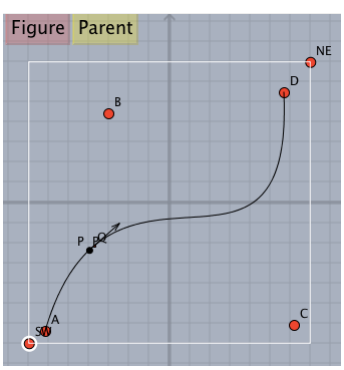
2.3.2 Drawing a Bézier curve

Bézier curves are useful to draw a free curve or to produce interactive teaching materials. The following commands are implemented in KETCindy.

```
Bezier("1",[A,D],[B,C]); // A,D are knots, B,C are control points
Mkbeziercrv("1",[[A,B,C,D],[[P,Q],[R,S],T]]); //A,B,C,D are knots
```

```
                                        // P,Q,R,S,T are control points
Mkbezierptcrv("1",[A,B,C]); //Control points created automatically
Ospline("1",[A,B,C,D]) // Oshima spline
```

The figure below shows an example in which the first command is used.




```
Ketinit();
Bezier("1",[A,D],[B,C],
["Num=100","dr,2"];
PutonCurve("P","bz1");
PutonCurve("Q","bz1");
pR=P+(Q-P)/|Q-P|};
Arrowdata("1",[P,pR],
["dr,1.5"];
Pointdata("1",[P]);
Letter(P,"nw","P");
Windispg();
```

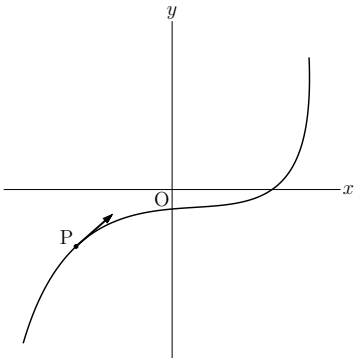


Figure 5: Bézier curve and the resulting `pdf`.

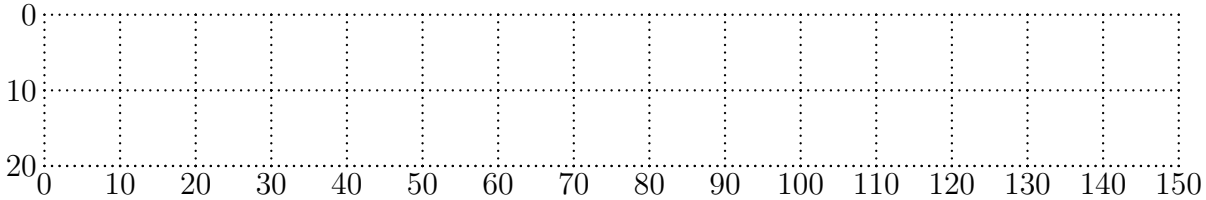
2.4 Placement of objects

When producing printed teaching materials or research papers, sometimes it is desirable to place objects at specific places. Very often this is difficult in LaTeX, so we have developed style files `ketlayer.sty`, for ordinary LaTeX, and `ketlayer2e.sty`, for pdfLaTeX, to deal with this issue. We explain here how to use these styles.

In `ketlayer.sty` we provide the `layer` environment, which takes two arguments: width and height (values in millimeters). Its usage is as follows:

```
\begin{layer}{150}{20}
\end{layer}
```

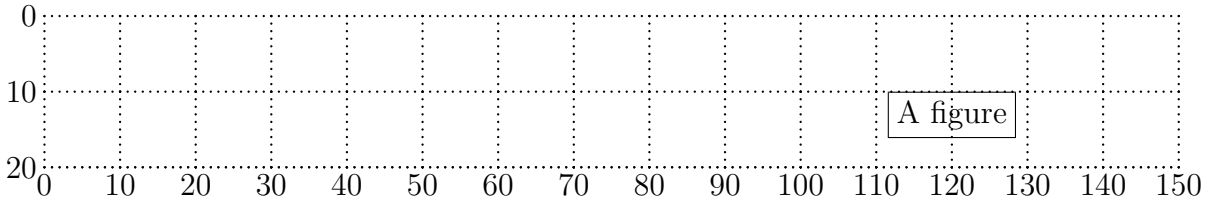
Compiling the source file, the following grids appears in the `pdf` file:



Any object can be placed at a desired, precise position by using a command such as `\putnote(+s,n,e,w,se,sw,ne,nw,c){xpos}{ypos}`, where `s`, `n`, `e`, `w`, `c` means south, north, east, west and center, respectively. A particular object can be placed independently of any other. For instance, the code

```
\begin{layer}{150}{20}
\putnote{120}{10}{\fbox{A figure}}
\end{layer}
```

will produce the following (useful for placing comments on figures, for example)



The grid, of course, only plays a guiding rôle. Once the object has been placed at the desired position, the value of the second argument of `layer` can be set to 0, to the effect that the grid will disappear. Thus, the code:

```
\begin{layer}{150}{0}
\putnote{120}{10}{\fbox{A figure}}
\end{layer}
```

simply produces
This technique has been used, for instance, in figure 6 below to correct the LaTeX

A figure

placement, which only takes into account the bounding box, or the brachistochrone graphics in page 8.

3 Calling a CAS and C from KETCindy

The use of a CAS is very convenient when producing teaching materials, because exact answers and analytic constructions can be obtained without effort, saving the time invested in long but straightforward computations. Thus, it is desirable to be able to call them from within KETCindy[3][4]. We have added new features allowing KETCindy to call CASs such as Maxima, FriCAS or Risa/Asir. In this section, we will take Maxima as an example. The overall process runs as follows:

1. Generate a script containing the callings to Maxima.
2. Execute the file. The result is returned by Maxima as a text or a list of texts.
3. Use the result in KETCindy.
4. Produce the `pdf` file.

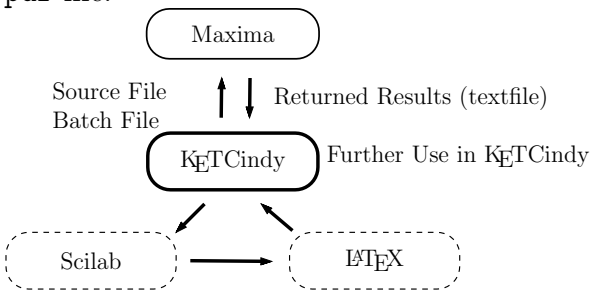


Figure 6: Flowchart of KETCindy with Maxima

When interfacing with Maxima, the command `Mxfun` is all we need to complete the task. Other commands such as `calcbyM` and `Mxtex` may be used for multistep and code conversion to LaTeX, respectively. The output of Maxima is returned to KETCindy in the form of a string of characters for further processing of the graphics. Used this way, KETCindy proves itself to be a powerful companion to Maxima as shown in the following examples.

3.1 Interactive material for the brachistochrone problem

The well-known brachistochrone problem is often used as a motivation in mathematics and physics courses related to the calculus of variations. It is a touchstone for any software whose goal is to ease the task of elaborating teaching materials, see [5].

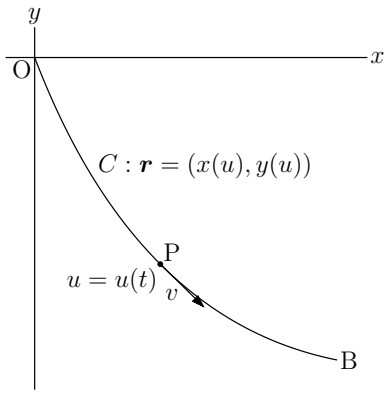
Let a point P start its motion without friction from point O, with the velocity $v = 0$, along a curve C represented parametrically as $\boldsymbol{r} = (x(u),\ y(u))$ $(0 \le u \le U)$, and suppose it reaches the final point B. For simplicity, place the coordinates so that O and B lie at $(0,\ 0)$ and $(5,\ -5)$, respectively.

A simple calculation leads to the differential equation of motion

$$\begin{cases} \dfrac{du}{dt} = \sqrt{\dfrac{-2gy}{\dot{x}^2 + \dot{y}^2}}\,, \\ u(0) = 0\,, \end{cases} \qquad (1)$$

and the formula for calculating total time T as

$$T = \int_0^U \sqrt{\frac{\dot{x}^2 + \dot{y}^2}{-2gy}}\,\mathrm{d}u\,. \qquad (2)$$



where $\dot{x} = \dfrac{dx}{du}$, $\dot{y} = \dfrac{dy}{du}$, and g is the acceleration of gravity. Maxima is then used:

1. To determine an inverted cycloid through B$(5,\ 5)$ represented by the parametric equations

$$\boldsymbol{r} = \big(a(u - \sin u),\ -a(1 - \cos u)\big)\ \ (0 \le u \le U)\,.$$

The main part of the script implementing this, has the following structure:

```
cmdL=[
  "assume",["a>0"],
  "fxy:[a*(u-sin(u)),-a*(1-cos(u))]",[],
  "eq:ratsimp",["(fxy[1]+fxy[2])/a"],
  "so1:find_root",["eq","u","%pi/2","%pi"],
  "eqr:ev",["fxy[1]-5","u=so1"],
```

```
 "so2:find_root",["eqr","a",0,5],
 "so1::so2",[]
];
CalcbyM("coeff",cmdL);println(coeff);
```

where `cmdL` is the command allowing to execute a sequence of Maxima commands written in KETCindy syntax, and the results are $U = 2.412011143913525$, $a = 2.864585187658752$.

2. To find the equation and the integrand of total time T. The relevant part of the script is now

```
"assume(g>0)",[],
"fxy:",[fxy],
"d2:diff(fxy[1],u)^2+diff(fxy[2],u)^2",[],
"d2:trigsimp(d2)",[],
"d2:factor(d2)",[],
"n2:2*g*(-fxy[2])",[],
"so1:ratsimp(n2/d2)",[],
"so2:ratsimp(sqrt(so1))",[],
"so3:ratsimp(1/so2)",[],
"so2::so3",[]
]);
CalcbyM(name,cmdL,[""]);
```

where `fxy` is a string describing the parametric equations of a curve; for example, `fxy` applied to a Bézier curve from O to B with control points C($c1,\ c2$) and D($d1,\ d2$) would be

```
fxy="[3c1*(1-u)^2*u+3d1*(1-u)*u^2+5*u^3,
      3c2*(1-u)^2*u+3d2*(1-u)*u^2-5*u^3]".
```

The resulting strings in the above script are called `so2` and `so3`, which represent the right side of the equation (1) and the integrand in (2), respectively. In the case of a Bézier curve, these strings are extremely complicated, as one would expect.

3. To find total time by analytical or numerical integration. In the case of a circle parameterized as

$$\boldsymbol{r} = \big(5(1-\cos u),\ -5\sin u\big)\ \ (0 \le u \le \frac{\pi}{2})$$

the total time can be obtained with the formula

$$T = \int_0^{\pi/2} \sqrt{\frac{5}{2g\sin u}}\,\mathrm{d}u$$

Maxima can compute this integral. The relevant KETCindy command is

```
Mxfun("1","integrate",["sqrt(5/(2*g*sin(u)))","u",0,"%pi/2"],
                                        ["Set=assume(g>0)"]);,
```

and the result is

$$\frac{\sqrt{5}\,\beta\left(\frac{1}{4},\frac{1}{2}\right)}{2\sqrt{2}\sqrt{g}} = 1.324339055215268\,.$$

In other cases, such as that of a Bézier curve, Maxima cannot compute the total time analytically. Then, we must resort to numerical methods of integration, some of which are implemented in Maxima. Here, because equation (1) is defined by an improper integral, one should use `quad_qags` instead of `romberg`. Here are the relevant commands in our case, along with the result in the case of a circle:

```
Mxfun("2","quad_qags",["sqrt(5/(2*9.8*sin(u)))","u",0,"%pi/2"]);
```

$$T = 1.324339055215265$$

In [5] we have developed an interactive teaching material to study the brachistochrone problem using Bézier curves. The idea is to model the descent curve by a Bézier one, adding two control points C and D, and asking the students to try to minimize the descent time by displacing these new points. For this material, item 3 above is not appropriate because Maxima is called each time a control point is displaced, so we adopt another strategy as follows:

1. Find the general formula of (1) using Maxima as in item 2 above.
2. Solve it numerically using `deqplot`, implemented in KETCindy.

Then the curve shape can be changed interactively, and the total time invested in the descent is shown in real time, allowing the student to compare the results with different curves (arcs of circumferences, parabolas, etc.)

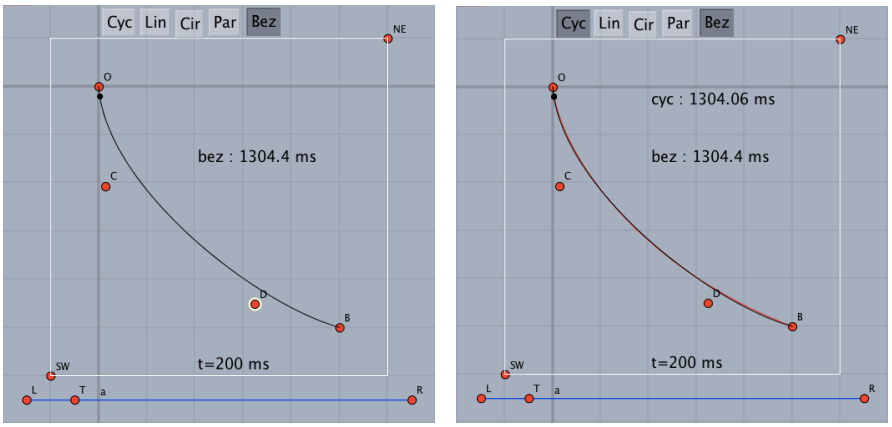


Figure 7: Fitting the Bézier curve to the inverted cycloid.

3.2 Dynamics of Poincaré sections

Recently, we have developed a package for studying Poincaré sections of Hamiltonian autonomous systems in Maxima, called `poincare.mac`, see [6]. Using this package,

KETCindy can produce a pdf presentation slide or even a movie showing the dependence of the structural properties of Poincaré sections with respect to the energy of the system, which can be an effective aid to illustrate the kind of behavior that the celebrated KAM theorem describes. Here we present the well-known example of the Hénon-Heiles model, which has the Hamiltonian

$$K(q_1, p_1, q_2, p_2) = \frac{1}{2}(q_1^2 + p_1^2 + q_2^2 + p_2^2) + q_2\, q_1^2 - \frac{q_2^3}{3}\,.$$

The commands provided by KETCindy to generate the Poincaré sections are as follows:

```
cmdL1=concat(Mxload("rkfun.lisp"),Mxbatch("poincare.mac"));
cmdL1=concat(cmdL1,[
 "K(q1,p1,q2,p2):=1/2*(q1^2+p1^2+q2^2+p2^2)+q2*q1^2-q2^3/3",[],
 "series1:realroots"
]);
cmdL2=[
"data1:poincare2d",
    ["K","XK","[-0.2,rhs(first(series1)),-0.2,0.1]",
      "[t,-300,300,0.05]","[q1,0,q2,p2]"],
 "data2:poincare2d",
   ["K","XK","[-0.2,rhs(second(series1)),-0.2,0.1]",
     "[t,-300,300,0.05]","[q1,0,q2,p2]"],
 "data1::data2",[]
];
```

where poincare2d is a function defined in poincare.mac. The function mf(s) describes the state with total energy $H = 1/s$:

```
mf(s):=(
 regional(p0,p1,tmp);
 cmdH=["K(-0.2,p1,-0.2,0.1)=1/"+text(s)];
 cmdL=append(cmdL1,cmdH);
 cmdL=concat(cmdL,cmdL2);
 CalcbyM("data",cmdL,[""]);
 forall(1..(length(data)),
   data_#=replace(data_#,"e-4","*10^(-4)");
   data_#=replace(data_#,"e-5","*10^(-5)");
   data_#=parse(data_#);
 );
 Setcolor("red");
 Pointdata("1",data_1,["Size=2"]);
 Pointdata("2",data_2,["Size=2"]);
 Setcolor("black");
 Expr([0.25,0.9],"c","H=1/"+text(s));
);
```

Then, KETCindy produces a set of `pdf` slides (that can be integrated into a short movie) illustrating how the Poincaré sections evolve with increasing energies. The main features of Hamiltonian dynamics are visually apparent: the persistence of closed orbits for low energies, and the apparition of chaos for higher values.

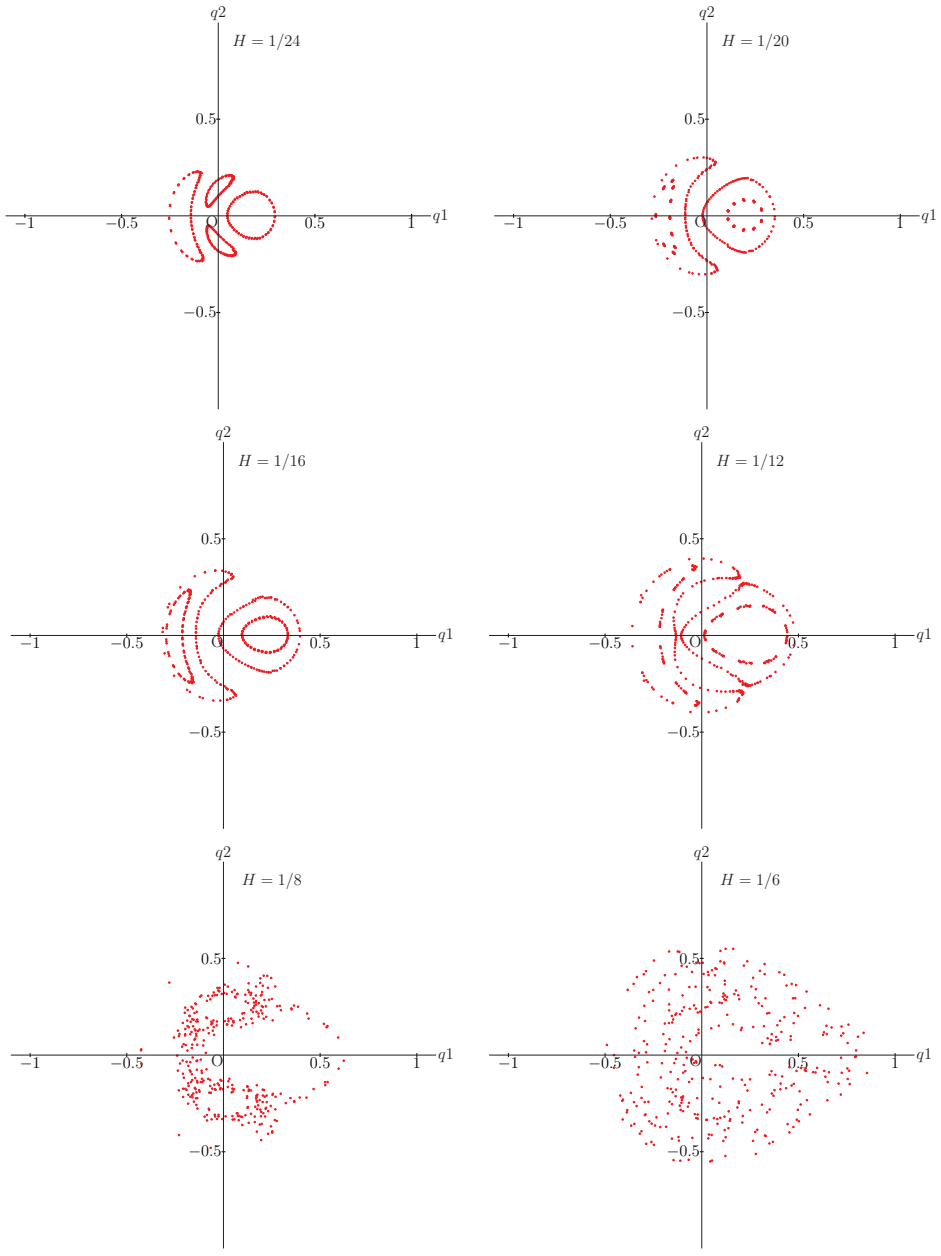


Figure 8: $2D$ Poincaré sections.

3.3 Interfacing C

Because Scilab and Cinderella are interpreted languages, their performance in computationally demanding tasks such as the removal of hidden parts in $3D$ graphics is suboptimal.

These tasks require a compiled language, and KETCindy has a extended functionality for calling C. This is illustrated in this section by a cone with sections by planes, the process to produce the figure below is as follows:

1. Find the silhouette lines of the cone.
2. Remove hidden parts of the cone.
3. Remove hidden parts of the axis and wire frames.
4. Compute the section of the cone by a plane.
5. Remove the hidden parts of these sections.

As stated, the removal of hidden parts takes a lot of time in Scilab (about 2–3 minutes in this case). This is the *raison d'être* for calling C from KETCindy. The main features of the calling function are:

(a) Definitions of the surface function and constants are written to a header file.
(b) The call to C syntax is the same as the call to CASs ones.
(c) The result is returned to KETCindy as a text.
(d) The graphics is drawn in the usual manner within KETCindy.

The following is the main part of the script that generates the graphics:

```
Fd=["p","x=u*cos(v)","y=u*sin(v)","z=4-2*u",
    "u=[0,2]","v=[0,2*pi]","e"];
FdC=[["x=u*cos(v)","y=u*sin(v)","z=4-2.0*pow(u,1.0)"],
      "u=[0.0,2.0]","v=[0.0,2*M_PI]","e",
      [100,100],[5000,3000,500],[0.02,0.1]];
MainC=["writesfbd",["sfbd"],
       "writeax",["ax3d"],
       "writewire",["wire",4,[[1,sqrt(2),sqrt(3)],pi/3*(1..6)]],
       "writecut",["sfcut1",Assign("2*x+a*z-b",["a",0,"b",2])],
       "writecut",["sfcut2",Assign("2*x+a*z-b",["a",1,"b",1.5])],
       "writecut",["sfcut3",Assign("2*x+a*z-b",["a",4,"b",5])]];
```

Pressing the `Ckc` button in the Cinderella window, we obtain the following LaTeX figure in a few seconds:

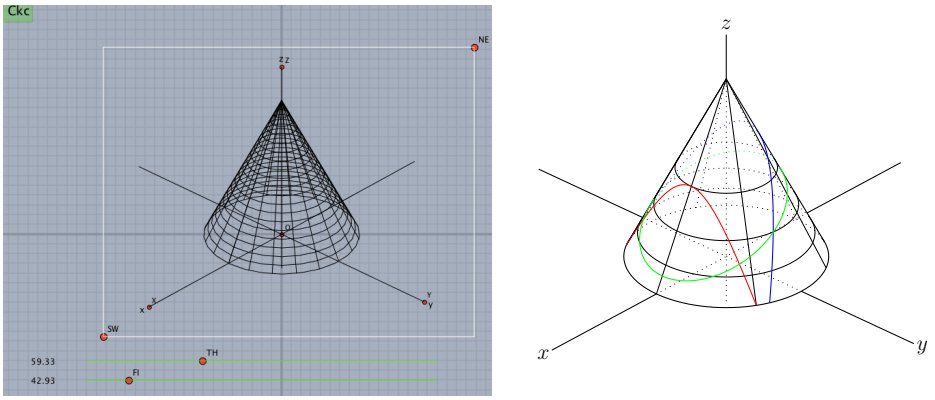


Figure 9: Drawing a cone and its sections by planes.

4 Conclusions

The ability of drawing high-quality graphics and properly placing objects such as figures, are essential elements to produce teaching materials, and even research papers. These can be easily achieved with the use of KETCindy, through the script language present in Cinderella. Free CASs such as Maxima are also very helpful in creating computational labs sessions, because of their symbolic capabilities allowing to tackle more advanced problems. Further challenges, as three-dimensional graphics, are within reach with the functionality to call the C programming language. This opens the door to the consideration of other applications requiring the collaborative use of DGS, CASs and C in teaching and researching activities.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers 15K01037, 15K00944, and 16K01152 (ST). It was also partially supported by a CONACyT research project CB-179115 (JAV).

References

[1] Takato S., What is and how to Use KETCindy – Linkage Between Dynamic Geometry Software and Collaborative Use of KETCindy and Free Computer Algebra Systems and LaTeX Graphics Capabilities –, Mathematical Software – ICMS 2016, LNCS **9725**, 371–379, Springer, 2016.

[2] Kaneko M., Yamashita S., Kitahara K., Maeda Y., Nakamura Y., Kortenkamp U, Takato S., KETCindy – Collaboration of Cinderella and KETpic, Reports on CADGME 2014 Conference Working Group, The International Journal for Technology in Mathematics Education, **22**(4), 179–185, 2015.

[3] Takato S., McAndrew A., Vallejo J. A., Kaneko M., Collaborative Use of KETCindy and Free Computer Algebra Systems, Mathematics in Computer Science, 1–12, 2017, `https://doi.org/10.1007/s11786-017-0303-7`

[4] Kobayashi S., Takato S., Cooperation of KETCindy and Computer Algebra System, Mathematical Software – ICMS 2016, LNCS **9725**, 351–358, Springer, 2016.

[5] Takato S., Brachistochrone Problem as Teaching Material–Application of KETCindy with Maxima, Computational Science and Its Applications–ICCSA, 251-261, Springer, 2017

[6] Vallejo, J. A., Poincaré sections of Hamiltonian autonomous systems in Maxima. Available at `http://galia.fc.uaslp.mx/~jvallejo/PoincareDocumentation.pdf`.