Computational Physics with Maxima or R:
Example 2
Surfaces of Section for the Hénon-Heiles Potential *

Edwin (Ted) Woollett

August 28, 2015

Contents

*The code examples use **R ver. 3.0.2** and **Maxima ver. 5.31** using **Windows 7**. This is a live document which will be updated when needed. Check `http://www.csulb.edu/~woollett/` for the latest version of these notes. Send comments and suggestions for improvements to `woollett@charter.net`

Code files which accompany example2.pdf are
1. example2.mac
2. example2.R
3. hhplota.R

Feedback from readers is the best way for this series of notes to become more helpful to users of **R** and **Maxima**. *All* comments and suggestions for improvements will be appreciated and carefully considered.

1 Introduction

A summary of the the Hénon-Heiles potential and its generalizations can be found on Wikipedia
http://en.wikipedia.org/wiki/H%C3%A9non-Heiles_equation
from which we quote:

> The Hénon-Heiles System [potential] is used to model stars. It is expressed as
>
> $$V(x,y) = \frac{1}{2}(x^2 + y^2 + 2\,x^2\,y - \frac{2}{3}y^3) \tag{1.1}$$
>
> While at Princeton in 1962, Michel Hénon and Carl Heiles worked on the non-linear motion of a star around a galactic center where the motion is restricted to a plane. They published a paper that describes their work in 1964.
>
> The Hénon-Heiles System (HHS) is defined by the following four equations:
>
> $$\begin{aligned} \dot{x} &= p_x \\ \dot{p_x} &= -A\,x - 2\,x\,y \\ \dot{y} &= p_y \\ \dot{p_y} &= -B\,y + \epsilon\,y^2 - x^2 \end{aligned}$$
>
> where A, B, and ϵ are real numbers and $A > 0$, $B > 0$ .... It can be solved for some cases using Painlevé Analysis. The Hamiltonian for the HHS is
>
> $$H = \frac{1}{2}\,(p_x^2 + p_y^2) + \frac{1}{2}\,(A\,x^2 + B\,y^2) + x^2\,y - \frac{1}{3}\,\epsilon\,y^3 \tag{1.2}$$
>
> ... [The] Hénon-Heiles system shows rich dynamical behavior.

And quoting a 2014 analysis by the mathematician A. Lesfari (**http://arxiv.org/pdf/1401.3575.pdf**)

> First studied as a mathematical model to describe the chaotic motion of a test star in an axisymmetric galactic mean gravitational field [7], this system is widely explored in other branches of physics. It [is] well-known from applications in stellar dynamics, statistical mechanics and quantum mechanics. It provides a model for the oscillations of atoms in a three-atomic molecule [5]. Usually, the Hénon-Heiles system is not integrable and represents a classical example of chaotic behaviour. Nevertheless at some special values of the parameters it is integrable ....

in which the reference [7] is to
Hénon, M. and Heiles, C.: The applicability of the third integral of motion; some numerical experiments, Astron. J., 69, 73-79 (1964).

A brief introduction is at
http://mathworld.wolfram.com/Henon-HeilesEquation.html
which includes some examples of "surfaces of section." The related webpage
http://mathworld.wolfram.com/SurfaceofSection.html
introduces the surface of section concept:

> A surface (or "space") of section, also called a Poincaré section, is a way of presenting a trajectory in n-dimensional phase space in an (n-1)-dimensional space. By picking one phase element constant and plotting the values of the other elements each time the selected element has the desired value, an intersection surface is obtained.

A good theoretical introduction to general properties of orbits in the context of astrophysics is Chap.3: The Orbits of Stars, in the text **Galactic Dynamics**, by James Binney and Scott Tremaine.

The website of Andreas Ernst, a computational astrophysicist (experimental stellar dynamics) at The University of Heidelberg
`http://wwwstaff.ari.uni-heidelberg.de/mitarbeiter/ernst/index.html\mvs`
has a link to some movies and plots at
`http://wwwstaff.ari.uni-heidelberg.de/mitarbeiter/ernst/movies.html`
One third of the way down that page are two surface of section plots for our potential, and he remarks

> This is a collection of some of my plots. One the one hand, you can get an impression what it means to do computational physics, on the other hand, you should see how much fun it is! For most of the plots, I used different integrators, like 4th and 8th order Runge-Kutta, the Hermite scheme, or fancy ones, like a time-transformed 8th order composition scheme where the highest accuracy was needed. For the plotting itself, I used gnuplot, IDL and Mathematica.

For the first plot, downloaded from his "Simply Integrate" webpage
`http://www.simplyintegrate.de/sim/index.html`
as a **`png`** file and converted to an **`eps`** file using Cygwin:

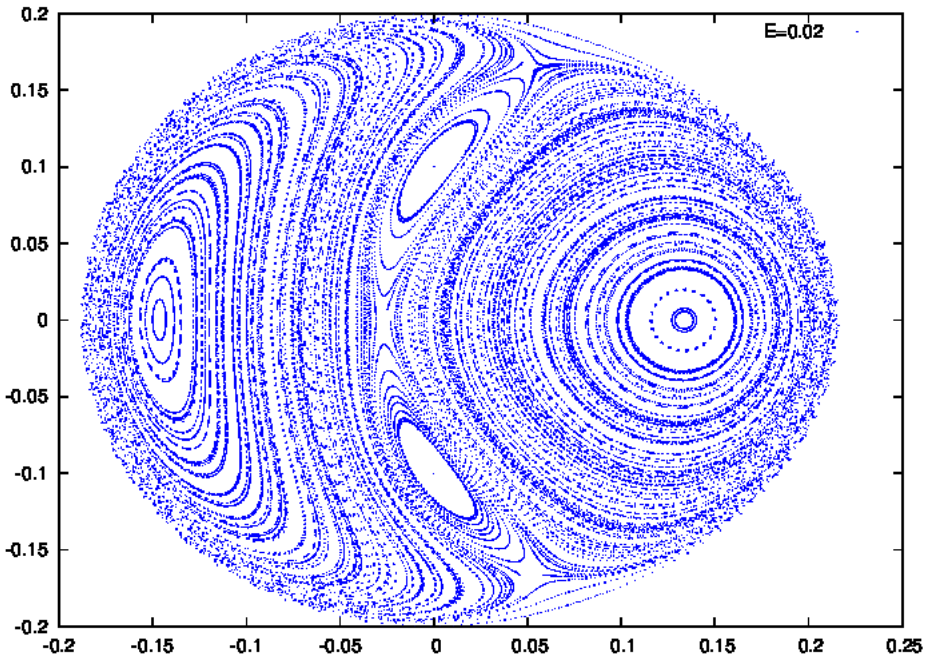

Figure 1: $E = 0.02$

he has the information:

> Poincaré section in the Hénon-Heiles potential at $E = 0.02$ (no chaos, plotted are all orbits at the instant of crossing $x = 0$ with $v_x > 0$)

And for the second plot, similarly downloaded and converted:

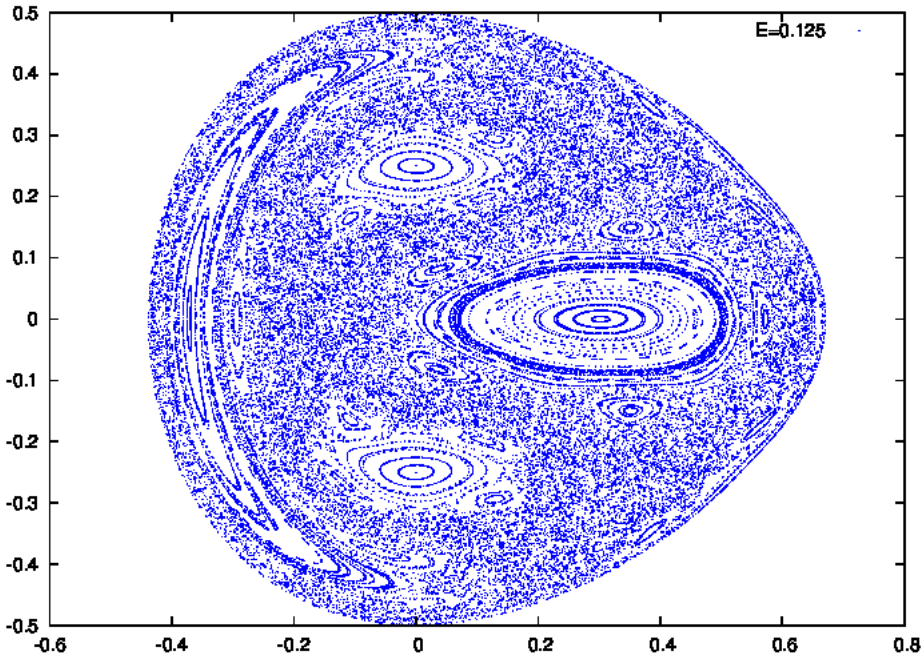

Figure 2: $E = 0.125$

he has the information:

> This time at $E = 0.125$ (mixture of chaos and order, i.e., it is a "system with divided phase space")

Two thirds of the way down the webpage **`http://www.massey.ac.nz/~rmclachl/gi.html`** created by Robert McLachlan at Massey University, New Zealand, is a row of 3 plots related to our potential. In reference to the third plot in the row, he remarks

> On the right, a (nonsymplectic) integrator has destroyed the torus. In the section picture, 10^5 time steps for 10 different initial conditions are shown. Smooth curves ("KAM tori") correspond to regular, quasiperiodic motion; clouds correspond to chaotic motion.

A general feature of quasi-periodic motion is the appearance of elliptic and hyperbolic fixed points in a surface of section plot.

We quote a comparison of elliptic and hyperbolic fixed points in the context of fluids and chaotic motion at the webpage
`http://plus.maths.org/content/births-and-deaths-fluid-chaos`
in an article "Births and Deaths in Fluid Chaos", by Marianne Freiberger, which discusses the research of Jerry Gollub (Haverford College) and Nick Ouellette (Yale University), and which has the figure

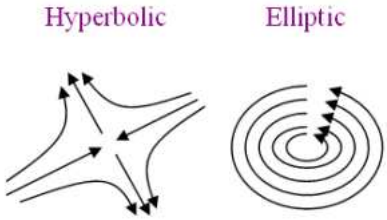

Figure 3: Fluid Flow Close to Elliptic and Hyperbolic Fixed Points

and discussion

> Rather than describing this complex motion by measuring the velocity at a great number of points over time, Ouellette and Gollub decided to look for geometric features that characterise the whole flow. They were interested in two sets of special points: those that lie at the centre of vortices, known as elliptic points, and those towards which the flow converges along one direction, but from which it diverges along another direction. These are known as hyperbolic points, or saddle points (see figure). Elliptic points have the flow whirling around them, while hyperbolic points arise, for example, at a point where four vortices meet, with each neighbouring pair of vortices turning in different directions. These types of points tell you, at a given point in time, where vortices are centred, and where the domain of one vortex ends and another starts – very important information in a flow that is dominated by vortices. Could this information be enough to characterise the whole flow?
>
> The idea was a good one, but there was a problem: how do you locate the special points? One possibility is to look for points of zero velocity, but this is hard to do accurately. But Ouellette had an ingenious idea. He noted that near elliptic points, tracer particles circulate in very small circles, and near hyperbolic points, they turn sharp corners. In both cases, the trajectories of particles exhibit very high curvature. Using ideas from differential geometry, Ouellette came up with a reliable way of locating the special points by measuring the curvature of the tracer particles' trajectories.

The Nonlinear Dynamics webpage
`http://dynamics.mi.fu-berlin.de/research/bif-eq-surfaces/index.php?q_menu=0`
has the figures:

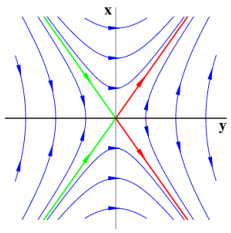

Figure 4: Particle Paths Near a Hyperbolic Fixed Point

and

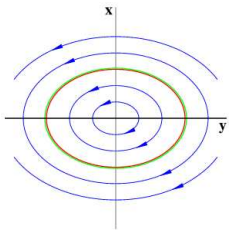

Figure 5: Particle Paths Near an Elliptic Fixed Point

The language of fixed points is taken over from particle and fluid trajectory paths into the patterns that appear in a surface of section plot (which is not a plot of particle paths). One speaks of "phase trajectories" in phase space (four dimensional for the case of a single particle moving in a plane). And watching these phase trajectories in a surface of section plot is a powerful method of finding regions of initial conditions which give rise to long term chaotic behavior.

Quoting G.L. Baker and J.P. Gollub (p.41) in the second edition of **Chaotic Dynamics**:

> The fundamental characteristic of a chaotic physical system is its sensitivity to the initial state. Sensitivity means that if two identical mechanical systems are started at initial conditions $\mathbf{x}$ and $\mathbf{x} + \boldsymbol{\epsilon}$ respectively, where [the magnitude of] $\boldsymbol{\epsilon}$ is a very small quantity, their dynamical states will diverge from each other very quickly in phase space, their separation increasing exponentially on the average.

We also quote from a 2014 paper by Jamal Sakhr (**`http://arxiv.org/pdf/1312.1978.pdf`**)

> In numerical Poincaré surface-of-section (SOS) computations, the usual procedure is to compute numerical trajectories (i.e., "pseudotrajectories") for a large number of initial conditions and then observe the resulting point pattern that ensues from the intersection of these numerically computed trajectories with a chosen SOS. In 2D conservative systems, the visual signature of fully developed chaos is an apparently random scatter of points (on the SOS) generated from one initial condition.
>
> ... Interestingly, this point pattern (obtained from classical deterministic laws and equations) is visually indistinguishable from a realization of a [two-dimensional homogeneous] Poisson point process in the SOS.
>
> ... For a fully chaotic 2D system, almost any pseudotrajectory of the Poincaré map will explore the full phase space of the map ergodically, i.e., almost any pseudotrajectory of the Poincaré map will densely and uniformly cover the entire SOS.
>
> ... individual chaotic trajectories need not densely cover the entire phase space. Successive point intersections of any such pseudotrajectory with a canonical SOS will (after sufficient time) uniformly cover some subset W of the SOS. The ensuing point set should be indistinguishable (insofar as its distance characteristics are concerned) from any suitably defined realization of a Poisson point process in W (of appropriate intensity).
>
> In generic mixed systems, there are no ergodic components (i.e., no positive-measure regions of phase space completely devoid of islands). Nevertheless, there generally exist values of a system parameter at which there are only a few observable islands and the fraction of the phase space volume occupied by these islands is small (e.g., < 0.1). In such cases, the chaotic sea covers nearly uniformly most of the available phase space, ...
>
> The system parameter referred to above could be a coupling or perturbation parameter (e.g., kicked rotor [31]), a shape parameter defining a family of billiards (e.g., the δ parameter defining the family of lemon billiards [32]), or even simply the total system energy in time-independent potential systems (e.g., Hénon-Heiles [33], Pullen-Edmonds [34] etc.).
>
> The typical scenario in the last case (the most fundamental case for smooth Hamiltonian systems) is a phase space dominated by regular trajectories at low energies and chaotic trajectories at high energies with a mix of regular and chaotic trajectories at intermediate values of the energy. In the well-known Hénon-Heiles system, for example, most trajectories are quasi-periodic at $E = 1/12$ and chaotic at $E = 1/6$ with a mix of regular and chaotic trajectories at $E = 1/8$ (see Ref. [33]).

2 Exploring the Hénon-Heiles Potential (Energy)

The Hénon-Heiles potential (energy) is, in the form originally explored for galactic motion of a star,

$$V(x,y) = \frac{1}{2}\,(x^2 + y^2) + y\,x^2 - \frac{1}{3}\,y^3 \tag{2.1}$$

which is obviously equal to zero when both x and y are zero. It is also obvious that the potential is an even function of x, so the potential has the same value (for given y) at positive and negative values of x.

We can search for local minima and maxima, using the necessary (but not sufficient) conditions $\partial V/\partial x = 0$ and $\partial V/\partial y = 0$.

```
(%i1) PE(x,y) := (x^2 + y^2)/2 + y*x^2 - y^3/3$
(%i2) eq1:diff(PE(x,y),x);
(%o2) 2*x*y+x
(%i3) eq2:diff(PE(x,y),y);
(%o3) -y^2+y+x^2
(%i4) soln:solve([eq1,eq2],[x,y]);
(%o4) [[x = 0,y = 1],[x = 0,y = 0],[x = -sqrt(3)/2,y = -1/2],
       [x = sqrt(3)/2,y = -1/2]]
(%i5) soln :  fullmapl('rhs,soln);
(%o5) [[0,1],[0,0],[-sqrt(3)/2,-1/2],[sqrt(3)/2,-1/2]]
(%i6) for s in soln do print(apply('PE,s))$
1/6
0
1/6
1/6
```

We recover the zero potential value at the origin, and find three points at the corners of an isosceles triangle where the potential has the value $1/6 = 0.1667$.

2.1 Equipotentials Using Maxima

We can use the contributed code **`implicit_plot`** to find equipotentials (contour lines) corresponding to specific numerical values, with the syntax

```
implicit_plot(expr,[x,x1,x2],[y,y1,y2],...)
implicit_plot([expr1,epxpr2,...],[x,x1,x2],[y,y1,y2],...)
```

to plot points where **`expr = 0`**. Thus we get points where the potential has value **`1/6`** with (we first have to load the code)

```
(%i7) load(implicit_plot);
(%o7) "C:/PROGRA~1/MAXIMA~3.2/share/maxima/5.31.2/share/
                          contrib/implicit_plot.lisp"
(%i8) implicit_plot (PE(x,y)-1/6,[x, -2, 2], [y, -2, 2],
       [style,[lines,2]],[gnuplot_preamble, "set grid"])$
```

which produces the plot

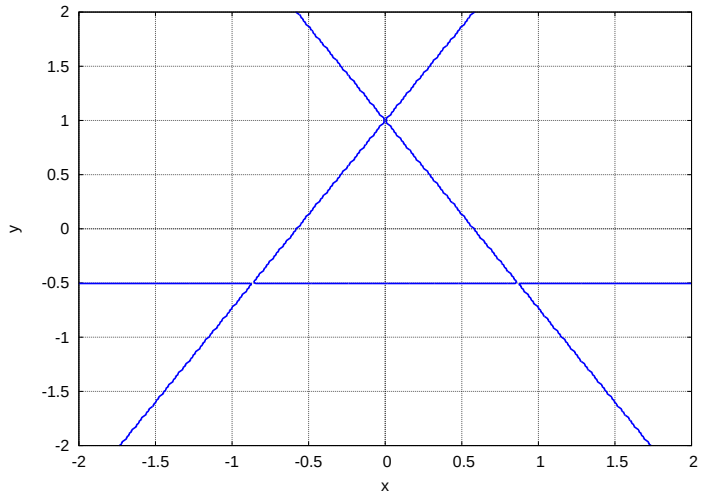

Figure 6: Locus of $V(x, y) = 1/6$

Here we plot six equipotential curves corresponding to $V = -4$ (cyan),$V = -2$ (green), $V = 0.05$ (black), $V = 1/6$ (blue), $V = 2$ (red), and $V = 4$ (magenta).

```
(%i9) implicit_plot ([PE(x,y)+4, PE(x,y)+2,PE(x,y)-0.05,
                      PE(x,y)-1/6,PE(x,y)-2,PE(x,y)-4],
           [x, -4, 4], [y, -4, 4],[style,[lines,2]],
           [color,cyan,green,black,blue,red,magenta],
           [legend,"-4","-2","0.05","1/6","+2","+4"],
           [gnuplot_preamble, "set grid"])$
```

which produces the plot

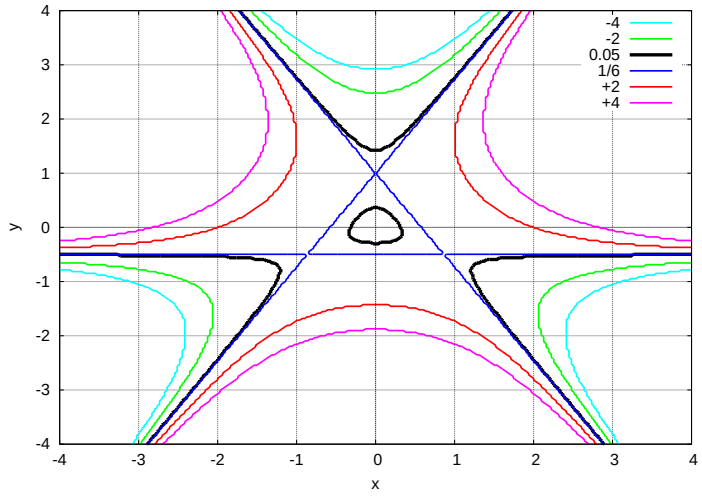

Figure 7: Equipotentials of $V(x, y)$ Using Maxima

From this plot, we see that the three points at the corners of the isosceles triangle where the potential has the value $1/6 = 0.1667$ are saddle points. Surrounding the central region where $V(x, y) < 1/6$ there are three valleys, which alternate with three ridges. With the total mechanical energy $E = KE + V(x, y)$, we need $E < 1/6$ to prevent the particle from escaping the central region near the origin.

2.2 Equipotentials Using R

We use the **contour** function in **R**. In the following, **x** and **y** are **R** vectors and **z** is a **R** matrix.

```
> x = seq(-4, 4, length=1000)
> y = seq(-4, 4, length=1000)
> z = outer(x,y,function(x,y) (x^2+y^2)/2 + y*x^2 - y^3/3)
> contour(x,y,z,
+     levels= c(-4,-2,0.05,1/6,2,4), lwd=2,
+     col=c("cyan","green","black","blue","red","magenta"),
+     drawlabels = FALSE, xlab = "x", ylab = "y")
> grid(lty="solid", col = "darkgray")
> legend("topright", lwd = 2,
+        col=c("cyan","green","black","blue","red","magenta"),
+        legend = c("-4","-2","0.05","1/6","2","4"), cex=1.2)
```

which produces the plot

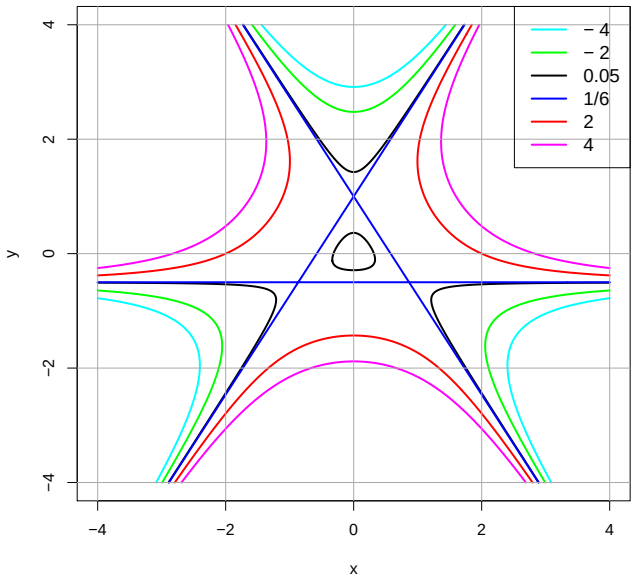

Figure 8: Equipotentials of $V(x, y)$ Using R

3 The Hamiltonian Equations of Motion

We describe the dynamical problem in terms of dimensionless energy, position vector, and momentum vector such that the mass of the particle disappears from the Hamiltonian, which is assumed to be

$$H = \frac{1}{2}\left(p_x^2 + p_y^2\right) + V(x, y) \tag{3.1}$$

The particle moves in the (x, y) plane. The momentum components conjugate to the position components (x, y) are (p_x, p_y). The equations of motion are then Hamilton's equations

$$\frac{dx}{dt} = \frac{\partial H}{\partial p_x} = p_x \tag{3.2}$$

$$\frac{dy}{dt} = \frac{\partial H}{\partial p_y} = p_y \tag{3.3}$$

$$\frac{d\,p_x}{d\,t} = -\frac{\partial H}{\partial x} = -\frac{\partial V}{\partial x} = -2\,x\,y - x \tag{3.4}$$

$$\frac{d\,p_y}{d\,t} = -\frac{\partial H}{\partial y} = -\frac{\partial V}{\partial y} = y^2 - y - x^2 \tag{3.5}$$

These equations (for any $V(x,y)$) conserve the energy E, so the constraint

$$H(x,y,p_x,p_y) = E \tag{3.6}$$

restricts the trajectory to lie in a three dimensional manifold embedded in the four-dimensional phase space.

These four coupled first-order ordinary differential equations are integrated using our homemade Runge-Kutta code **`rk4`** and **`rk4_step`** (see the code file **`example2.mac`**). The Runge-Kutta code returns a list of trajectory points which include the times and have the form **`[t,x,y,px,py]`**. We extract the x and y values (for a trajectory plot) using our homemade **`take`** function, for example **`xL : take(rkpts,2)`** and **`yL : take(rkpts,3)`**. A trajectory plot is then produced using **`plot2d([discrete,xL,yL],[xlabel,"x"],[ylabel,"y"])$`**, (plus other optional elements of the plot).

A function **`getsection`** takes as input that list of Runge-Kutta points, and notices when x changes sign. Whenever x changes sign, **`getsection`** calls **`goback`**, which integrates the dynamical equations backwards to $x = 0$, and captures the values of (y, p_y) at that moment. **`getsection`** then returns a list of (y, p_y) pairs which can be immediately used to make a "surface of section" plot.

Because the right-hand sides of the four first-order differential equations do not explicitly contain the independent variable t, the code **`getsection`** uses x as the independent variable, and the equations of motion used there and in **`goback`** are

$$\frac{d\,y}{d\,x} = \frac{d\,t}{d\,x}\,\frac{d\,y}{d\,t} = \frac{1}{dx/dt}\,\frac{d\,y}{d\,t} = \frac{p_y}{p_x} \tag{3.7}$$

$$\frac{d\,p_x}{d\,x} = \frac{d\,t}{d\,x}\,\frac{d\,p_x}{d\,t} = -\frac{1}{p_x}\,\frac{\partial V}{\partial x} \tag{3.8}$$

$$\frac{d\,p_y}{d\,x} = \frac{d\,t}{d\,x}\,\frac{d\,p_y}{d\,t} = -\frac{1}{p_x}\,\frac{\partial V}{\partial y} \tag{3.9}$$

With x as the independent variable, we use these equations to integrate the dependent variables $(y(x), p_x(x), p_y(x))$ from the small value of x found (after the change of sign) back to $x = 0$ using the special code **`rk41`**.
An alternative method **`xsection`** is also coded in which, rather than searching through a trajectory list for a change in sign of x, the result of each Runge-Kutta step is examined for a change in sign, and if found, **`goback`** is immediately called to integrate back to $x = 0$, and the new (y, p_y) pair found at $x = 0$ is added to a list.

`xsection` returns a list of two lists, the surface of section list first, and then the trajectory list. The function **`xsection_plot`** only accumulates the surface of section list, makes an immediate plot, and also returns the list for possible later use (such as combining surface of section results on the same plot eventually). Both of these functions call our homemade **`rk4_step`**.

4 Hénon-Heiles Trajectory and Surface of Section

The code file **`example2.mac`** contains the Maxima function **`trajectory(E,tf,dt)`**, in which **`E`** is the dimensionless energy, **`tf`** is the dimensionless final time for the integration (starting at **`t = 0`**), and **`dt`** is the Runge-Kutta step size. The initial value of **`x0`** is hard-wired to be **`x0 = 0`**.

`trajectory` expects **`0 < E <= 1/6 = 1.66667`** (to produce a confined trajectory), and interactively asks for the value of **`y0`** and **`py0`**. The value of **`px0`** is then calculated from conservation of energy:

$$E = \frac{1}{2}\,\left(p_{x0}^2 + p_{y0}^2\right) + V(x_0, y_0). \tag{4.1}$$

The required initial value of **y0** must satisfy

$$0 < V(0, y_0) < E, \tag{4.2}$$

(so that a zero-kinetic-energy particle at the initial position does not have the potential energy to allow escape from confinement in the potential well), which is equivalent to solving for the value of y such that

$$E = \frac{1}{2}\, y^2 \left(1 - \frac{2}{3}\, y\right) \tag{4.3}$$

We can use the Maxima function **realroots** to find the real roots of a polynomial. If **E = 0.1**, for example,

```
(%i1) fpprintprec:7$
(%i2) ratprint:false$
(%i3) float(realroots(0.1 -y^2/2 + y^3/3));
(%o3) [y = -0.39761,y = 0.56707,y = 1.330541]
```

The third root returned is a value of y in the $y > 1$ valley, which is irrelevant to motion in the central region around the origin (which is where our initial conditions set the particle).

We can then construct a function **yminmax(E)** which returns (as a list) **ymin** and **ymax**.

```
(%i4) yminmax(E) :=
block([soln,numer],numer:true,
   soln : float(realroots(E - y^2/2 + y^3/3 )),
   soln : map('rhs,soln),
   rest(soln, -1))$
(%i5) yminmax(0.1);
(%o5) [-0.39761,0.56707]
```

These values will be used to check the initial value of **y** used to begin the integration at **t = 0**. The user of the function **trajectory** will be asked to input an initial value of **y** inside a loop.

Here is a function **gety0(E)** which incorporates a **while cond do** loop design.

```
gety0(E) :=
block([ylim,ymin,ymax,ok,y0,numer],numer:true,
    ylim : yminmax(E),
    ymin : ylim[1],
    ymax : ylim[2],
    ok : false,
    while not ok do (
        print(" need ",ymin," < y0 < ",ymax),
        y0 : read(" input y0 "),
        if (y0 > ymin) and (y0 < ymax) then ok : true,
        if y0 < -10 then ok : true),
    y0)$
```

with the behavior

```
(%i6) gety0(0.1);
 need  -0.39761  < y0 <  0.56707
 input y0
-20;
(%o6) -20
(%i7) gety0(0.1);
 need  -0.39761  < y0 <  0.56707
 input y0
-1;
 need  -0.39761  < y0 <  0.56707
 input y0
1;
 need  -0.39761  < y0 <  0.56707
 input y0
0.095;
(%o7) 0.095
```

Given E and y_0, we then require p_{y0} such that even with $p_{x0} = 0$ there is not sufficient kinetic energy to escape confinement, or

$$\frac{1}{2}\, p_{y0}^2 < E - V(0, y_0) \tag{4.4}$$

so that (assuming $p_{y0} > 0$),

$$(p_{y0})_{max} = \sqrt{2\,(E - V(0, y_0))}. \tag{4.5}$$

We then require that $0 < p_{y0} < (p_{y0})_{max}$ in order to start the integration.

Given E, y_0, and p_{y0}, we then use conservation of energy Eq.(4.1) together with the assumption $p_{x0} > 0$ to determine p_{x0}.

Here is an example of the use of **`trajectory`**, taking $E = 0.1$, $y_0 = 0.095$, $p_{y0} = 0.096$, and $t_f = 20$.

```
(%i1) load(example2);
(%o1) "c:/k2/example2.mac"
(%i2) rkpts : trajectory(0.1, 20, 0.1)$
 need  -0.39761  < y0 <  0.56707
 input y0
0.095;
 need  0.0  < py0 <  0.43766
 input py0
0.096;
 E =  0.1  x0 =  0  y0 =  0.095
 px0 =  0.427  py0 =  0.096
 odeL =  [px,py,-2*x*y-1.0*x,1.0*y^2-1.0*y-x^2]
 varL =  [x,y,px,py]  initL =  [0,0.095,0.427,0.096]
   working ...
(%i3) fll(rkpts);
(%o3) [[0.0,0.0,0.095,0.427,0.096],
          [20.0,0.14835,0.24069,0.22181,-0.26375],201]
(%i4) xL : take(rkpts,2)$
(%i5) fll(xL);
(%o5) [0.0,0.14835,201]
(%i6) yL : take(rkpts,3)$
(%i7) fll(yL);
(%o7) [0.095,0.24069,201]
(%i8) plot2d([discrete,xL,yL],[xlabel,"x"],[ylabel,"y"],
          [style,[lines,2]],[x,-0.5,0.5],[y,-0.5,0.5])$
```

which produces the trajectory plot

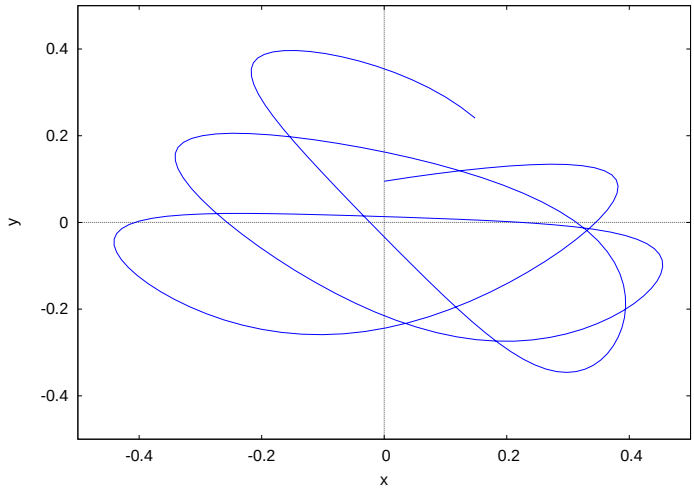

Figure 9: $E = 0.1$, $y_0 = 0.095$, $p_{y0} = 0.096$, $t_f = 20$

The code file **example2.mac** contains the function **getsection(pts, xtol)**, in which **pts** is the output list of integration points produced by **trajectory** and **xtol** is the precision requested to go back to an integration point at which $x = 0$.

The function **getsection** searches for integration points where **x** has changed sign, and calls **goback** (which calls **rk41**, designed to land on the last "x") to integrate backwards to **x = 0** to find the corresponding values of **y** and **py**. A list of these **[y,py]** values is returned by **getsection**.

```
(%i9) ypy : getsection(rkpts,0.01)$
   working...
(%i10) fll(ypy);
(%o10) [[0.095,0.096],[0.35446,-0.14214],7]
(%i11) plot2d([discrete, ypy],[xlabel,"y"],[ylabel,"py"],
          [style,[points,1,5,1]],[x,-0.6,0.6],[y,-0.6,0.6])$
```

which produces the "surface of section" plot

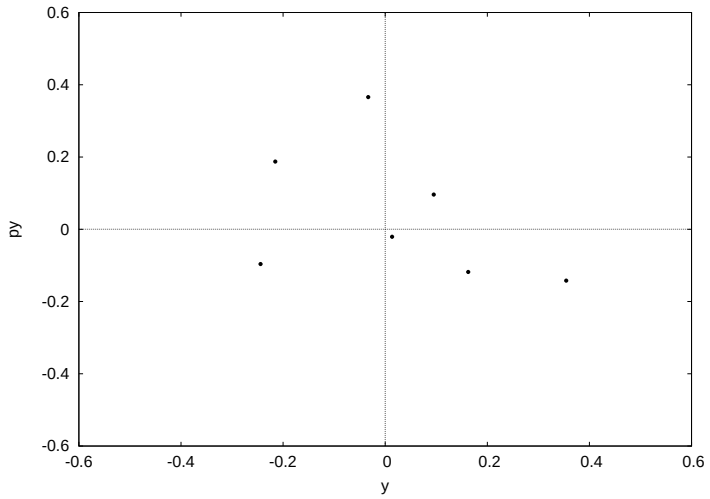

Figure 10: $E = 0.1$, $y_0 = 0.095$, $p_{y0} = 0.096$, $t_f = 20$

An alternative method is coded in the function **xsection(E,tf,dt,xtol)** in **example2.mac**. This alternative method calls **rk4_step** to perform a single Runge-Kutta step, and looks immediately for a sign change in x, and if found, calls **goback** to integrate backwards to recover the coordinates of the surface of section point corresponding to $x = 0$.

xsection returns a list with two elements, the first being the list of the surface of section coordinates (as returned by **getsection**), and the second being the list of Runge-Kutta points (as returned by **trajectory**).

This alternative method is used as follows:

```
(%i12) out : xsection(0.1,20,0.1,0.01)$
 need  -0.39761  < y0 <  0.56707
 input y0
0.095;
 need  0.0  < py0 <  0.43766
 input py0
0.096;
 E =  0.1  x0 =  0  y0 =  0.095
 px0 =  0.427  py0 =  0.096
  working...
(%i13) ypy : out[1]$
(%i14) rkpts : out[2]$
(%i15) xL : take(rkpts,2)$
(%i16) yL : take(rkpts,3)$
```

```
(%i17) plot2d([discrete,xL,yL],[xlabel,"x"],[ylabel,"y"],
          [style,[lines,2]],[x,-0.5,0.5],[y,-0.5,0.5])$
(%i18) plot2d([discrete, ypy],[xlabel,"y"],[ylabel,"py"],
          [style,[points,1,5,1]],[x,-0.6,0.6],[y,-0.6,0.6])$
```

and produces plots identical to those above.

The above results do not identify elliptic fixed points in the surface of section plot, so we use **`trajectory`** and **`getsection`** as above for the longer integration intervals $t_f = 500$ and $t_f = 1000$ which produce the following plots.

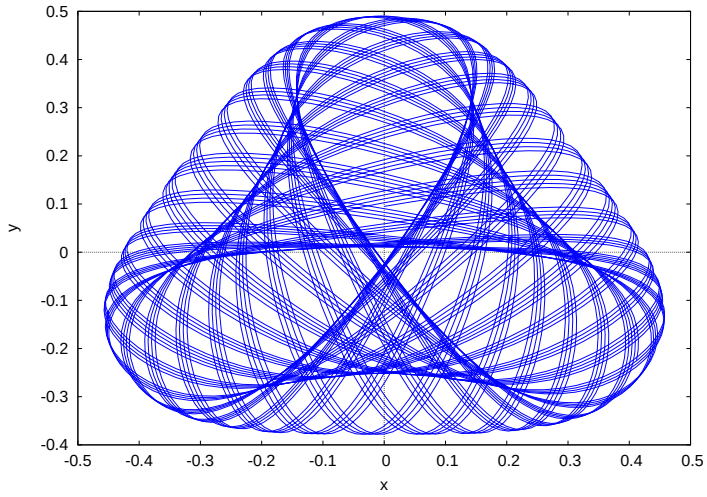

Figure 11: $E = 0.1$, $y_0 = 0.095$, $p_{y0} = 0.096$, $t_f = 500$

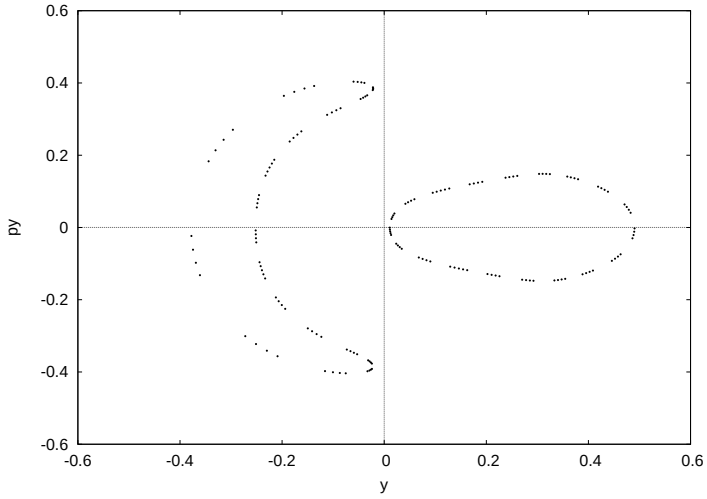

Figure 12: $E = 0.1$, $y_0 = 0.095$, $p_{y0} = 0.096$, $t_f = 500$

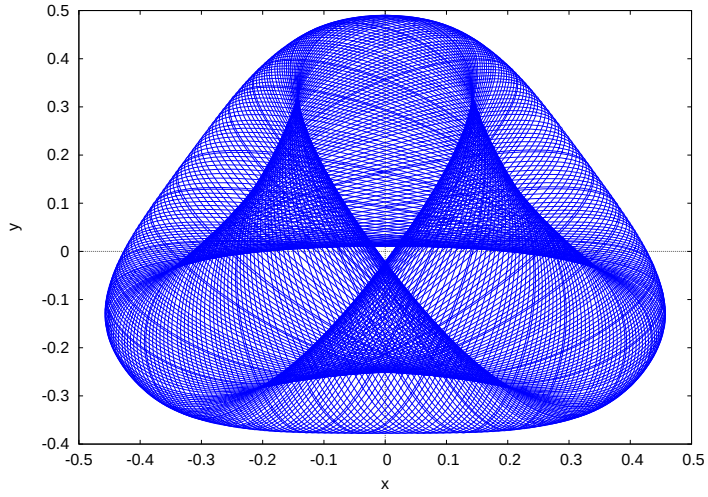

Figure 13: $E = 0.1$, $y_0 = 0.095$, $p_{y0} = 0.096$, $t_f = 1000$

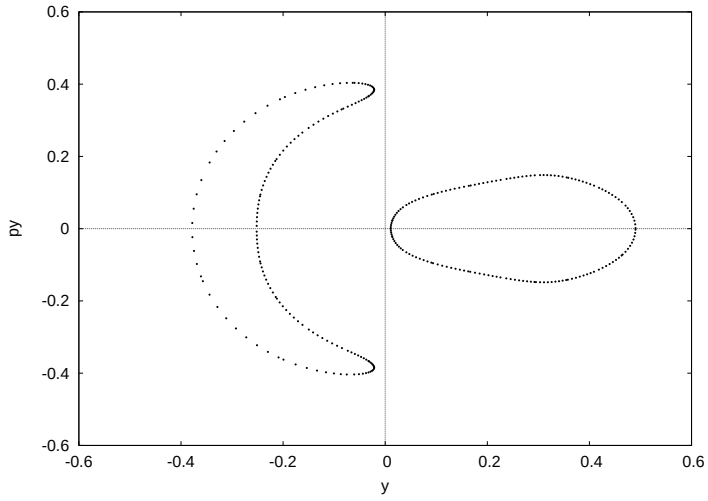

Figure 14: $E = 0.1$, $y_0 = 0.095$, $p_{y0} = 0.096$, $t_f = 1000$

A pair of elliptic fixed points which both have $p_y = 0$ and whose y values have opposite signs are now apparent.

5 Sensitivity of the Section Plot to Δp_{y0}

When exploring the sensitivity of the surface of section plot to changes in one or more initial conditions, it is convenient to code a function **`xsection_plot(E,y0,py0,tf)`** in which the starting parameters are arguments to the function (which are checked by the code) so we can avoid the interactive entry delays.

In addition, we want an automatic plot of the resulting surface of section, and we also want this function to return the list of the surface of section points for later use. This function can avoid saving the trajectory points and just return the surface of section points.

In the following we use **`xsection_plot`** to get enough surface of section examples to illustrate how changes in the initial value of p_y (holding E and y_0 constant) can give rise to new elliptic fixed points and to hyperbolic fixed points, as well as transitional cases and additional structure.

We also show two examples of small changes in p_{y0} leading to significant structural changes in the surface of section plot.

The first plot shows two cases of a pair of elliptic fixed points having $p_y = 0$ and y values with opposite signs. Notice the additional structure with another set of sub-elliptic fixed points appearing. We bind the surface of section coordinate lists to symbols, such as **ypy1**, to be able to combine with other cases on a future plot. When initially exploring, we don't know in advance which cases will be combined, since we don't know in advance what the plot will look like.

```
(%i1) load(example2);
(%o1) "c:/k2/example2.mac"
(%i2) ypy1 : xsection_plot(0.1,0.095,0.096,1000)$
(%i3) ypy2 : xsection_plot(0.1,0.095,0.03,800)$
(%i4) plot2d([[discrete,ypy1],[discrete,ypy2]],
         [style,[points,1]],[color,blue,red],
         [xlabel,"y"],[ylabel,"py"],[legend,"0.096","0.03"])$
```

which produces the plot

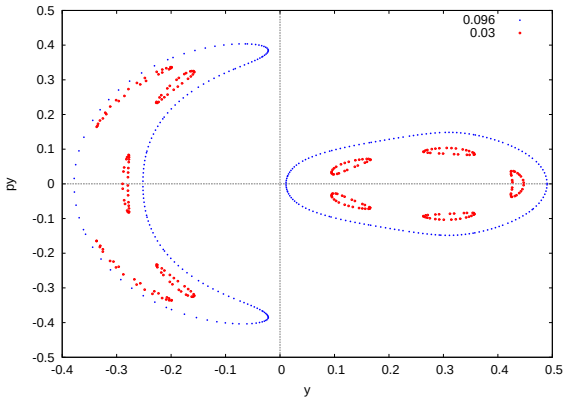

Figure 15: $E = 0.1$, $y_0 = 0.095$, Two Values of p_{y0}

Two cases of a pair of elliptic fixed points having centers on the $y = 0$ axis and corresponding to positive and negative values of p_y are shown next. Notice the additional structure with a set of sub-elliptic fixed points appearing.

```
(%i5) ypy3 : xsection_plot(0.1,0.095,0.2,800)$
(%i6) ypy6 : xsection_plot(0.1,0.095,0.301,800)$
(%i7) plot2d([[discrete,ypy3],[discrete,ypy6]],
         [style,[points,1]],[color,blue,red],
         [xlabel,"y"],[ylabel,"py"],[legend,"0.2","0.301"])$
```

which produces

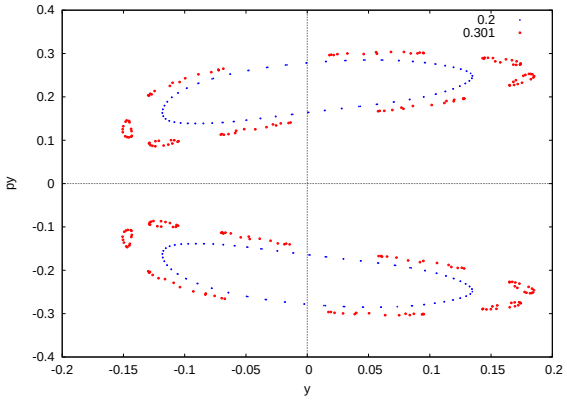

Figure 16: $E = 0.1$, $y_0 = 0.095$, Two Values of p_{y0}

We next show a transitional case and a case with three (loose) hyperbolic fixed points .

```
(%i8) ypy5 : xsection_plot(0.1,0.095,0.33,800)$
(%i9) ypy8 : xsection_plot(0.1,0.095,0.32018,800)$
(%i10) plot2d([[discrete,ypy8],[discrete,ypy5]],
         [style,[points,1]],[color,blue,red],
         [xlabel,"y"],[ylabel,"py"],[legend,"0.32018","0.33"])$
```

which produces

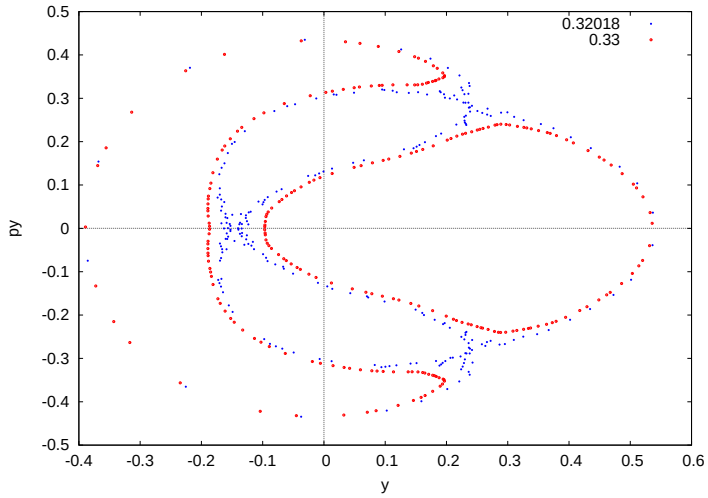

Figure 17: $E = 0.1$, $y_0 = 0.095$, Two Values of p_{y0}

Here is a first example of small changes in p_{y0} producing significant structure changes in the surface of section plot.

```
(%i11) ypy9 : xsection_plot(0.1,0.095,0.047,500)$
(%i12) ypy10 : xsection_plot(0.1,0.095,0.049,500)$
(%i13) plot2d([[discrete,ypy10],[discrete,ypy9]],
         [style,[points,1]],[color,blue,red],
         [xlabel,"y"],[ylabel,"py"],[legend,"0.049","0.047"])$
```

which produces

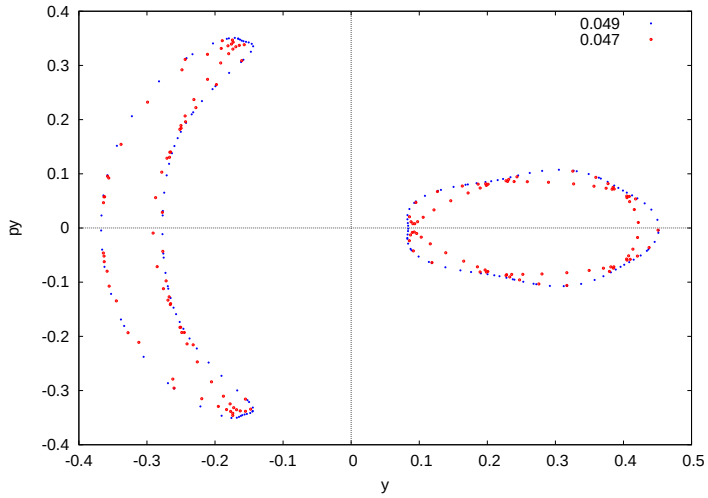

Figure 18: $E = 0.1$, $y_0 = 0.095$, Two Close Values of p_{y0}

Here is a second example of close values of p_{y0} leading to large structural changes in the surface of section plot.

```
(%i14) ypy14 : xsection_plot(0.1,0.095,0.32,500)$
(%i15) ypy15 : xsection_plot(0.1,0.095,0.31,500)$
(%i16) plot2d([[discrete,ypy15],[discrete,ypy14]],
         [style,[points,1]],[color,blue,red],
         [xlabel,"y"],[ylabel,"py"],[legend,"0.31","0.32"])$
```

which produces the plot

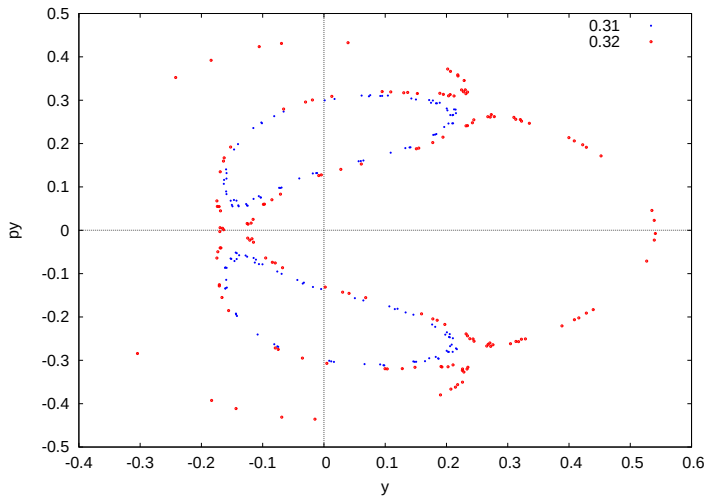

Figure 19: $E = 0.1$, $y_0 = 0.095$, Two Close Values of p_{y0}

6 Trajectory and Surface of Section Plots Using R

As in our Maxima code, we have a interactive **R** function **gety0(E)** in a code file **example2.R**:

```
gety0 = function(E) {
    ylim = yminmax(E)
    ymin = ylim[1]
    ymax = ylim[2]

    ok = FALSE
    while (! ok) {
       cat(" need ",ymin," < y0 < ",ymax,"\n")
       y0 = as.numeric(readline(" input y0: "))
       if ((y0 > ymin) && (y0 < ymax)) ok = TRUE
       if (y0 < -10) ok = TRUE}
    y0}
```

As in our Maxima code, we have a **R** function **yminmax(E)** in the same code file **example2.R**:

```
##  yminmax(E) assumes KE = 0 and x = 0
##    to find limits on initial y from energy conservation

yminmax = function(E) {
    mroots = sort(Re(polyroot(c(E,0,-1/2,1/3))))
    mroots[1:2]}
```

Likewise, a function **get_py0(pymax)**.

Since we use the fixed step homemade 4'th order Runge-Kutta code **myrk4** (also included in **example2.R**), we have a simplified form of a derivatives function which returns a **R** vector:

```
##  global derivatives function in proper form for myrk4
##  y[1] = x, y[2] = y, y[3] = px, y[4] = py

henon_heiles = function(t,y) {
  with( as.list(y),  {
     dx = y[3]
     dy = y[4]
     dpx = -2*y[1]*y[2] - y[1]
     dpy = y[2]^2 - y[2] - y[1]^2
     c(dx,dy,dpx,dpy)})}
```

We have defined **myrk4** to expect an external global list of times, such as **tL = seq(0,500,0.1)**. This global **R** vector **tL** of times must be defined before calling **trajectory(E)** which returns the solutions **list(xL,yL,pxL,pyL)**. The last line of **trajectory** is **myrk4(initL,tL,henon_heiles)}**.

The **R** function **getsection(rkvecs,xtol)** can be immediately used with the list output of **trajectory** to produce the list of the surface of section vectors **list(ysL,pysL)**.

```
> source("example2.R")
> tL = seq(0,800,0.1)
> ##  rk1 and ypy1 for py0 = 0.096
> ##  (with E = 0.1, y0 = 0.095)
> rk1 = trajectory(0.1)
 need  -0.3976099  < y0 <  0.5670689
 input y0: 0.095
 need  0  < py0 <  0.4376604
 input py0 0.096
 E =  0.1  x0 =  0  y0 =  0.095
 px0 =  0.4270019  py0 =  0.096
> ypy1 = getsection(rk1,0.01)
 ns =  251
> yL1 = ypy1[[1]]
> pyL1 = ypy1[[2]]
> ## rk2 and ypy2 for py0 = 0.03
> rk2 = trajectory(0.1)
 need  -0.3976099  < y0 <  0.5670689
 input y0: 0.095
 need  0  < py0 <  0.4376604
 input py0 0.03
 E =  0.1  x0 =  0  y0 =  0.095
 px0 =  0.4366309  py0 =  0.03
> ypy2 = getsection(rk2,0.01)
 ns =  256
> yL2 = ypy2[[1]]
> pyL2 = ypy2[[2]]
> ## show SOS points for ypy1 case
> plot(yL1,pyL1,pch=19,xlab="y",ylab="py",cex=0.4)
> ## add SOS points for ypy2 case
> points(yL2,pyL2,pch=19,col = "red",cex=0.4)
> mygrid()  ## homemade grid function in example2.R
> legend("topright",
+   legend = c("black - py0 = 0.095","red - py0 = 0.03"),cex=1.5)
```

which produces the plot

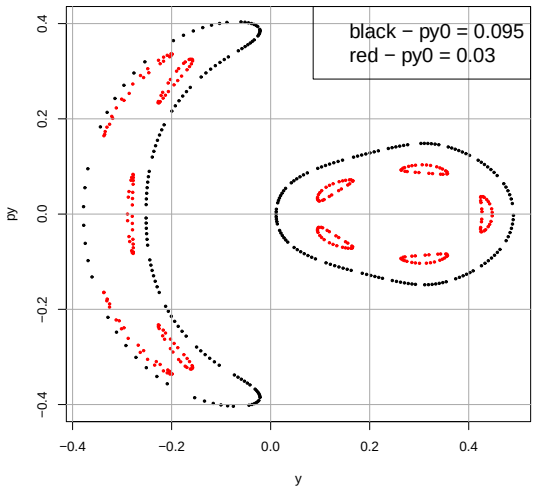

Figure 20: $E = 0.1$, $y_0 = 0.095$, $p_{y0} = 0.095$, $p_{y0} = 0.03$

We still have access to the lists of solution vectors **rk1** and **rk2**, which we now use to get the **xL** and **yL** vectors in order to plot an actual spatial trajectory of the particle.

```
> xL1 = rk1[[1]]
> fll(xL1)
  0   0.07211231   8001
> yL1 = rk1[[2]]
> fll(yL1)
  0.095   0.1080686   8001
> plot(xL1,yL1,type="l",xlab="x",ylab="y")
```

which produces

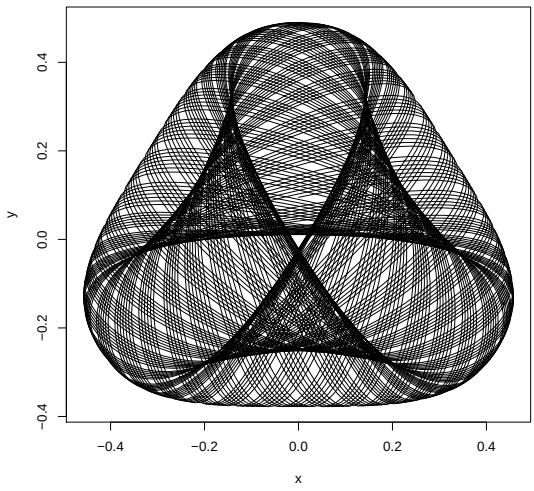

Figure 21: $E = 0.1$, $y_0 = 0.095$, $p_{y0} = 0.095$

Likewise with case 2 and the list **rk2**:

```
> xL2 = rk2[[1]]
> fll(xL2)
  0   -0.08530082   8001
> yL2 = rk2[[2]]
> fll(yL2)
  0.095   0.4267989   8001
> plot(xL2,yL2,type="l",xlab="x",ylab="y")
```

which produces

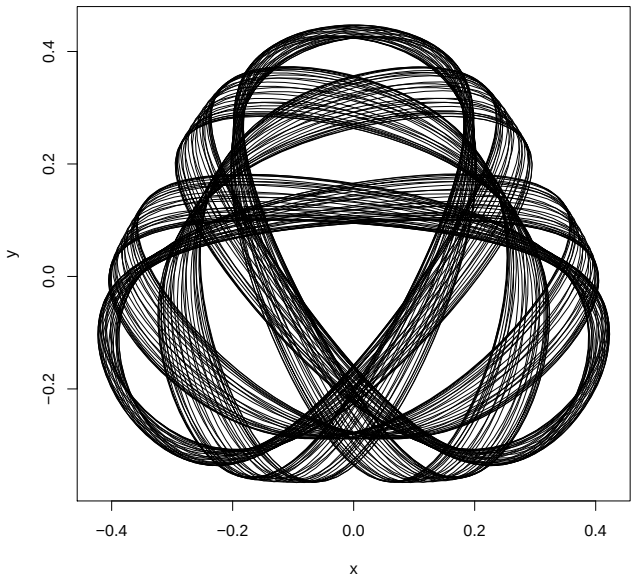

Figure 22: $E = 0.1$, $y_0 = 0.095$, $p_{y0} = 0.03$

7 Combining Surface of Section Plots with R

The **R** function **xsection_plot(E,y0,py0)** (in the file **example2.R**) was used, with an external definition of **tL**, repeatedly with different values of **py0**, to identify interesting values of **py0**.

Since that function not only makes an immediate plot but also returns **list(yL,pyL)**, one should use the syntax (for example)

```
> tL = seq(0,1500,0.1)
> ypy1 = xsection_plot(0.1,0.095,0.096)
   ns =  470
> ypy2 = xsection_plot(0.1,0.095,0.03)
   ns =  480
```

The file **hhplota.R** was created to automatically build up a plot of surface of section points for fixed E, and fixed y_0, and variable p_{y0}.

```
##   hhplota.R
##   combine surface of section plots
##  21 values of py
E = 0.1
y = 0.095
```

```
py = 0.096
tL = seq(0,1000,0.1)
ypy = xsection(E, y, py)
plot(ypy[[1]],ypy[[2]], pch=19, cex=0.05, xlab="y",
         ylab="py", xlim=c(-0.5,0.6), ylim=c(-0.6,0.6))
pyval = c(0.005,0.02, 0.03, 0.032, 0.04,0.05, 0.06, 0.07,
            0.08, 0.09,0.1, 0.2, 0.25, 0.301, 0.302, 0.303,
            0.32018, 0.3202, 0.33, 0.4)
for (py in pyval) {
   ypy = xsection(E, y, py)
   points(ypy[[1]],ypy[[2]],pch=19,cex=0.05)}
```

A **R** function **xsection(E,y0,py0)** (in the file **example2.R**) was used in this script. **xsection(E,y0,py0)** is a simplified version which does not itself make a plot, but just returns the surface of section points **list(yL,pyL)**. (It also does not do checks on initial values.) One can then watch the plot gradually build up with new initial values added:

```
> source("example2.R")
> source("hhplota.R")
 ns =  314
 ns =  322
 etc., etc.
```

to produce the plot corresponding to those 21 values of **py0**:

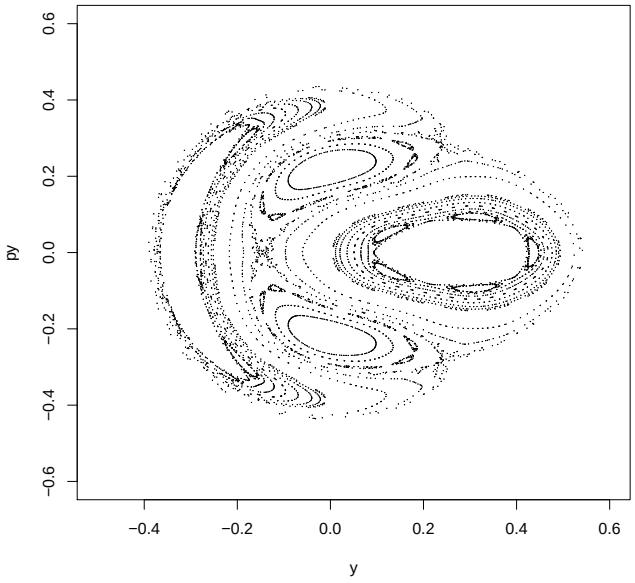

Figure 23: $E = 0.1$, $y_0 = 0.095$, 21 values of p_{y0}

8 A Single Initial Condition Integrated for a Long Time

We integrate for a long time (32,000 "sec") at the maximum confined energy $E = 1/6$ and for the single case $y_0 = 0$ and $p_{y0} = 0$.

```
> tL = seq(0,32000,0.1)
> ypy = xsection_plot(1/6,0,0)
> plot(ypy[[1]],ypy[[2]],pch=19,cex=0.05,xlab="y",
+          ylab="py",xlim=c(-0.6,1),ylim=c(-0.6,0.6))
```

which produces the plot

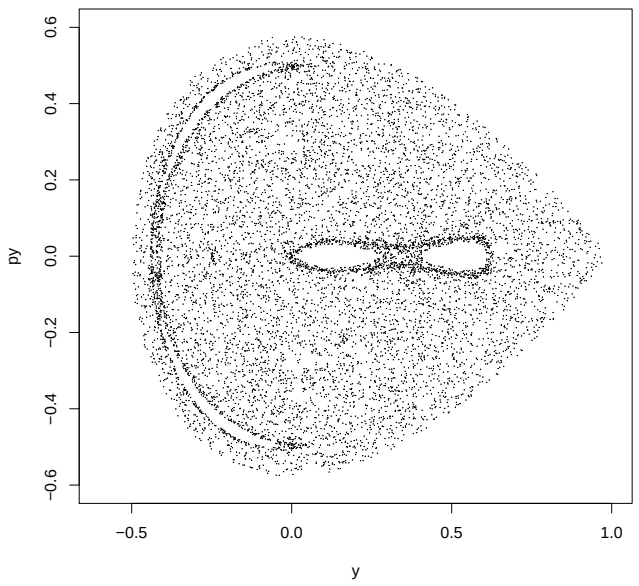

Figure 24: $E = 1/6$, $y_0 = 0$, $p_{y0} = 0$

which illustrates a case of chaotic behavior.

9 Watching Section Points Accumulate in “Real Time”

The use of **`xsection_plot`** above produced the surface of section plot with all the points appearing together suddenly, and hence with no sense of the history of the points appearing in different locations.

An alternative is to plot each surface of section point as soon as its location is determined. The function **`plot_sos(E,y0,py0,cex_val)`** calls **`section_plot(rkvecs,xtol,cex_val)`** which plots each section point as it is found. (In addition the function returns **`list(ysL,pysL)`**.) The syntax to use is, for example,

```
> tL = seq(0,5000,0.1)
> ypy = plot_sos(1/6,0,0,0.1)
 ns =  1542
```

which allows one to watch the time development of the surface of section points. The x and y axis limits are hardwired into the code of **`section_plot`**, as is a sleep command to slow down the development of the plot.

```
ysL[1] = rkvecs[[2]][1]
pysL[1] = rkvecs[[4]][1]
## plot first SOS point
plot(ysL[1], pysL[1], pch=19, cex = cex_val, xlab="y",
         ylab="py", xlim=c(-0.6,1), ylim=c(-0.6,0.6))
## and later in the code:
ysL[k] = rgoback[1]
pysL[k] = rgoback[2]
points(ysL[k], pysL[k],pch=19, cex = cex_val)
Sys.sleep(0.01)
```

and would need to be edited to fit the problem being investigated or incorporated as additional arguments.

10 Investigating Accuracy of Integration and Surface of Section Points

The following general suggestions do not take into account the special behavior one confronts when the system exhibits chaotic behavior and is very sensitive to not only the initial conditions but also the order of the arithmetic operations needed to calculate the trajectory.

10.1 Decrease the Step Size

Halving the step size in the integration initially done by **`myrk4`**, and also halving the value used for **`xtol`** when calling **`goback`** and then comparing the results with the original step size **`0.1`** and **`xtol=0.01`** will give an indication of the reliability of results produced.

10.2 Integrate Backwards

When integrating over many time steps, one should check the accuracy of the integrator by taking the final **`[t,x,y,px,py]`** value, use it as the initial time and phase space point and integrate backwards (with the same step size) to **`t = 0`**, and compare the resulting values of **`[x,y,px,py]`** with the original **`t=0`** values used. Ideally, the values should closely agree.

10.3 Watch the Values of the Energy During the Integration

We know theoretically that the total energy (kinetic plus potential) remains a constant of the motion. Lack of accuracy in the integration will show up in changes in the computed energy **`(pxˆ2 + pyˆ2)/2 + V(x,y)`**. This sum can be retrieved from the output of **`trajectory`** and plots of the energy versus time can reveal a lack of accuracy.

10.4 Use More Accurate Integrators

An obvious extension of what has been done here is to make use of the **deSolve** package integrators in **`R`**. Note that the derivatives function needed with **`ode`** has a different structure than that needed with our homemade **`myrk4`**.

11 Suggestions for Further Exploration

In addition to the accuracy checks mentioned in the previous sections, we quote Steven Koonin's suggestions:

> Use the code to construct surfaces of section for the Hénon-Heiles potential at energies ranging from 0.025 to 0.15 in steps of 0.025. For each energy, consider various initial conditions and integrate each trajectory long enough in time (some will require going to $t \approx 1000$) to map out the surface-of-section adequately. For each energy, see if you can find the elliptic fixed points, the tori (and tori of tori) around them, and the chaotic regions of phase space and observe how the relative proportions of each change with increasing energy. ....