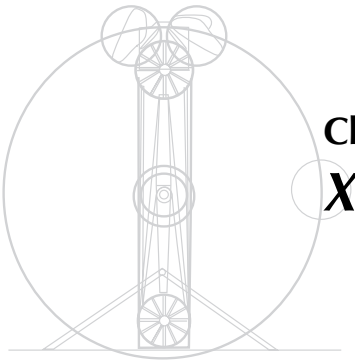

Chapter 4
XPPAUT in the Classroom

XPPAUT, although developed as a tool for researchers, also has been used by many instructors in a variety of courses to allow students to look at models and differential equations on the computer. Since the program is free and runs on a variety of platforms, it is suitable for classroom use. While the interface is not as simple as some dedicated systems such as Java™ applets or the Macintosh program, MACPHASE, it is flexible and, with a little thought by the instructor, it becomes relatively easy to write ODE files illustrating all the standard figures in most textbooks. I have used *XPPAUT* with many differential equations and modeling texts at both undergraduate and graduate levels. Some of these have been specialty courses like computational neuroscience or quantitative cardiac physiology. However, I will not delve into such specialized applications in this chapter. Rather, you can go to the *XPPAUT* homepage and see some examples from various advanced courses.

I will start with some simple examples using *XPPAUT* to create plots in two and three dimensions without any reference to differential equations. I will then consider a variety of topics in first order discrete dynamical systems. I will illustrate how to make cobweb diagrams, draw bifurcation diagrams, compute rotation numbers, and create diagrams of the maximal Liapunov exponent. I will then demonstrate how to use *XPPAUT* to make Julia sets and the famous Mandelbrot set. In the next subsection, I will discuss one-dimensional nonautonomous differential equations and, finally, planar dynamical systems.

4.1 Plotting functions

Suppose that you want to plot the function of one variable $f(x)$ from x_0 to x_1. Then the following ODE file will do the trick:

```
# plot1.ode
# plot f(x)
par xlo=-2,xhi=2
f(x)=x*(1-x^2)
s'=1
x_=xlo+s*(xhi-xlo)
```

```
aux y=f(x_)
aux x=x_
@ xp=x,yp=y
@ xlo=-2,xhi=2,ylo=-4,yhi=4
@ total=1.001,dt=.01
done
```

Most of this file is for defining the plotting area and numerics. To change the ranges of plotting, just vary the parameters `xlo`, `xhi` and, to change the function, just click on `File Edit Functions` and change the function. To keep the original graph and subsequent graphs, click on `Graphics Freeze On freeze`, and up to 26 curves can be kept.

Here is an example in which the first five terms of the Taylor series for the function $\sin(x)$ are plotted on the interval $[-\pi, \pi]$:

```
# sintayl.ode
# first 5 terms of the Taylor series for sin
z1=x
z2=z1-x^3/6
z3=z2+x^5/120
z4=z3-x^7/5040
z5=z4+x^9/362880
x'=2*pi
init x=-3.1415926
aux y=sin(x)
aux y[1..5]=z[j]
@ total=1,dt=.005
@ xlo=-3.15,xhi=3.15,ylo=-1,yhi=1
@ nplot=6,xp=x,yp=y
@ yp[2..6]=y[j-1],xp[j]=x
done
```

Note. By defining a series of internal variables, `z1, ...` I can add just the required new terms sequentially. The last line tells *XPPAUT* that all these quantities should be plotted. They will appear as different colored curves.

You can easily use *XPPAUT* to make polar coordinate plots of the form $r = f(\theta)$ (or $\theta = f(r)$) as follows:

```
# polarpl.ode
# polar plots
f(z)=1+cos(z)
theta'=1
init theta=-12
r=f(theta)
aux x=r*cos(theta)
aux y=r*sin(theta)
@ total=25
@ xp=x,yp=y,xlo=-2,xhi=2,ylo=-2,yhi=2
done
```

There is a parameter a so that one could plot this over a range of values of a. Edit the function to get some other polar curves.

Other parametric plots are just as easy to plot. For example, the following plots $(x(t), y(t))$ in the plane:

```
# parametric.ode
f(t)=cos(a*t)
g(t)=sin(b*t)
par a=4,b=1
s'=1
aux x=f(s)
aux y=g(s)
@ xp=x,yp=y,xlo=-2,ylo=-2,xhi=2,yhi=2
@ total=50
done
```

Similarly, three-dimensional parametric plots are also possible. Here is one where I have set up all the axes. Change parameters and plot away:

```
# param3d.ode
# 3d parametric plots
f(t)=cos(a*t)
g(t)=sin(b*t)
h(t)=sin(c*t+d)
par a=4,b=1,c=2,d=.75
s'=1
aux x=f(s)
aux y=g(s)
aux z=h(s)
@ xp=x,yp=y,zp=z,xlo=-2,ylo=-2,xhi=2,yhi=2
@ axes=3d
@ xmax=1.25,ymax=1.25,zmax=1.25,xmin=-1.25,ymin=-1.25,
zmin=-1.25
@ total=50
done
```

As a last example of plotting in three-dimensions, here is a nice trick in which I plot the three-dimensional parametric plot along with a two-dimensional projection below it:

```
# param3d2.ode
# 3d parametric plots with 2d projection
f(t)=cos(a*t)
g(t)=sin(b*t)
h(t)=sin(c*t+d)
par a=1.34,b=1.12,c=2.28,d=.75
par zlo=-4
s'=1
aux x=f(s)
```

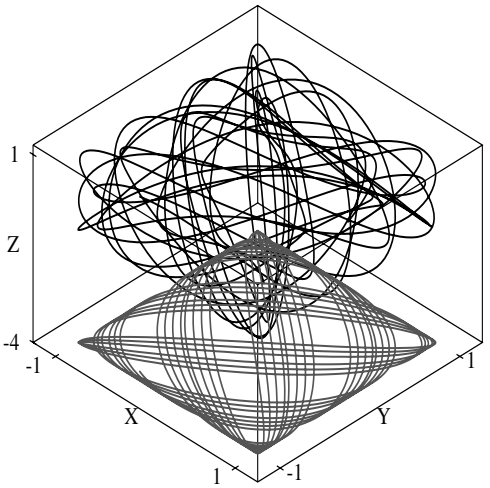

Figure 4.1. *Three-dimensional plot illustrating a projection on the coordinate plane.*

```
aux y=g(s)
aux z=h(s)
aux zplane=zlo
@ xp=x,yp=y,zp=z,xlo=-2,ylo=-2,xhi=2,yhi=2
@ axes=3d
@ xmax=1.25,ymax=1.25,zmax=1.25,xmin=-1.25,ymin=-1.25,zmin=-4
@ nplot=2,xp2=x,yp2=y,zp2=zplane
@ total=100
done
```

I have added another plot in which I keep the z-value constant. This is a useful trick and lets one plot a variety of interesting projections against the coordinate planes. Only `zmin` and the parameter `zlo` need to be changed in order to change the position of the planar projection. By plotting z, y and holding x at some fixed value, you can similarly plot the (y, z) projection. See Figure 4.1 for the result.

4.2 Discrete dynamics in one dimension

Many discussions of dynamical systems begin with the study of one-dimensional maps. These have the form

$$x_{n+1} = f(x_n).$$

One of the standard ways to illustrate the dynamics of these maps is to draw a cobwebbing diagram (see Figure 4.2). This is a plot of x_{n+1} versus x_n along with the line $y = x$ and the function $y = f(x)$. *XPPAUT* does not have a specific cobwebbing function built into it, but it is very easy to fool the program into making a cobweb plot. Every other point shares either the same x value or the same y value, thus the cobweb plot consists of a series of

horizontal and vertical lines. Once this is done, you also want to plot the function and the line $y = x$. Here is an example cobweb map for the logistic function:

```
# cobweb.ode
# way to fool XPP into cobwebbing
# First I define a function that every other step evaluates
# the map.  In the alternate steps, it just keeps the same
# value so that it alternates between horizontal and vertical
# jumps
g(x,y)=if(mod(t,2)<.5)then(f(x))else(y)
# note that 't' is the iteration number 0,1,2,...
# if t is even evaluate f otherwise keep the old y
y(t+1)=g(x,y)
x(t+1)=if(t==0)then(x)else(y)
# note that x(t+2)=f(x(t)) so every other point is the map!
f(x)=a*x*(1-x)
par a=3.95
# these are just useful for plotting f(x),x
par xlo=0,xhi=1,nit=25
#
init y=0,x=.25
# always start y=0
#
# here I create scaled x-values to plot f(x)
xx=xlo+(xhi-xlo)*t/nit
aux map=f(xx)
aux st=xx
# some convenient settings for the graphics
@ xlo=0,ylo=0,xhi=1.001,yhi=1.001
@ xp=x,yp=y
@ nplot=3
# add the plots y=x and y=f(x)
@ xp2=st,yp2=st
@ xp3=st,yp3=map
# tell xpp that it is discrete and iterate 25 times
@ meth=discrete,total=25
done
```

This may seem a bit involved, but it works and it is very easy to adapt to other functions. Simply edit the function $f(x)$ and perhaps the ranges of the map `xlo,xhi`. Run this for a variety of values of $a < 4$. Try the map $f(x) = \mathrm{mod}(x + 3/4 + a \sin 2\pi x, 1)$ for $a < 1/2$.

4.2.1 Bifurcation diagrams

Another classic picture that is shown in many books is the bifurcation diagram for the logistic map. This is done by plotting the parameter a along the x-axis and the iterates (after

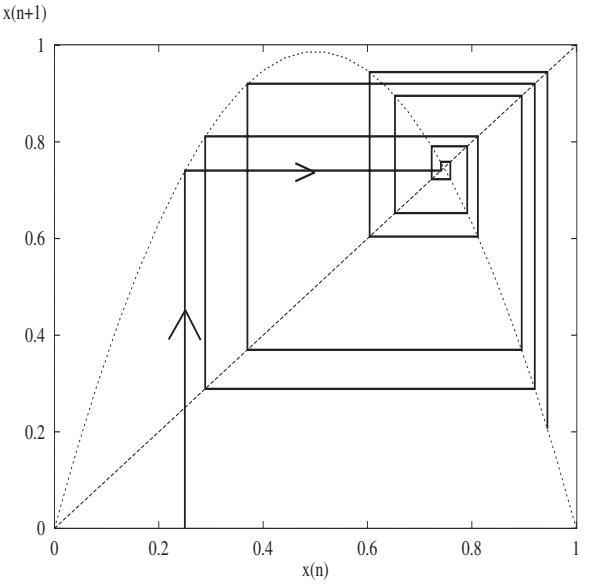

Figure 4.2. *A cobweb plot of the logistic map.*

transients) of the map along the y-axis. To do this in *XPPAUT*, you will want to make the parameter a variable and then use the `Range Integration` option to draw the diagram. Here is the *XPPAUT* file:

```
# logbif.ode
# the logistic map bifurcation diagram
f(x)=a*x*(1-x)
x'=f(x)
a'=a
init x=.1
init a=2
@ maxstor=100000,total=500,trans=350,meth=discrete
@ xlo=2,xhi=4.001,ylo=0,yhi=1.001,xp=a,yp=x
done
```

Notes. I have set `maxstor=100000` so that *XPPAUT* will store many points. I throw away the first 350 iterates to avoid transients. I treat the parameter as a variable so it can be plotted; parameters are not generally allowed to lie on axes.

Run this file and then do the following. First click on `Graphics Edit` and select `0`. Then, in the resulting dialog box, change the linetype from 1 to 0. This makes *XPPAUT* plot just points rather than lines. Now we can draw the diagram. Click on `Initialconds Range` and fill in the dialog box as follows:

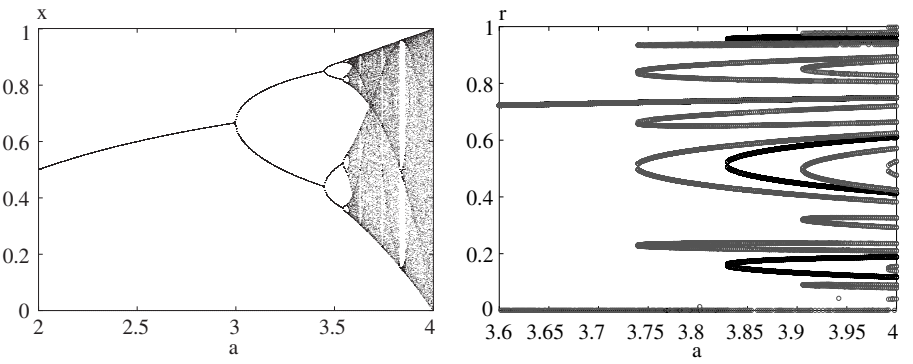

Figure 4.3. *The logistic map showing an orbit diagram on the left and a plot of the periodic orbits of period 3 and period 5 on the right.*

Range over: a
Steps: 200
Start: 2
End: 4
Reset storage: N

and click on `Ok`. You should see the bifurcation diagram appear. You can put more points in it by changing the number of steps from 200 to 500. The `Range` command allows you to range over either parameters or initial data and save the results. Figure 4.3 shows the orbit diagram we have just computed.

4.2.2 Periodic points

The "bifurcation" diagram constructed above is actually more correctly an orbit map. That is, it is simply a record of the *stable* solutions for any particular choice of parameters. A true bifurcation diagram should show solutions which are both stable and unstable. For example, one can ask where all the period 3 points are for the map. Period 3 is notable since it implies that there are periodic points for every possible period. This is one of the standard measures of chaotic behavior. How can one find the periodic points of a map as a function of a parameter? Consider a fixed value of the parameter and the third iterate of the map. Since the periodic points are not necessarily stable (and in fact are unstable), we need a method of finding them that is independent of stability. Newton's method provides a good way to search for them, as the convergence of the iterates is independent of the stability. Thus, for a fixed parameter value, we could sweep through a range of starting values and plot those for which the Newton's iterates converge. This shotgun approach has the advantage that we can pick up new branches of orbits as the parameter changes. In *XPPAUT*, there is a way to run a two-parameter range of simulations. This allows us to sweep through both the parameter and the state space. Given the map $f(x)$, a point of period p satisfies $f^p(x) = x$. (Here, $f^2(x)$ means $f(f(x))$, and so on.) Let $g(x) = x - f^p(x)$. We need to find roots of

$g(x)$. Newton's method provides an iterative scheme

$$r_{n+1} = r_n - g(r_n)/g'(r_n).$$

This rapidly converges if we have a close starting guess. But by sweeping over many values of r we are almost assured of catching all of them. Since we are solving for the roots numerically, we want to make sure that convergence to a true zero has occurred. Thus, if $|g(r)|$ is less than a specified tolerance, we will assume that we have found a root. Finally, we compute $g'(r)$ numerically to simplify the calculation. With these preliminaries, here is an *XPPAUT* file that will let you compute periodic points of any specified period:

```
# logper.ode
# finds the periodic points of the logistic map
#
f(x)=a*x*(1-x)
# recursively define pth iterate
ff(x,p)=if(p<=0)then(x)else(ff(f(x),p-1))
# find zeros of g
g(x)=x-ff(x,p)
#
q=g(r)
# Newtons method with numerical derivative
r'=r-eps*q/(g(r+eps)-q)
a'=a
# if within tol of root, then OK
aux err=tol-abs(q)
par p=3,eps=.000001,tol=1e-7
@ meth=discrete,total=20,maxstor=50000
@ xp=a,yp=r,xlo=3,xhi=4.0001,ylo=0,yhi=1.00001
done
```

Notes. I have defined the pth iterate of f recursively. The main iteration is the implementation of Newton's method. I define a fixed variable `q` since I need to refer to it three times, and thus the computation is sped up. The parameter `tol` gives the tolerance that I require for a root. The parameter `eps` is for computing the numerical derivative

$$g'(x) \approx \frac{g(x+\epsilon) - g(x)}{\epsilon}.$$

Let's use this to compute the period 3 bifurcation curves. Run *XPPAUT*. Click on `Graphics Edit` (**G E**) and choose curve `0`. Change the linetype to `0` so that dots will be drawn, and click `Ok`. Next, click on `Numerics Poincare map Section`. Fill in the dialog box as follows:

`Variable: err`
`Section: 0`
`Direction: 1`
`Stop on section: y`

This tells *XPPAUT* to only plot the points once the quantity `err` crosses zero. This means that the value of the root is within the specified tolerance. It also guarantees that Newton iterates which do *not* converge to a root will not be plotted. Now click on `Esc` to exit the numerics menu. We now will set up the two-parameter range integration. Click on `Initialconds 2 Par range` (**I 2**) to get a rather unwieldy dialog box. Fill this in as follows:

`Vary1: R`	`Reset storage: N`
`Start1: 0`	`Use old ic's: Y`
`End1: 1`	`Cycle color: N`
`Vary2: a`	`Movie: N`
`Start2: 3`	`Crv(1) Array(2): 2`
`End2: 4`	`Steps2: 200`
`Steps: 50`	`:`

This dialog box tells *XPPAUT* which pair of parameters you want to vary. The entry `Crv(1) Array(2)` allows you to vary the two parameters together or independently. Click on `Ok`, and the period 3 points will be plotted. If you want to compute at a finer resolution, increase the number of steps for the bifurcation parameter (`Steps2`). If you compute for period 2 points, then you should also increase the number of steps for the initial guesses (`Steps2`), as there are many such points and you will want to get them all. To save these points and superimpose a diagram with other periodic points, click on `Graphics Freeze Freeze` (**G F F**) and choose `-1, -2, ..., -9` for the `color` entry to make *XPPAUT* use points instead of lines. Figure 4.3 shows the periods 3 and 5 points.

4.2.3 Liapunov exponents for one-dimensional maps

One of the hallmarks of chaos is the local exponential divergence of nearby points under the dynamics. For one-dimensional maps, this is easy to measure. One needs to look at the average of the logarithm of the absolute value of the derivative of f evaluated at a point on the attractor. Thus, for a one-dimensional map, the maximal Liapunov exponent is

$$\lambda = \lim_{N\to\infty} \frac{1}{N} \sum_{j=1}^{N} \ln |f'(x_j)|.$$

If $\lambda < 0$, then the attractor is asymptotically stable. If $\lambda = 0$, the attractor is a periodic orbit. Finally, if the attractor is chaotic, then $\lambda > 0$. For the logistic map with $a = 4$ it is possible to show that $\lambda = \ln 2 \approx 0.693147$. The following *XPPAUT* file illustrates how to compute this exponent as a function of the parameter. I keep a running sum of the log of the derivative after throwing out the first 100 iterates. I compute the running average of this sum and keep only the final value computed after 2000 iterates:

```
# logliap.ode
# the liapunov exponent of the logistic map
f(x)=a*x*(1-x)
fp(x)=a*(1-2*x)
```

```
init x=.1,a=2,z=0
x'=f(x)
a'=a
z'=z+heav(t-100)*ln(abs(fp(x))+1e-8)
aux liap=z/max(t-100,1)
@ xlo=2,xhi=4,ylo=-2,yhi=1
@ total=2000,trans=2000
@ meth=disc,bound=1000000
@ xp=a,yp=liap
done
```

Notes: The statement `z'=z+heav(t-100)*ln(abs(fp(x))+1e-8)` serves two purposes. First, if $f' = 0$, the addition of the small amount 1e-8 prevents a singularity in the log. Second, premultiplying by `heav(t-100)` adds only the numbers after 100 iterations. The quantity `liap` is just the running average of `z` and is the quantity of interest. The line `@ total=2000,trans=2000` tells *XPPAUT* to iterate for 2000 and keep only the last point. Run this and integrate using the range option. Use the same parameters in this as you did in computing the bifurcation diagram.

There is an easier way to compute Liapunov exponents over a range of parameters. *XPPAUT* has a built-in algorithm for the computation of Liapunov exponents for differential equations and maps. It works similarly to the idea described above. Load the following logistic equation:

```
# logistmap2.ode
x(t+1)=a*x*(1-x)
par a=3
init x=.54321
@ total=1000,trans=500,meth=disc
done
```

I have set the total iterates to 1000 and thrown away the first 500. Once you have *XPPAUT* up and running, click on `nUmerics stocHastic Liapunov` and choose `1` to look at a range. Choose `a` as the parameter to range over, choose `500` as the number of steps, and choose `3` and `4` as the start and end. Click on `Ok`, and after a short wait, the calculation will be done. Click on `Escape` to go to the main menu, and then click on `Xi vs t` and hit `Enter`. You will get a nice plot of the maximal exponent as a function of the parameter a. Note that when $a = 4$, $\lambda_{Max} = 0.6911265$ which is close to the actual value of $\ln(2) = 0.69314$. The difference is due to the finite number of points computed. See Figure 4.4.

4.2.4 The devil's staircase

If you periodically force an oscillator, then a number of interesting phenomena can occur. One of these is phase-locking, in which the resulting behavior is periodic. If the forced oscillator cycles N times while the forcing function cycles M times, this is called $N : M$ locking. One way to model a periodically forced oscillator is to look at a one-dimensional

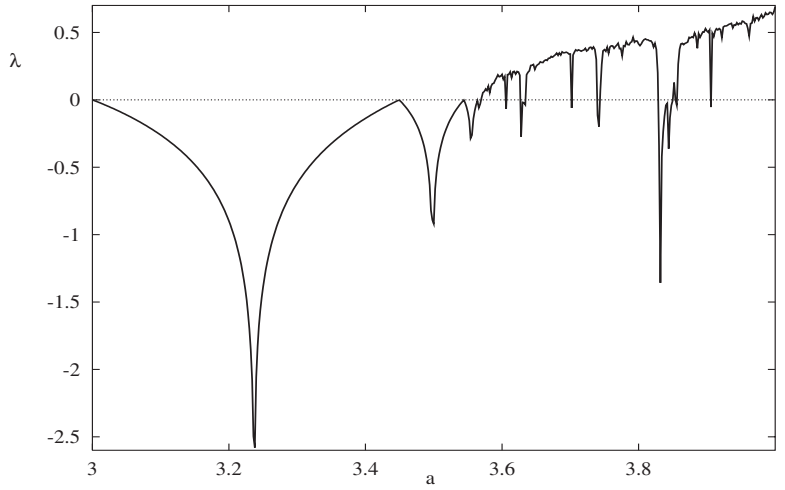

Figure 4.4. *Maximal Liapunov exponent as a function of the parameter a for the logistic map.*

map

$$x_{n+1} = f(x_n),$$

where the function f takes the unit circle back to itself. x_n is the phase of the oscillator after the nth stimulus. The standard map has the form

$$f(x) = x + b + a\sin(2\pi x) \bmod 1.$$

Consider $a = 0$ for a moment. Suppose that $b = N/M$. Then after M stimuli, $x_n = N + x_{n-M}$, that is, x has traversed N cycles while the stimulus has traversed M. This is $N : M$ locking. However, it is not very robust. Consider the ratio

$$R_n = \frac{x_n}{n}.$$

As $n \to \infty$, clearly $R_n \to N/M \equiv \rho$. This limiting value ρ is called the rotation number. Now, suppose that $a \neq 0$. Then we can no longer explicitly write the solution x_n. However, the rotation number still exists (if $f'(x) > 0$, e.g., if $|a| < 1/(2\pi)$) and in fact depends continuously on the parameter b. This is a consequence of Denjoy's theorem [18, p. 301]. Furthermore, it is constant almost everywhere. Thus, if we plot ρ as a function of b, we will get what is known as the *devil's staircase.* We can use *XPPAUT* to plot the rotation number as a function of the parameter b to see what this looks like. Here is the *XPPAUT* file:

```
# rotnum.ode
# the rotation number for the std map
f(x)=x+b+a*sin(2*pi*x)
x'=f(x)
b'=b
par a=.15
```

```
init b=0
init x=0
aux rho=x/max(t,1)
@ bound=100000
@ total=1000,trans=1000,meth=disc
@ xp=b,yp=rho,xlo=0,ylo=0,xhi=1.001,yhi=1.001
@ rangeover=b,rangestep=200,rangelow=0,rangehigh=1,
rangereset=no
done
```

Notes. As with the other bifurcation problems, we have treated the parameter as a variable so that it is available for plotting. We do not mod x by 1 here so that we can compute the total accumulation of phase. The rotation number is approximated by the finite ratio x/t, where t is the iteration number. As only the last value is of interest, the `trans=1000` tells *XPPAUT* to look only at the final point. The last line sets up the range integration so that you won't even have to fill in a dialog box.

Run this with *XPPAUT*. Click on `Initialconds Range` (**I R**). The dialog box should look like this:

`Range over: b`
`Steps: 200`
`Start: 0`
`End: 1`
`Reset storage: N`

Click on `Ok`. The devil's staircase will be drawn for you, as in Figure 4.5. (If you want a really detailed picture, make the number of steps 2000.) Then zoom into different regions and see that there are many flat regions corresponding to constant rotation numbers. If the amplitude parameter is larger than $1/(2\pi) \approx 0.16$, the map is no longer invertible and the assumptions for Denjoy's theorem no longer hold. Change a to 0.4 and redraw the staircase.

4.2.5 Complex maps in one dimension

The Mandelbrot set has often been called the most complex object in mathematics. While this is likely to be false, the computation of this set has led to a huge number of little applications which will draw it for you to any desired resolution. Many of these applications offer little or no explanation of the set nor of its mathematical significance. Here, I will offer a brief description of what the set means. First, let's consider the logistic equation that we looked at above:

$$x_{n+1} = a x_n (1 - x_n).$$

Set $a = 4$ and make the following transformation: $u = 2x - 1$. Then

$$u_{n+1} = 1 - 2u_n^2.$$

Let $z = u + iv$ and consider the complex map

$$z_{n+1} = z_n^2$$

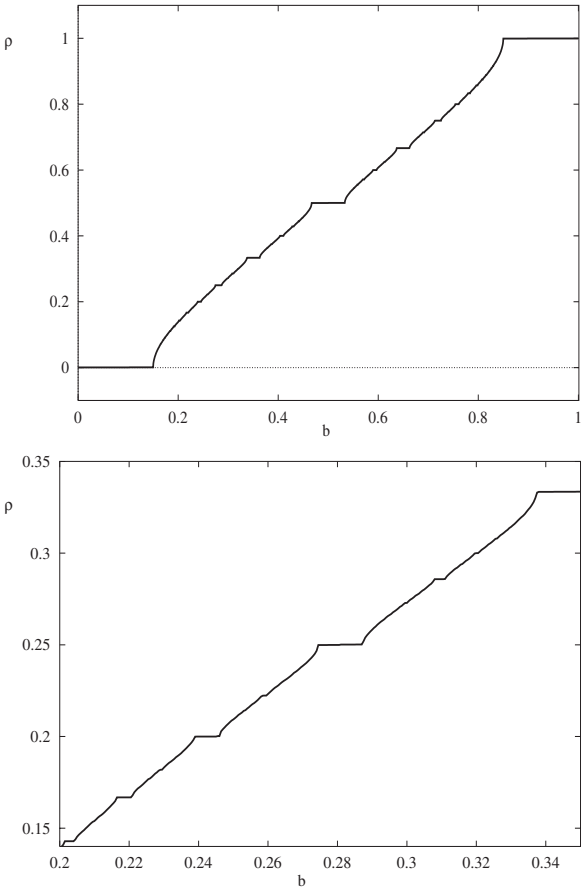

Figure 4.5. *Rotation number for the standard map and an expanded view over a small range of the parameter b.*

or, in real coordinates,

$$(u_{n+1}, v_{n+1}) = (u_n^2 - v_n^2, 2u_n v_n).$$

Suppose that we restrict z to have a magnitude of 1 so that $u^2 + v^2 = 1$. Then $u^2 - v^2 = 1 - 2u^2$, and we see that the logistic map with $a = 4$ is equivalent to the complex map $z \to z^2$ restricted to the unit circle. The dynamics on the unit circle can best be studied by writing $z = \exp(2\pi i\theta)$ so that the map is now

$$\theta_{n+1} = 2\theta_n \bmod 1.$$

This means that the dynamics is chaotic in almost every sense of the word. If we express a number between 0 and 1 in base 2, then the map just shifts to the left and drops the lowest bit. Every possible periodic solution is found, e.g., $0.100100100\ldots$ is a period 3 point. Furthermore, the Liapunov exponent of this is $\ln 2$ since the map expands all points by a factor of 2.

Looking again at the complex map $z \to z^2$, it is clear that any initial condition that starts inside the unit circle will tend to the fixed point 0, while any initial condition which starts outside of the unit circle will tend to infinity. Thus, the unit circle serves to separate points in the complex plane which are attracted to the fixed point 0 from those that are attracted to the fixed point at infinity. The set of points which are not attracted to these fixed points is called the Julia set. Thus, the Julia set of the map $z \to z^2$ is the unit circle. Now consider the general quadratic map with a complex parameter c:

$$z_{n+1} = z_n^2 + c.$$

If the parameter c is small, then the Julia set of this map should be close to the unit circle. It is a connected set with an inside and an outside. However, if c is large, then the Julia set will not look at all like the circle. In particular, there is no separation between initial conditions that stay bounded and those which go off to infinity. The famous Mandelbrot set is the set of all points c for which the Julia set is connected.

Now that we have connected the logistic map to the Mandelbrot set through the Julia set, how can we compute the Julia sets? Since the Julia set is a separatrix between fixed points, we can compute it by starting inside the unit circle and iterating backwards. The inverse of the map $z^2 + c$ is $\pm\sqrt{(z-c)}$. The way we will compute it is to *randomly* choose between the two square roots at each iterate. (Note that the idea of randomly choosing one of the square roots makes this computation related to so-called iterated function systems, which we will discuss below.) To compute the square roots, we first write the number $z - c$ in polar form $r\exp(i\theta)$ and then take the square root of r and divide θ by 2. Recall that if $u + iv$ is a complex number, then $r = \sqrt{(u^2 + v^2)}$ is its magnitude and $\theta = \tan^{-1}(v/u)$ is its argument. We then write the result of this as a two-dimensional real map. Here is the *XPPAUT* file `julia.ode`:

```
# julia.ode
# julia set for z -> z^2+c
par cx=0,cy=0
r=sqrt(sqrt((x-cx)^2+(y-cy)^2))
th=atan2(y-cy,x-cx)/2
s=sign(ran(1)-.5)
init x=.1
x'=s*r*cos(th)
y'=s*r*sin(th)
@ total=2000, meth=disc,trans=100
@ xp=x,yp=y,xlo=-2,xhi=2,ylo=-2,yhi=2
@ lt=0
done
```

Notes. The function `atan2(v,u)` takes into account the signs of u, v to return the correct argument θ for the number $u + iv$. I have thrown away the first 100 points and iterated it 2000 times. The last line `lt=0` tells *XPPAUT* that it should not connect the points with lines.

Start up *XPPAUT* with this file. Before iterating, turn off the axes so you don't confuse them with the Julia set: click `Graphics aXes opts` (**G X**) and change the

three entries `X-org(1=on)`, etc. from `1` to `0`, thus turning off the axes. Now integrate the equation and you will see a circular Julia set. (It looks elliptical since the graphics view is rectangular.) Change `cy` to -0.5 and integrate the equations again. Try `cy=0.5`. Can you say something about the symmetry of the Julia set? Try `cy=0,cx=-.5`. Try `cy=1.5,cx=0`. Do these look connected? Try `cy=1,cx=0` and iterate 4000 times. This may or may not be connected! It is hard to tell.

Animated Julia sets. Let's make many Julia sets and animate the result as the parameter varies. We set `cx=0` and vary `cy` from `-1` to `1`. First, change the total number of iterates back to 2000. Click on `Initialconds Range` (**I R**) and fill in the dialog box as follows:

`Range over: cy`
`Steps: 20`
`Start: −1`
`End: 1`
`Reset storage: Y`
`Use old ic's: Y`
`Cycle color: N`
`Movie: Y`

Notice that you should type in `Y` in the `Movie` entry since this tells *XPPAUT* you want to save the contents of the window. Also, make sure that no windows other than perhaps the dialog window obscure the main graphics window. Click on `Ok`, and 20 Julia sets will be computed. Each screen image is saved in memory so that you can play them back. Now click on `Kinescope Playback` (**K P**) and you can step through each of the plots one at a time by clicking on a mouse button. Click `Esc` to exit. To get a smooth animation of the Julia sets, click on `Kinescope Autoplay` (**K A**). Then tell *XPPAUT* how many times you want to repeat the animation (e.g., 2) and how many milliseconds between frames (e.g., 100). After this, the animation will be repeated as a smooth movie.

You can save the animations in several ways. One of these requires that you have a separate software program that can encode a series of still images into a movie. An easier way, but one which is not quite as efficient, is to create an animated GIF. Animated GIF's can be played on most web browsers such as Netscape or Explorer. Click on `Kinescope Make Anigif` and your movie will be replayed. At the same time, a file called `anim.gif` will be produced on your disk. You can view this with `xanim`, or other viewing software, or just open it with your web browser.

The Mandelbrot set

We have used *XPPAUT* to animate Julia sets. For open sets of parameters, the Julia set is connected. The set of parameters c for which the Julia set is connected is the Mandelbrot set. Another way to look at the Mandelbrot set is to define it as the set of parameters c such that iterates of $f(z) = z^2 + c$ starting at the origin remain bounded. (Since the Julia set is connected, there is no "escape" to infinity.) This set is typically computed by choosing a parameter c and iterating a number of times until z_n exceeds some bound. If, after this fixed number of iterates, z_n is still bounded, then c is considered to be in the Mandelbrot set. The number of iterates needed to determine whether a point is in the Mandelbrot set

becomes large near the boundary of the set. Often, points that are not in the set are color coded according to the number of iterates it takes to become unbounded.

To compute this set in *XPPAUT*, we will use a number of features which control the output and color. Here is the ODE file for computing the Mandelbrot set `mandel.ode`:

```
# mandel.ode
#  drawing the mandelbrot set
#
x'=x^2-y^2+cx
y'=2*x*y+cy
# so that the parameter axes are plottable
cx'=cx
cy'=cy
init x=0,y=0
#
# if this crosses 0 then we are out of the set
aux amp=x^2+y^2-4
#
#
@ xp=cx,yp=cy,xlo=-1.5,xhi=.5,ylo=-1,yhi=1,lt=-1
@ maxstor=40000,meth=disc,total=50
@ poimap=section,poivar=amp,poipln=0,poisgn=1,poistop=1
done
```

Notes. This is a rather simple ODE file with lots of controls added. We could easily set these controls in *XPPAUT*, but for ease of use set them here. The first two lines are the map written in the real variables, $z = x + iy$. The next lines are needed so that *XPPAUT* can use the parameters as coordinate axes. *XPPAUT* only allows auxiliary variables and dynamic variables as axes. The auxiliary variable `amp` changes from negative to positive when the magnitude of z exceeds 2. It can be shown that once this happens, z_n diverges to infinity. The controls in the first line set up the plot axes and bounds. The next line tells *XPPAUT* to allocate room for 40,000 points—the default is 4000; the method of integration; and the total iterations per run. We use only 50 since we are only crudely estimating the Mandelbrot set. The last option line starting with `poimap=section` tells *XPPAUT* that we are going to compute Poincaré maps. That is, we will plot only certain points when certain events happen. There are several different types of Poincaré maps that are possible in *XPPAUT*. If some quantity, say Q, crosses a value, say v, with a positive sign, then we plot all the variables at the time at which this happens. In the present case, we want the quantity `amp` to vanish, as that is when the magnitude of z_n is 2. The statements `poivar=amp,poipln=0,poisgn=1` tell *XPPAUT* to check the quantity `amp` hitting 0 with a positive derivative. The statement `poistop=1` means to stop the simulation when this happens.

Start *XPPAUT* with this input file. Turn off the coordinate axes by clicking on `Graphic stuff aXes opts` and setting `X-org`, etc. to 0. Next, we will do a two-parameter range integration like in the calculation of the bifurcation diagram for the logistic equation. Click on `Initialconds 2 par range` (**I 2**) and fill in the dialog box as follows:

Vary1: cx	Reset storage: N
Start1: -1.5	Use old ic's: Y
End1: .5	Cycle color: N
Vary2: cy	Movie: N
Start2: -1	Crv(1) Array(2): 2
End2: 1	Steps2: 200
Steps: 200	:

Click on Ok to run the simulation. This might take a bit of time; you are running 40,000 simulations. You should see the Mandelbrot set appear on your screen. The points in the set itself are empty. The reason for this is that in the Poincaré map option of *XPPAUT*, if the condition never occurs in the allotted time, no points are captured. Thus, a point is plotted at (cx,cy) if iterates with that parameter value leave a circle of radius 2 after fewer than 50 iterates.

Since *XPPAUT* knows how many iterates it takes to exit, we can color code the dots accordingly. Look at the first column in the **Data Viewer**. This is the time (interpolated) at which the iteration left the circle of radius 2. Thus, we can code these times by color. Click on nUmerics Colorcode Another quant (**U C A**). Accept the default of T. Click on Choose and choose 2 for Min and 12 for Max. Click on Esc to get to the main menu, and redraw the graph by clicking **R**. You should see the familiar, albeit somewhat crude, colorized Mandelbrot set. To get a nice picture of it, just create a new smaller window which is 200 x 200 dpi. Click on Makewindow Create and resize it. You should see something like Figure 4.6.

Exercise

Make a Julia set animation with the function $f(z) = z^3 + c$ and then make a Mandelbrot set for this function. (Note that, for cube roots, you want to randomly choose between three possible roots.) A good range for the cubic Mandelbrot set is c_x between -0.8 and 0.8 and c_y between -1.5 and 1.5. (Hint: $z^3 = x^3 - 3y^2x + i(yx^3 - y^3)$.) Create some art by changing some of the coefficients in the nonlinearities obtained above. Add some quadratic terms, change a few signs—whatever you want. To speed things up, change the 200 to 100 in the two-parameter range. When you find something good, then do a finer version. You don't have to create a new *XPPAUT* file for this exercise. Just click on FileEdit RHSs to edit the right-hand sides of the equations.

4.2.6 Iterated function systems

A variety of interesting fractals can be created by using randomly chosen affine maps in the plane. An affine map is just a transformation of the form

$$x \mapsto Ax + b.$$

If you take a series $\{A_n, b_n\}$ of such maps along with probabilities p_n for choosing each one, then this is called an iterated function system (IFS). Starting with some point x in the plane, choose a map, apply it to get a new point, and plot the point. Do this thousands of

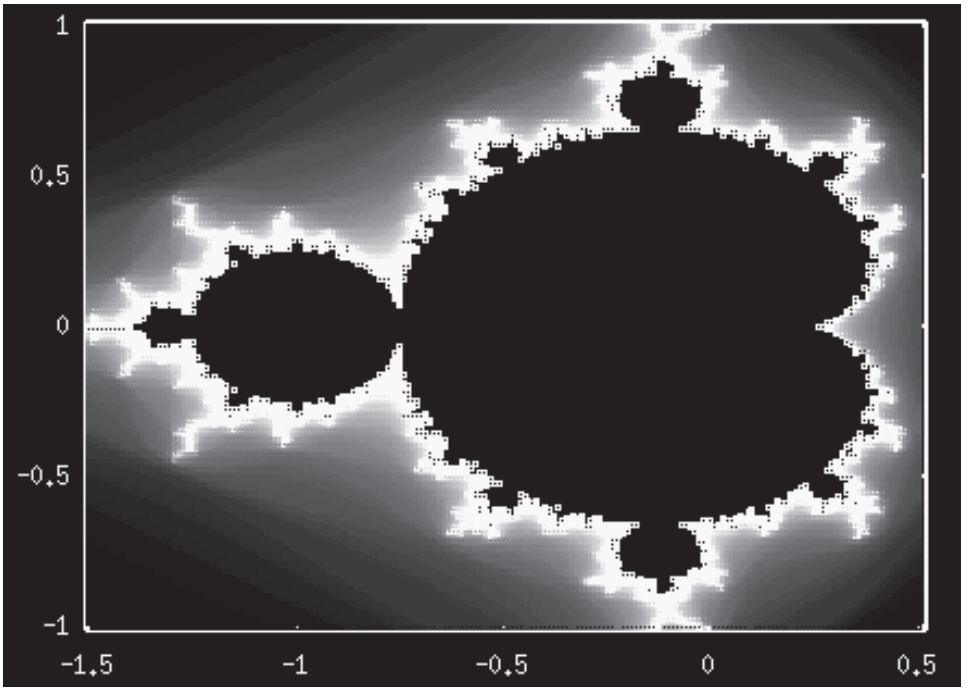

Figure 4.6. *Mandelbrot set.*

times and the result will often be an interesting image. Barnsley and others have developed image compression algorithms based on these ideas. However, we will use them to create fractal pictures. As an illustrative example that requires no computer, lets look at a one-dimensional example. At each step multiply x by 1/3. Then flip a coin. If it is heads add 2/3, otherwise add 0. From this, it is clear that the middle third of the interval can never be plotted. Similarly, the middle third of the left and right intervals can never be plotted. Thus, the points that are plotted consist of the interval with all the middle thirds removed—the Cantor set. This is a simple example and you can look at it via the following ODE file:

```
# cantor.ode
# plots the Cantor set
x'=x/3+2*heav(ran(1)-.5)/3
y'=.5
@ xlo=0,xhi=1,xp=x,yp=y,ylo=.4,yhi=.6
@ meth=discrete,total=40000,lt=0,maxstor=40001
done
```

Notes. The first equation is exactly the iteration. The term `heav(ran(1)-.5)` is 0 if the random number between 0 and 1 is less than .5; otherwise it is 1. The equation for `y` is a dummy; since we want to plot a one-dimensional set we keep the y-coordinate constant. The statement `lt=0` is to set the linetype to 0 which plots single dots.

Run this ODE file. Then use the `Window/zoom Zoom in` command to zoom into the left or right halves. Notice you get the same sort of set. Continue zooming in like this. At each step you will see the same set. The Cantor set is self-similar; that is, magnified regions of it look the same as the whole set.

Here is another cool trick: Make a histogram for the Cantor set with 100 bins. Click on `Numerics Stochastic Histogram` (**U H H**). Choose `100` for the number of bins, and accept the defaults. Exit the numerics menu by clicking on `Esc`. Next change the linestyle: click on `Graphic stuff Edit curve`. Choose `0` as the curve to edit, change the linetype to 1, and click on `Ok`. Finally, click on `X vs t` and choose `X` as the plot variable. You will see the histogram for the Cantor set. Go back into the `Numerics` menu and make another histogram, this time with 1000 bins, and replot it. The histogram has many more dips than before. If you zoom in, you will find the histogram also is fractal.

Now, let's make a two-dimensional iterated function system. We will start with a classic, the Sierpinsky triangle. This fractal is obtained by dividing a right triangle into four equal pieces and removing the middle piece. This is repeated in each of the three smaller triangles which remain, and so on. The result is like a two-dimensional version of the Cantor set. The iterated function system consist of three maps. Each map consists of first dividing the coordinate in half. Then with probability 1/3, 1/2 is added to the x coordinate, 1/2 is added to the y coordinate, or nothing is added. Here is an ODE file:

```
# simple iterated function system
# sierpinsky triangle
par c0=0,c1=.5,c2=0
par d0=0,d1=0,d2=.5
p=flr(ran(1)*3)
x'=.5*x+shift(c0,p)
y'=.5*y+shift(d0,p)
@ meth=discrete,total=20000,maxstor=100000
@ xp=x,yp=y,xlo=0,xhi=1,ylo=0,yhi=1,lt=0
aux pp=p
done
```

Notes. I define 3 values for constants that are added after multiplying by 1/2: `(c0, d0)`, `(c1, d1)`, and `(c2, d2)`. Then I flip a "three-sided" coin, `p`, which takes values of 0, 1, or 2. The function `shift(c0,p)` will be `c0,c1,c2` according to whether `p` is 0, 1, or 2, respectively.

Run the above ODE file and you will see the Sierpinsky triangle appear on the screen. Zoom in to see similar versions of it.

We close this section by writing the following simple IFS that has four general probabilities and four general affine maps. It should be clear from this how to generalize to more maps.

```
# ifs4.ode
# general code for ifs with 4 choices
par a1=0,a2=.85,a3=.2,a4=-.15
par b1=0,b2=.04,b3=-.26,b4=.28
par c1=0,c2=-.04,c3=.23,c4=.26
```

```
par d1=.16,d2=.85,d3=.22,d4=.24
par e1=0,e2=0,e3=0,e4=0
par f1=0,f2=1.6,f3=1.6,f4=.44
par p1=0.01,p2=0.85,p3=0.07,p4=0.07
s1=p1
s2=s1+p2
s3=s2+p3
z=ran(1)
i1=if(z>s1)then(1)else(0)
i2=if(z>s2)then(2)else(i1)
i=if(z>s3)then(3)else(i2)
x'=shift(a1,i)*x+shift(b1,i)*y+shift(e1,i)
y'=shift(c1,i)*x+shift(d1,i)*y+shift(f1,i)
@ meth=discrete,total=20000,maxstor=100000
@ xp=x,yp=y,xlo=-3,xhi=3,ylo=0,yhi=10,lt=0
done
```

Notes. The first six lines give values for the affine maps:

$$x' = ax + by + e, \quad y' = cx + dy + f.$$

The following lines give the four probabilities of each of the maps. Notice that these probabilities are not uniform. The numbers `s1,s2,s3` are the cumulative probabilities. Next, a random number `z` is selected. The next three lines of code determine which map to use based on `z` by testing whether or not `z` is larger than each of the cumulative probabilities. This sequence of code is similar to that used in our implementation of the Gillespie algorithm (see Chapter 5). Finally, the map is defined by using the shift operator on each of the parameters for the map. Thus, if `i=0` then the first map is used, and so on. For a much larger set of parameters, you would likely use lookup tables (see, for example, the cellular automata example in Chapter 6). If you run this ODE file, you will see the famous fractal fern.

4.3 One-dimensional ODEs

Many courses in differential equations start off by describing a variety of exactly solvable first order differential equations. However, recent texts, such as those by Blanchard, Devaney, and Hall [3], Strogatz [37], and Borelli and Coleman [4] take a more qualitative approach and emphasize graphical and geometric methods for solving differential equations. There are several ways to gain a qualitative view of the behavior of a differential equation. One way is to draw the *phase-line*. Consider the differential equation

$$\frac{dx}{dt} = f(x).$$

To draw the phase-line for this, we plot $f(x)$ versus x. Then we use the sign of $f(x)$ at any point x to determine the direction along the x axis. Fixed points are precisely zeros of $f(x)$. Thus, to draw a phase-line plot, we draw $f(x)$ and then draw representative trajectories on the x-axis. *XPPAUT* can animate this entire process. We will construct an animation file

which plots the function $f(x)$ and a moving ball showing the behavior of $x(t)$. The point of this is to see how the value of $f(x)$ determines the speed of the particle. First, we make an ODE file for the scalar system. Then we add the animation file. Here is the ODE file:

```
# phase-line.ode
# one-dimensional autonomous system
f(x)=x*(x-.25)*(1-x)
x'=f(x)
init x=.26
# transformation information for the animator
par x0=-.5,x1=1.5,y0=-1,y1=1
tr(x)=x0+(x/100)*(x1-x0)
ff(x)=(f(tr(x))-y0)/(y1-y0)
@ ylo=-.25,yhi=1.25,total=40,xhi=40
done
```

The file is pretty straightforward. I have defined four parameters `x0,x1,y0,y1` to scale the x-axes and the y-axes for the animator. The depicted phase-space (in the animation) will be the interval $[x_0, x_1]$. The parameters `y0,y1` give the maximum and minimum vertical values in which $f(x)$ is plotted. Since the x-axis is the phase-space, the parameter `y0` should be less than 0 and `y1` should be greater than zero so that $y = 0$ (the x-axis) will be visible. The function `tr(x)` is used by the animator. If you run this file, the usual $x(t)$ versus t will be plotted. The fun begins when we look at the animation. In the animation file, I plot the function $f(x)$ and then animate a little ball along the x-axis. Here is the animation file:

```
# animation file for a phase-line
# phase-line.ani
permanent
# this code draws a function ff(x)
line [0..99]*.01;ff([j]);[j+1]*.01;ff([j+1]);$RED;2
# a straight line along y=0
line 0;-y0/(y1-y0);1;-y0/(y1-y0);$BLACK
transient
# following the dancing ball
fcircle (x-x0)/(x1-x0);-y0/(y1-y0);.02;$BLUE
end
```

Since we have not looked at animation files in depth, I will explain this one in detail. (However, you may want to first look at the chapter on animation, Chapter 8.) Lines that begin with # are comments. The line `permanent` tells the animator that whatever follows is static and is not animated. For example, titles and other things in the animation are usually not dependent on time. The code starting with `line [0..99]` is somewhat mysterious, so we will pick it apart. The general form for the `line` command is

```
line x1;y1;x2;y2;color;thick
```

This will draw a line from `(x1,y1)` to `(x2,y2)` in color `color` with thickness (in pixels) `thick`. The lower left and upper right coordinates of the animation window are (0,0) and

(1,1), respectively. The `[0..99]` notation is new; this just tells *XPPAUT* to repeat this line 100 times with a dummy index going from 0 to 99. (See Chapter 6 for more details on this notation.) Thus, 100 lines will be drawn. The first coordinate of the first line is 0, the first coordinate of the second line is 1*.01, and so on. The vertical coordinate of the first line is `ff(0)` since `[j]` refers to the index of the current line, which is 0. The last line in this 100-line sequence is expanded by *XPPAUT* to be

```
line 99*.01;ff(99);100*.01;ff(100);$RED;2
```

The function `ff(u)` is defined in the ODE file. Thus, this code simply draws the function $f(x)$. The next line of code tells the animator to draw the horizontal axis scaled to be at $y = 0$. The code `transient` tells *XPPAUT* that all that follows should be animated. A filled circle is drawn at the scaled x coordinate of the dynamical system with radius .02 units (that is, 2% of the entire animation window) and it is colored blue.

Click on `View axes Toon` (**V T**) to open the animation window. Click on `File` in the animation window and load the animation file `phase-line.ani`. Unless there is an error, you should be ready to run the animation. (If there is an error, then you did not type in the animation filename correctly.) Click on the `Go` button in the **Animation Window** and the graph of f will show up along with a little ball showing the progress of the dynamical system.

Notes.

1. You can edit animation files and load them into *XPPAUT* without having to leave *XPPAUT*.

2. If you click on the little box in the upper right-hand side of the animation window, the animation will run simultaneously with the simulation. However, this slows down the simulation.

Exercises

1. Try animating the following equation:

$$x' = \sin \pi x + 1.05.$$

   To do this, in the **Main Window** click on `File Edit Functions` (**F E F**), change the first function to `sin(pi*x)+1.05`, and then click on `Ok`. Then run the ODE with $x(0) = -0.5$ as an initial condition. Rescale the phase-variable axes for the animation by setting the parameters `x0=-.5` and `x1=10.` Then run the animation.

2. Try to create an ODE file and an animation file for a one-dimensional nonautonomous equation. Note that since the function is nonautonomous, we must also redraw the function curve at every frame. This is not the best way to look at nonautonomous one-dimensional systems. The next section suggests a better way. Here are the two files:

```
# phase-line2.ode
# one-dimensional nonautonomous system
```

```
f(x,t)=x*(x-.25)*(1-x)+a0+a1*sin(w*t)
par a0=0,a1=0,w=0
x'=f(x,t)
init x=.26
# transformation information for the animator
par x0=-.5,x1=1.5,y0=-1,y1=1
tr(x)=x0+(x/100)*(x1-x0)
ff(x,t)=(f(tr(x),t)-y0)/(y1-y0)
@ ylo=-.25,yhi=1.25,total=40,xhi=40
done

# animation file for a phase-line2.ode
# has time-dependence
permanent
# a straight line along y=0
line 0;-y0/(y1-y0);1;-y0/(y1-y0);$BLACK
transient
# this code draws a function ff(x,t)
line [0..99]*.01;ff([j],t);[j+1]*.01;ff([j+1],t);$RED;2
# following the dancing ball
fcircle (x-x0)/(x1-x0);-y0/(y1-y0);.02;$BLUE
end
```

4.3.1 Nonautonomous one-dimensional systems

Nonautonomous systems are somewhat more difficult to study. The exercise above gives you one way to look at them but this is strictly numerical. Consider

$$\frac{dx}{dt} = f(x,t). \tag{4.1}$$

One way to look at such a system is in the (t,x)-plane. *Direction fields* are very helpful in understanding the dynamics. At a regular array of (t,x) values in the plane, we draw a little arrow whose direction is that of the vector $(1, f(x,t))$. This tells us the tangent point to the solution to (4.1) through the point (t,x), and thus we can sketch an approximate solution without ever having to actually solve the differential equation. *XPPAUT* has tools for drawing direction fields. However, these tools work only for two-dimensional dynamical systems. This is no problem. Consider the differential equation

$$\frac{ds}{dt} = 1, \quad \frac{dx}{dt} = f(x,s).$$

This is identical to (4.1) but is now sitting in two dimensions. With these considerations, we introduce the general one-dimensional nonautonomous differential equation file

```
# oned.ode
# generic one-dimensional ODE file
f(x,t)=x*(1-x)
```

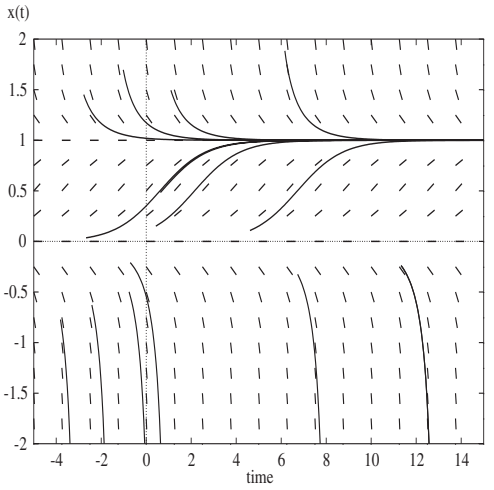

Figure 4.7. *Direction fields and sample trajectories for the equation* $x' = x(1 - x)$.

```
s'=1
x'=f(x,s)
@ xp=s,yp=x,xlo=-5,xhi=15,ylo=-2,yhi=2
done
```

Run *XPPAUT* with this file. The axes are set up so that the time-like variable s is on the horizontal axis and x is on the vertical axis. Click on `Dir.field/flow` `Scaled dir. field` (**D S**) and change `10` to `15` at the prompt. You will see a direction field appear. (For fun try the nonscaled direction fields, **D D**, and see why I used the scaled ones. In the unscaled fields, the length is proportional to the magnitude of the vector $(1, f)$.) Based on the direction fields, you should have no trouble figuring out the trajectories starting at any point. To check if you are right, click on `Initialconds` `Mice` (**I I**) and use the mouse to click on a bunch of different points in the window. Click outside the view window to quit inputting initial data.

Hardcopy hint

XPPAUT keeps only the latest integration in memory so that if you want hardcopy of all the trajectories you computed along with the direction fields, you won't be able to get it. However, there is a way to keep up to 26 trajectories. Click on `Graphic` `stuff` `Freeze` `On` `freeze` (**G F O**) to automatically "freeze" the trajectories onto the screen. Now redraw the direction fields and compute a bunch of trajectories. Finally, save your work: Click on `Graphic` `stuff` `Postscript` and accept the defaults; choose the default filename `oned.ode.eps` and *XPPAUT* will create a PostScript file in your directory which you can plot or view later. Delete the saved trajectories (if you want) by clicking on `Graphic` `stuff` `Freeze` `Remove all`. Figure 4.7 shows an example.

Exercises

Draw direction fields for the following functions $f(x,t)$: (i) $x - t/10$; (ii) $\sin(4x)\cos(t/2)$; (iii) $t - x^2$; (iv) $t/10 - x$; (v) $\sin(xt)$. Try to guess what the solutions will look like based on the direction fields, and verify your guess by computing a number of trajectories. (Hint: To change the function $f(x,t)$ just click on `File Edit Function` and change $f(x,t)$.)

4.4 Planar dynamical systems

Two-dimensional differential equations play an important role in many undergraduate differential equations courses. This is because of the special topological properties of the plane, in particular, the Jordan Curve Theorem. This constraint implies that the only attractors in the plane are fixed points and limit cycles. Most textbooks that emphasize the qualitative aspects of differential equations include a chapter or two on planar ODEs. *XPPAUT* has many features that are useful in the analysis of these systems such as the ability to draw direction fields, nullclines, and separatrices for saddle-points. I have used *XPPAUT* with several undergraduate texts on ODEs. Used intelligently, the analysis of planar systems can be automated for teaching some complicated dynamical systems concepts. I will start this section with the analysis of two-dimensional linear systems. The general linear system has the form

$$x' = ax + by,$$
$$y' = cx + dy,$$

where a, b, c, d are parameters. The behavior of this system is completely determined by the eigenvalues of the matrix

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

If the eigenvalues are both real and have the same sign, the origin is a *node*; if they are real and have opposite signs, it is a *saddle*; and if the eigenvalues are complex with nonzero real parts, the origin is a *vortex* or *spiral*. In the cases of nodes and spirals, if the real part of the eigenvalues is negative (positive) the node or spiral is attracting (repelling). There are several degenerate cases. If there is a zero eigenvalue, then the eigenvector corresponding to the zero eigenvalue is a line of fixed points for the system. Finally, if there are pure imaginary eigenvalues, then the origin is called a *center*.

Before introducing the computer code, we recall a few facts about linear two-dimensional systems. Define the *trace* to be $T = a + d$, the *determinant* to be $D = ad - bc$, and the discriminant to be $Q = T^2 - 4D$. Then the following hold:

- If $D < 0$, then the origin is a saddle-point;
- If $T < 0$, $D > 0$, and $Q > 0$, then the origin is a stable node;
- If $T > 0$, $D > 0$, and $Q > 0$, then the origin is an unstable node;
- If $T < 0$, $D > 0$, and $Q < 0$, then the origin is a stable spiral;
- If $T > 0$, $D > 0$, and $Q < 0$, then the origin is an unstable spiral;
- If $T = 0$ and $D > 0$, then the origin is a center;
- If $D = 0$, then there is a line of fixed points.

We will illustrate these with the following generic two-dimensional linear differential equation with parameters:

```
# twodl.ode
# simple linear planar dynamical system
x'=a*x+b*y
y'=c*x+d*y
par a=1,b=2,c=-3,d=-1
@ xlo=-2,ylo=-2,xhi=2,yhi=2,xp=x,yp=y
@ total=30,nmesh=100
done
```

Note. The file is straightforward; the statement `nmesh=100` tells *XPPAUT* to make a finer mesh for drawing nullclines (which are explained below).

Run this file. Click on `Dir.field/flow Direct field` (**D D**) to draw unscaled direction fields. Since $T = 0$ and $D > 0$, it follows that the origin is a center. Try $d = -2$. This changes the center into a stable spiral. Try $d = -3$. What is the origin? At what value of d does it become a node? At what value of d does it become a line of equilibria? Draw the direction fields and some trajectories when $d = -6$. You should be able to make out the line of fixed points, $y = -x/2$. Change d back to -2. Click on `Dir.field/flow Flow` (**D F**) and change the grid to 5 to draw a flow diagram of the system. You will see a nice vortex. Unfortunately, *XPPAUT* does not save these trajectories, but they do give a nice picture of what is going on.

Hardcopy hint

Click on `Kinescope Capture` to grab the screen image. Now click on `Kinescope Save` and choose any name you want for the `base` name, e.g., `twodl`, and then choose `2:GIF` as the format and a standard GIF file called `twodl_0.gif` will be created. `Kinescope Capture` can be used to capture a sequence of frames and these can be saved in sequence or even animated. Single frames can be viewed with a standard browser or most any graphics software (e.g., `xv` on Linux systems). On Windows systems, you can capture the window to the clipboard and dump it into a Paint or Word file. (This is done by clicking on the window and pressing the `Alt` and `PrtScrn` keys.)

Erase the screen and use the mouse to start some initial data at roughly $(2, -2)$. You will get a nice spiral. Let's color code this trajectory according to the absolute velocity, $\sqrt{\dot{x}^2 + \dot{y}^2}$. Click on `nUmerics Colorcode Velocity` (**U C V**) and choose `Optimize`. Escape back to the main menu. Your trajectory is nicely colored with the minimum velocity blue and the maximum red. Let's color the whole phase-plane according to this color scheme. Click on `Dir.field/flow Colorize` and change the grid to 100. You can colorize according to many different quantities (such as energy for a conservative system, as we shall see later). While *XPPAUT* does not create a direct PostScript version of your picture, you can save the screen image as a GIF file as described above. Individual color-coded trajectories are rendered in the PostScript file. Erase the screen (**E**) and go back into the numerics menu and turn off the colorizing (`nUmerics Colorize No color`) and return to the main menu (`Esc`).

XPPAUT can tell you the nature of the origin. Click on `Sing pts Go` (**S G**) and answer `Yes` to the dialog box. The *singular point* or *fixed point* or *equilibrium* is computed. If you look in the terminal window from which you started *XPPAUT*, you will see that the eigenvalues are printed and have values of $-1/2 \pm i1.93$. A new window appears which gives a summary description of the stability of the fixed points, the nature of the eigenvalues, and value of the fixed point. From this window, you see that the origin is stable and that it has two complex eigenvalues with negative real parts (as shown by the text `c-=2`). Change the parameter `c` from -3 to 3. Draw the direction fields and a flow diagram. This is a saddle-point since $D = -8$ is the determinant. Determine the stability of the origin. Click on `Sing pts Go` and this time answer `No` to the `Print eigenvalues` dialog box. Another message box will appear. This asks if you want to draw invariant sets. Answer yes to this. Then press `Enter` four times. You will see four trajectories coming out of the origin. They go northeast, northwest, southeast, southwest. Two are yellow—these are the unstable manifolds of the saddle-point and (since this is a linear system) are identical to the eigenvector associated with the positive eigenvalue. The other pair of trajectories is blue; they are the branches of the stable manifold and are identical to the eigenvectors associated with the negative eigenvalue. When *XPPAUT* drew these, the first two were the unstable manifolds and the second were the stable ones. Whenever there is a one-dimensional unstable manifold, *XPPAUT* will draw this first. If you erase the screen, you will find that you have lost these manifolds. However, *XPPAUT* keeps the initial data required to recompute them. To draw the *unstable manifolds*, click on `Initialconds sHoot` (**I H**). You will be prompted to choose a number from 1–4. Since the first two trajectories made up the unstable manifold, pick either 1 or 2 and one or more branches will be drawn. To get both of the branches, pick the second of the two after freezing the first trajectory. To get the stable manifolds, you must change the direction of integration; that is, you must solve the equations backward in time. To solve ODEs backwards in time in *XPPAUT*, click on `nUmerics Dt` and change it to a negative value (e.g., `-0.05`), then exit the numerics menu. Click on `Initial conds sHoot` and choose either 3 or 4, as the stable manifolds are always the last two. Make sure that you change the time step `Dt` back to a positive value if you want to explore the system more. (**Note.** As with the flow-fields, you can use the Kinescope to capture the screen with the manifolds as a quick and dirty way of getting a permanent record.)

4.5 Nonlinear systems

Since *XPPAUT* uses numerical methods, it doesn't matter whether or not the system is linear. We now turn to the analysis of the following two-dimensional nonlinear systems:

$$\frac{dx}{dt} = f(x, y), \tag{4.2}$$

$$\frac{dy}{dt} = g(x, y). \tag{4.3}$$

As with linear systems, direction fields can lead to a good qualitative picture of the behavior. Another qualitative method related to the computation of direction field is the use of **nullclines**. Nullclines break the phase-plane into regions, where $(dx/dt, dy/dt)$ are of

constant sign. Thus, they serve to give a sense of the direction of the flow. The x-nullcline (respectively, y-nullcline) is the curve defined by $f(x, y) = 0$ (respectively, $g(x, y) = 0$). Furthermore, the points where the x- and y-nullclines intersect are fixed points. *XPPAUT* easily computes the nullclines; it can even animate them as a function of a parameter.

Suppose that a nonlinear system has a fixed point and that the linearization has no eigenvalues with zero real part. Then, locally, the behavior of the nonlinear system is identical to that of the associated linear system. In particular, (i) the stability is unchanged, and (ii) the stable and unstable manifolds of saddle-points are tangent to those of the corresponding linearized system. This means that *XPPAUT* can compute both the stability and the stable/unstable manifolds for nonlinear systems.

With these tools, it is easy to put together a complete picture of the dynamics of any two-dimensional system. Let's consider the following example from a textbook [13]:

$$\frac{dx}{dt} = x + y + x^2 - y^2 + 0.1,$$

$$\frac{dy}{dy} = y - 2xy + x^2/2 + y^2.$$

This is not simple to analyze by hand. That's why we use the computer to complete the picture. The first thing to do is to put it into *XPPAUT*. Here is the ODE file `ex2d.ode`:

```
# example from Golubitsky
#
x'=x+y+x^2-y^2+0.1
y'=y-2*x*y+x^2/2+y^2
@ xp=x,yp=y
@ xlo=-6,xhi=6,ylo=-6,yhi=6,dt=.02
done
```

I have taken the liberty of setting it up as a phase-plane in a 12×12 window centered at the origin. To get a general picture of what it looks like, plot the direction field by clicking `Dir.field/flow` `Direct Field` (**D D**) and accept the defaults. Another quick way to get a picture of the behavior is to make a flow picture which draws a bunch of trajectories both forward and backward in time: click on `Dir.field/flow` `Flow` (**D F**) and accept the defaults. A nice picture will emerge.

It is not clear how many fixed points there are, but there are certainly more than one. To get a feel for the number of fixed points and where they are, click on `Nullcline` `New` (**N N**). The x-axis nullcline is red and the y-axis nullcline is green. (**Note.** Sometimes the nullclines look quite choppy. You can make them much smoother by increasing the numerical mesh size—click on `nUmerics` `Ncline ctl` and change the mesh from 40 to 100. `Escape` back to the main menu, erase the screen, and recompute the nullclines.) It looks like there are either 2 or 4 fixed points. We can blow up the region a bit where the fixed points are to better resolve this. Click on the `Window/zoom` `Zoom` (**W Z**) command and zoom (by clicking the mouse) into an area which contains all the equilibria. (A good window is [−5,4] X [−2,5]). Click on `Erase` to erase the screen. Redraw the nullclines and the direction fields (**N N D Enter Enter**). You can now see clearly that there are 4 fixed points. Now let's assess the nature of the fixed points indicated by the intersections of the

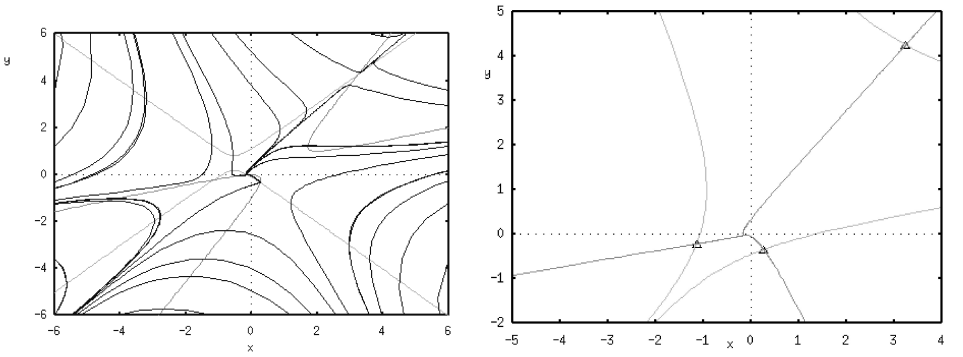

Figure 4.8. *Phase-plane showing trajectories and nullclines for the two-dimensional nonlinear example. Right: The stable and unstable manifolds of the fixed points.*

nullclines. Click on `Sing. Pts Mouse` and click with your mouse on the upper-right fixed point. Answer **No** to the question about eigenvalues and **Yes** to the question about invariant sets. A yellow trajectory (one branch of the unstable manifold) is drawn and a window appears warning you that the trajectory is out of bounds. Press any key and the other branch of the unstable manifold is drawn. Press a key and a branch of the stable manifold (blue) is drawn. Press the `Esc` key to get the other branch. You should see four trajectories for this saddle-point: two in yellow (the unstable manifolds) and two in blue (the stable manifolds). Repeat this at the other three fixed points—the second and third are saddles and the fourth is an unstable vortex. You should see something like Figure 4.8. (In the figure, for clarity, I have split the picture into two parts, showing the flow and nullclines to the left and the stable and unstable manifolds to the right.) Arrows go out of all the yellow/orange trajectories emerging from the saddle-points (triangles) and come into the the blue trajectories toward the saddle-points. Thus a point starting near one of the blue trajectories will go toward the saddle to which it is connected and then out along a yellow/orange trajectory. Since there are no stable fixed points and no limit-cycles in this example, integrating the equations forward in time will almost always result in solutions that go to infinity. Try a few initial conditions to convince yourself of this fact.

We can study a different example either by writing a new ODE file or editing the right-hand sides of the equation. Let's examine the following equation:

$$\frac{dx}{dt} = x(x(1-x) - y), \quad \frac{dy}{dt} = y(x - .4).$$

Click on `File Edit RHS` to edit the right-hand sides. Enter these two equations (carefully) and then click on `Ok`. Change the window to [−0.25,1.25] X [−0.25,1.25] by using the `Window/Zoom Window` dialog box. Draw some direction fields using `Scaled` or `Unscaled`. Recall that scaled direction fields represent only the direction, whereas unscaled fields also change the length as a function of the size of the vector field. I prefer the former, but this is a matter of taste. Try either one. Click on **D S** and use a grid size of 16 (instead of 10) and you will get pure direction fields. Click on **N N** to get the nullclines.

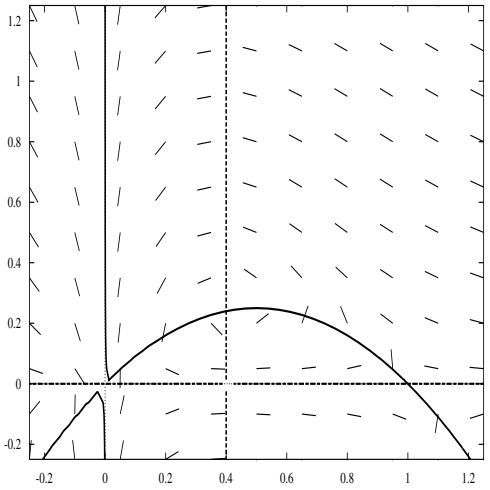

Figure 4.9. *Direction fields and nullclines of a predator–prey example.*

You will see a picture like Figure 4.9. There are clearly two fixed points and the third (0,0) is ambiguous—the nullcline algorithm is not perfect. Verify that (0,0) is a saddle-point and that the stable and unstable manifolds are the y- and x-axes, respectively. Examine the fixed point $x = 1$, $y = 0$. Show it is a saddle-point as well. In this case, note that one branch of the unstable manifold (in yellow) appears to have a limit cycle as its limit. You will have to click on `Esc` to stop the integration of this trajectory. Finally, note that the fixed point $x = 0.4$, $y = 0.24$ is an unstable vortex. Choose some initial data in the positive quadrant to verify that solutions all tend to a stable limit cycle. Recall that a **limit cycle** is an isolated periodic orbit for a differential equation.

4.5.1 Conservative dynamical systems in the plane

The equations for a frictionless particle with mass m operating under the influence of a potential function $P(x)$ have the following form:

$$\frac{d}{dt}\left(m\frac{dx}{dt}\right) = -\frac{\partial P}{\partial x} \equiv -f(x), \tag{4.4}$$

where $f(x)$ is the force and mdx/dt is the momentum. If we write this as a system of equations for the position x and the velocity v, we obtain

$$\frac{dx}{dt} = v, \quad m\frac{dv}{dt} = -f(x),$$

where we have assumed that the mass is constant. If we multiply (4.4) by dx/dt we can integrate the equation, revealing that the total energy (the kinetic plus the potential) is conserved:

$$E = \frac{1}{2}mv^2 + F(x),$$

that is, $dE/dt = 0$. In general, we say the system (4.2) has an *integral* if there is a function, $I(x, y)$ such that $dI/dt = 0$ along trajectories. An easy way to test if a system is integrable in two dimensions is to show that

$$\frac{\partial f}{\partial x} + \frac{\partial g}{\partial y} = 0.$$

In this case an integral is easy to find and I leave it as an exercise. For example, the system

$$x' = xy + y^3, \quad y' = -y^2/2 + x^2(1 - x)$$

is integrable, and the function

$$I(x, y) = xy^2/2 + y^4/4 + x^4/4 - x^3/3$$

is constant along solutions. Let's first explore this system, and then we will turn to some more standard mechanical systems. Here is the *XPPAUT* file for this system:

```
# integrable.ode
# an integrable system with its integral
x'=x*y+y^3
y'=-y^2/2+x^2*(1-x)
# here is the integral
aux i=x*y^2/2+y^4/4+x^4/4-x^3/3
# here is the log of the integral
aux li=log(x*y^2/2+y^4/4+x^4/4-x^3/3)
@ xp=x,yp=y,xlo=-2,ylo=-2,xhi=2,yhi=2
@ total=100
done
```

Note. To improve scaling, I have also computed the log of the integral—taking the log of a quantity can be used to better delineate large orders of magnitude.

I have set up the plot as a phase-plane. Explore this system by setting some initial conditions in the plane. Note that almost every solution that you compute is periodic. This is a common feature of conservative systems. Check in the **Data Viewer** that the quantity I remains constant along every trajectory. Turn on the freeze option (`Graphic stuff Freeze On freeze`) and then use the mouse to choose a bunch of nice trajectories (`Initial conds mIce`). Let's colorize the plane according to the log of the integral. *XPPAUT* allows you to colorize trajectories and the phase-plane according to the values of the vector field at each point along the trajectory or in the phase-plane. You choose to color according to the current time along the trajectory, the speed, or some other quantity, such as the integral. This other quantity must be a single variable or plottable quantity. Click on `nUmerics` to go to the numerics menu. Then click on `Colorize Another quantity` and select `li`, the log of the integral as the quantity. Then for the color scaling, select `Choose`. Choose -7.5 as the minimum and 1 as the maximum. `Escape` to the main menu. Click on `Dir.field/flow` and `Colorize` and choose a grid of 100. You should see the plane fill with color. Red dominates, as away from the origin, I is large. Notice that near the origin, all the colors change rapidly. You should see something like Figure 4.10.

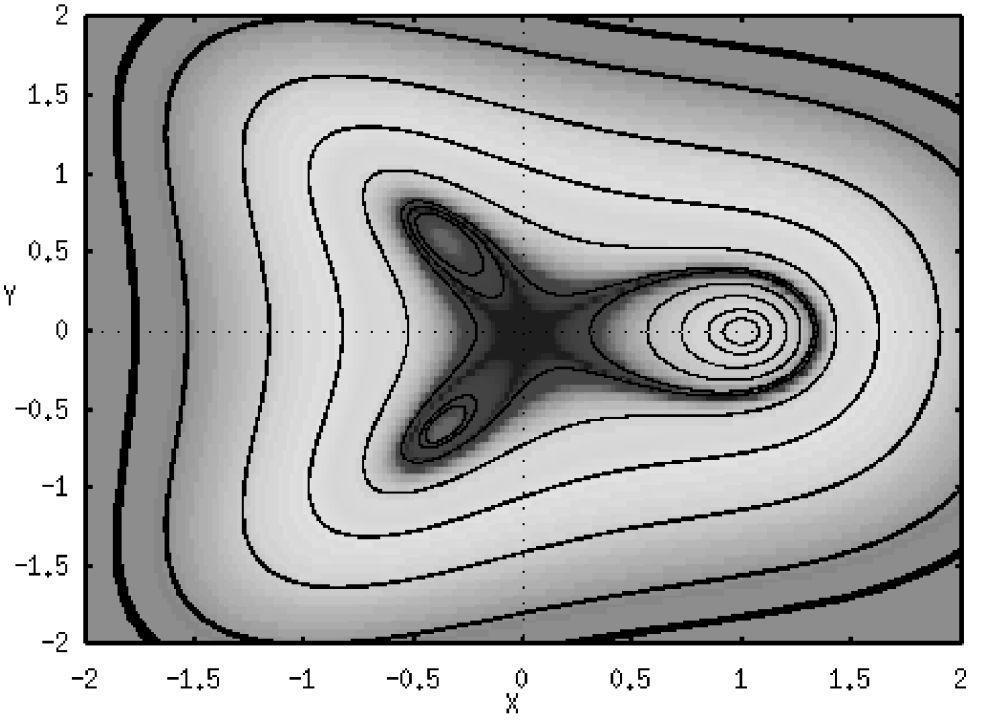

Figure 4.10. *Colorized conservative system: Greyscale indicates log of the integral.*

Now let's turn to the double well potential, $P(x) = x^4/4 - x^2/2$. The differential equation is

$$m\ddot{x} = x - x^3$$

which we write as a system

$$\dot{x} = y, \quad \dot{y} = (x - x^3)/m.$$

The total energy is

$$E = my^2/2 + x^4/4 - x^2/2$$

so that you can either write a new ODE file or edit the right-hand sides of the previous one. Here is my ODE file, `double_well.ode`:

```
# double well potential
x'=y
y'=(x-x^3)/m
par m=1
aux e=m*y^2/2+x^4/4-x^2/2
@ xp=x,yp=y,xlo=-1.5,ylo=-1.5,xhi=1.5,yhi=1.5
done
```

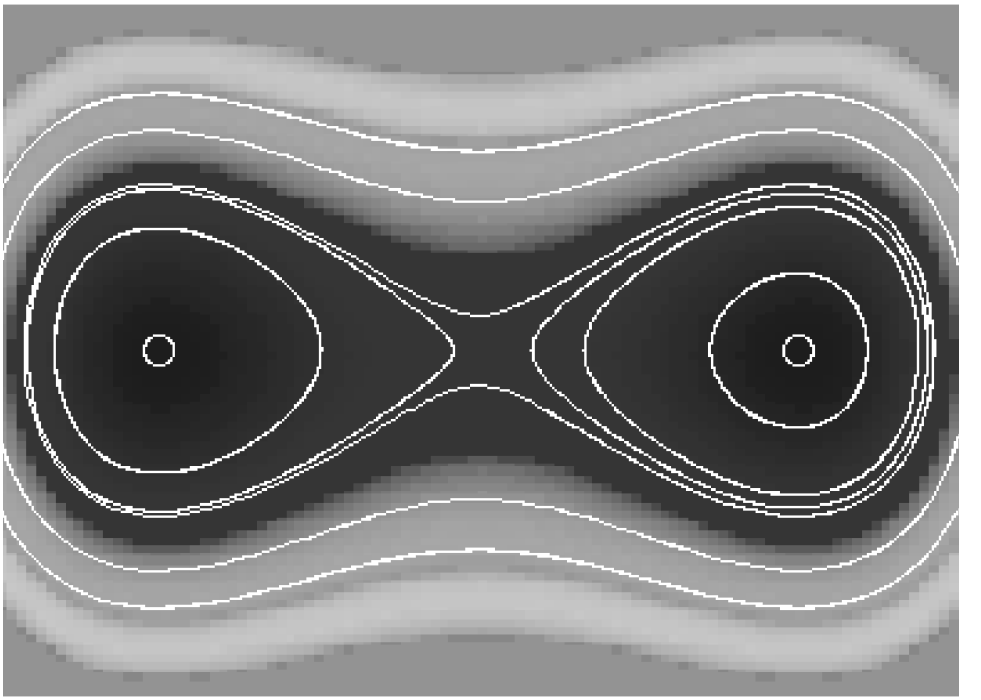

Figure 4.11. *Double well potential.*

Do the following steps:

1. Click on `Graphic stuff` `Freeze On freeze`;
2. Click on `Initial conds` `mIce` and choose 10 or so initial conditions to get some nice trajectories;
3. Click on `nUmerics` `Colorize Another quantity` and choose `E` as the quantity and select `Choose`; set the minimum to $-.25$ and the maximum to .75; `Escape` to exit to the main menu;
4. Click on `Dir.field/flow` `Colorize` and choose a grid of 100;
5. Redraw by clicking `Restore`.

You should see a nice figure-eight pattern such as in Figure 4.11.

4.5.2 Exercises

Here are some equations to study. Some of these are integrable so you should find the integral. For each of these, draw the nullclines, the direction fields, and the stable and unstable manifolds of the saddle-points.

1. $x' = y(1-x^2)$, $y' = 3-x^2-y^2$ in the window $[-3,3]$ X $[-3,3]$. This is an interesting vector field since on one side it looks like it is integrable while on the other side the two fixed points are nodes.

2. $x' = y(1-x^2)$, $y' = x(3-x^2-y^2)$ in the same window.

3. $x' = x + 4x^3 - x^5 - y$, $y' = x$ in the window $-3 < x < 3$ and $-6 < y < 6$. How many limit cycles are there and how many fixed points? (**Hint:** To find the unstable limit cycle(s), you should integrate backwards in time; click on `nUmerics Dt` and change it to -0.02.)

4. Draw the phase-portrait for this system, $x' = y - y^3 - x^3$, $y' = x^3 - x + 3x^2y$. Show that it is integrable. How many fixed points are there and what is their nature? Draw all the connecting trajectories from saddle to saddle.

5. $x' = y(1-y^2)$, $y' = -x(1-2x^2)$. This system can be perturbed to produce a system with 11 limit cycles!

6. Here is a classic two-neuron neural network:

$$\tau_1 u_1' = -u_1 + f(w_{11}u_1 + w_{12}u_2 - \theta_1), \quad \tau_2 u_2' = -u_2 + f(w_{21}u_1 + w_{22}u_2 - \theta_2),$$

where $f(u) = 1/(1+\exp(-u))$. The four numbers w_{jk} are the weights, τ_j are the time constants, and θ_j are the thresholds. The function $f(u)$ is the firing rate as a function of the inputs. This system has an incredible range of different phase-portraits. Type this in as an ODE file.

   6a. Using the following values for the parameters: $w_{11} = 12$, $w_{12} = -6$, $w_{21} = 13$, $w_{22} = -2$, $\theta_1 = 3.5$, $\theta_2 = 6$, $\tau_1 = 1$, and $\tau_2 = 1$, draw the nullclines. There are three fixed points. The middle fixed point is a saddle. Draw the unstable and stable manifolds. Note that the two branches of the unstable (yellow) manifolds form a loop which terminates on the left-most fixed point. Now decrease θ_1 to around 3.2 and draw the nullclines. Note that the two lower-left fixed points have disappeared. Integrate the equations to see what happens. There is a stable limit cycle that has emerged from the loop formed by the unstable manifolds of the saddle-point (which no longer exists). This is a classical bifurcation of a large amplitude periodic orbit from a saddle-node on a circle. There is a value of θ_1 in which the saddle-point and the stable-node coalesce into one point. Try to find this point. (**Hint:** It is between 3.2 and 3.5.) Change τ_2 to 0.9 and θ_1 back to 3.5. Again look at the stable and unstable manifolds for the saddle-point. Change θ_1 to 3.4 and redo the manifold calculation. Note that the right branch of the unstable manifold now terminates on a limit cycle. For θ_1 between 3.4 and 3.5, the right branch of the unstable manifold, rather than terminating on the stable node, terminates back in the saddle along the stable manifold. This is called a homoclinic loop. For θ_1 slightly smaller than this critical value, a stable limit cycle emerges from this homoclinic. In this parameter regime there are three fixed points (the left-most stable) and a stable limit cycle. The limit cycle

that emerges from the homoclinic orbit is stable because the so-called *saddle-quantity*, $\lambda_1 + \lambda_2$, is negative, where λ_j are the eigenvalues for the saddle-point. (Check that this is true!)

6b. **More fun with limit cycles.** Verify that there are three limit cycles for the neural network model and one fixed point for the parameters $w_{11} = 8$, $w_{12} = -6$, $w_{21} = 16$, $w_{22} = -2$, $\theta_1 = 0.34$, $\theta_2 = 2.5$, $\tau_1 = 1$, and $\tau_2 = 6$. Determine their stability. (**Hint:** Unstable limit cycles in planar systems can be found by integrating backwards.) Change $\theta_1 = 0$, $\theta_2 = 3.55$, and $w_{21} = 7$ and show that there is a stable limit cycle, a stable fixed point, and an unstable limit cycle. This bistability occurs for a completely different mechanism from the bistability induced by the homoclinic bifurcation above.

6c. Find some parameters such that there are nine fixed points! (**Hint:** Try w_{11}, w_{22} large and positive (say, around 10), θ_1, θ_2 positive, and w_{12},w_{21} small and negative.)

4.6 Three and higher dimensions

XPPAUT is able to create three-dimensional plots of higher dimensional dynamical systems that are occasionally encountered in textbooks. Here, we will examine a few classic examples and then show some fancy projection plots. Consider the Lorenz equations which are an approximation of a large-scale atmosphere model. Here are the equations:

$$\frac{dx}{dt} = s(y - x),$$

$$\frac{dy}{dt} = rx - y - xz,$$

$$\frac{dz}{dt} = -bz + xy.$$

The standard parameters are $s = 10$, $r = 27$, $b = 8/3$. The *XPPAUT* file that I show below is rather complicated since I will do some fancy graphics tricks. Here is the idea: I will draw a three-dimensional plot and then, on one or more of the coordinate planes, plot the corresponding two-dimensional projections, as in Figure 4.1. We will also explore Poincaré maps, Fourier spectra, and Liapunov exponents with this example. Here is the ODE file, `lorenz.ode`:

```
# the lorenz equations
# with some fancy graphics options
x'=s*(-x+y)
y'=r*x-y-x*z
z'=-b*z+x*y
par r=27,s=10,b=2.66
init x=-7.5,y=-3.6,z=30
# now add some projection planes for 2D plots
```

```
aux x2=xplane
aux y2=yplane
aux z2=zplane
par xplane=-35,yplane=60,zplane=-10
# set up the numerics
@ total=50,dt=.02
# set up 3D plot
@ xplot=x,yplot=y,zplot=z,axes=3d
# tell XPP there are 4 plots altogether
@ nplot=4
# here are the 3 projections
@ xp2=x,yp2=y,zp2=z2
@ xp3=x,yp3=y2,zp3=z
@ xp4=x2,yp4=y,zp4=z
# set up the 3D window
@ xmin=-40,xmax=18,ymin=-24,ymax=64,zmin=-12,zmax=45
@ xlo=-1.4,ylo=-1.7,xhi=1.7,yhi=1.7
# and rotate the plot a bit so it looks nice
@ theta=35
done
```

Notes. As I mentioned, this looks quite complicated for a simple ODE. The first three lines after the comments are the actual equations. Then, I add three auxiliary variables which are just constants. So if I plot `(x,y,z2)`, for example in three dimensions, the z coordinate is held constant. This results in a projection on the (x, y) plane. The choice of planes was done after some experimentation in order to give the best view. The rest of the file is simply setting *XPPAUT*'s plot parameters (all of which could be done within the program). The comments should make clear exactly what I am doing. The parameters `phi`, `theta` control the view angle for three-dimensional graphs.

Run *XPPAUT* with this file and integrate the equations. You should see something like Figure 4.12. On the screen these three two-dimensional projections appear in various fall colors with the big white Lorenz attractor in the middle. The axes have labels in the figure. To label axes, click on `View axes 3D` and fill in the last three dialog box entries. We can rotate this graph around the vertical axis in the form of a little movie to get a better view of the attractor. Click on `3D params` and fill in the right-side of the dialog box as follows:

`Movie(Y/N):` Y
`Vary (theta/phi):` theta
`Startangle:` 35
`Increment:` 15
`Number increments:` 23

This tells *XPPAUT* to make a series of 23 screen captures in which the rotation angle `theta` is varied by 15 degrees at a time starting with 35 degrees. These screen shots can be played

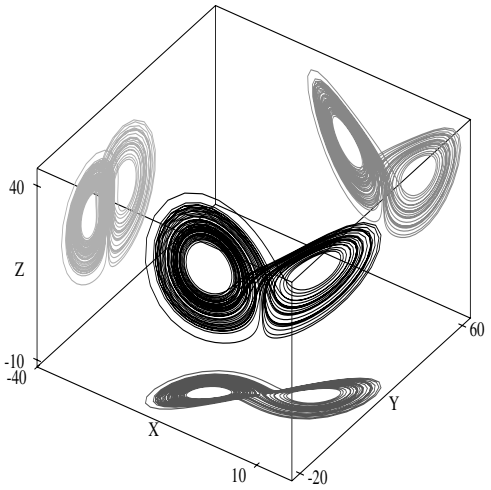

Figure 4.12. *The Lorenz attractor.*

back smoothly like a movie using the `Kinescope` command. Click on `Ok` after you have filled in the dialog box. (Make sure that the window in which the graph is drawn is not obscured by any window other than, at most, the dialog box. Otherwise the capture will not work properly.) A series of pictures will be drawn on the screen. These are captured in a buffer and can be played back either one frame at a time or in smooth succession using the `Kinescope` command. Click on `Kinescope Autoplay` to smoothly run through all the captured screens. You will be prompted for the number of loops you would like (choose 3) and the amount of time between snapshots in milliseconds (50–100 are reasonable values). You will then see the three-dimensional plot smoothly rotated and this is repeated three times. If you want to manually step through the screen shots, choose `Kinescope Playback`. Clicking any mouse button advances to the next frame. Clicking `Esc` gets you out of the loop. You can save these as an animated GIF file also by using the `Kinescope Make Anigif` command. The file is always saved as `anim.gif`.

4.6.1 Poincaré maps, FFTs, and chaos

The Lorenz equation is famous for being one of the first numerical examples of chaotic behavior. There are many ways to demonstrate chaos in a deterministic dynamical system, one of the most common is the existence of infinitely many unstable periodic orbits. There is a classic mathematical theorem due to Sarkovskii, who showed that for discrete dynamical systems, if there is period 3 orbit, then there are infinitely many periodic orbits. This was later reproved by Li and Yorke in a classic paper [27]. Recall that we found all the period 3 orbits in the logistic equation in section 4.2.2. Another (and better known) way to demonstrate chaos is through sensitive dependence on initial conditions. That is, if one starts with two initial conditions close to each other, the solutions will diverge from each other at an exponential rate.

Sensitive dependence

Delete all the graphs other than the original plot of the Lorenz equation by clicking `Graphic stuff` `Remove all` (**G R**). Then plot X as a function of time in your window by clicking **X** and choosing `x` at the prompt. Freeze this curve (**G F F**) and choose color 1 (red). Now in the **Initial Data Window**, change the initial value of `z` from `30` to `30.001` and then click on the `Go` button. The trajectories are pretty close up to about $t = 10$ after which they diverge. This gives a ballpark estimate of the rate of divergence. The exact rate of divergence is governed by the maximal Liapunov exponent. This is a numerically determined quantity which is found by integrating a linear differential equation obtained by linearizing the original equation around the trajectory. By definition, the divergence $d(t)$ away from the trajectory is approximately

$$d(t) = d(0)\exp(\lambda t),$$

where λ is the maximal exponent. Taking $t = 10$, $d(0) = 0.001$, and $d(10) = 1$ we get that $\lambda \approx \ln(1000)/10 = 0.69$ which is close to the actual published value of about 0.9. You can do better than this by clicking on `nUmerics` `Stochastic` `Liapunov` (**U S L**), choosing `0` to not get a range of values, and then clicking on `Esc` to get back to the main menu after the exponent is presented. The number you get here is also a little low. You can improve the estimate by integrating away all the transients and integrating for a long time. If you use `Initialconds` `Last` (**I L**) and extend the time of integration to 99, you will get a value closer to 0.9. The main point is that the maximal Liapunov exponent is **positive** which implies that there is divergence of nearby trajectories. (**Note:** *XPPAUT* computes the maximal Liapunov exponent along a computed trajectory by linearizing about each point in the trajectory, advancing one time step using a normalized vector, computing the expansion, and summing the log of the expansion. The average of this over the trajectory is a rough approximation of the maximal Liapunov exponent.)

We can see this sensitivity in a rather dramatic fashion by solving the Lorenz equations over 50 different initial conditions that are close to each other and by then looking at the solutions in a two-dimensional projection. We will use the animator for this and create an ODE file with 50 independent versions of the Lorenz equation. The file is called `lorenz50.ode` and here it is:

```
# 50 lorenz equations
x[1..50]'=s*(y[j]-x[j])
y[1..50]'=r*x[j]-y[j]-x[j]*z[j]
z[1..50]'=-b*z[j]+x[j]*y[j]
par r=27,s=10,b=2.66
init x[1..50]=-7.5,y[j]=-3.6
init z[1..50]=30.00[j]1
@ total=50,dt=.02
done
```

Notes. I have used an idiosyncratic aspect of the *XPPAUT* file reader to initialize `z[j]`. The expression `[j]` is expanded to the number value of `j` so that if `j=15`, the expression `30.00[j]1` is expanded to `30.00151`. Thus, the initial data are all close but differ

slightly! The trailing "1" is included to distinguish `30.0041` (`j=4`) from `30.00401` (`j=40`).

Run this program and integrate the equations (**I G**). Now, click on `View axes Toon` to bring up the animation window. The animation file for this illustration is called `lorenz50.ani` and has the following form:

```
# animation for lorenz50.ode
fcircle (x[1..50]+20)/40;z[j]/50;.02;[j]/50
done
```

What this does is plot little colored balls in the (x, z) plane, where the coordinates have been suitably scaled. (If you want to know exactly what is going on in this animation file, see Chapter 8.) In the **Animation Window**, click on `Go` to start the animation. You will see a single little ball that becomes a comet as the individual trajectories diverge and eventually, the attractor is filled out by the 50 little balls.

Exit from the file `lorenz50.ode`. Now we will get back to work on the Lorenz equation. If you closed the original Lorenz file, you should start it up again and remove the projection graphs. Integrate the equation and plot x against time. Now, we will look at the power spectrum. Click on `nUmerics Stochastic Power` and choose `X` at the prompt. Then click on `Esc` to get back to the main menu. The power is in the second data column (`x`) and the mode number is in the first data column (`t`), so click on `Xvst` and choose `X` to plot the spectrum. You may want to zoom into the lowest 200 or so modes. Notice the spectrum has no real peaks, which is indicative of chaos.

4.6.2 Poincaré maps

As a final look at the Lorenz attractor, we will compute a Poincaré map. For our purposes, a Poincaré map consists of a discrete set of values chosen whenever one of the variables passes through some prescribed value. In his seminal paper, Lorenz chose to plot successive maxima of the variable z. We will do this now and will see that the map turns out to be essentially one-dimensional. Note that since we are fixing one variable in a three-dimensional system, we expect to see a two-dimensional map. This computation may take a few seconds, as we have to make a small time step (for accuracy in finding the maximum) and also, we should integrate for a long time. Click on `nUmerics` to get the numerics menu. Click on `Total` and change the total to 200. Click on `Dt` and change the time step to 0.01. Now click on `Poincare map` and `Max/min` to select maxima or minima of a variable. Fill in the resulting dialog box as follows:

`Variable: Z`
`Section: 0`
`Direction(+1,-1,0): 1`
`Stop on sect (y/n): N`

and click on `Ok`. What you have done is tell *XPPAUT* that you want to plot all the variables every time Z has a local maximum (the $+1$ direction is for local maxima; -1 is for local minima; 0 is for both) and to not stop when this maximum is crossed. Escape to the main menu (`Esc`). Now, integrate the equations. This could take several seconds. Now, we have

a series of values (t_n, x_n, y_n, z_n) that are the values of the variables at every time that $z(t)$ has a local maximum. Lorenz plotted z_{n+1} against z_n to obtain a nice one-dimensional map. *XPPAUT* allows you to do this very easily. We will first set up a view with z plotted against itself. Click on `Viewaxes 2D` and fill in the resulting dialog box as follows:

`X-axis: Z`	`Xmax: 45`
`Y-axis: Z`	`Ymax: 45`
`Xmin: 30`	`Xlabel:`
`Ymin: 30`	`Ylabel:`

and click on `Ok`. You will see a diagonal line from the lower left to the upper right. Click on `Graphics Edit` and choose `0`. Change the `Linetype` to -1 and click `Ok`. The diagonal line becomes a series of points. Now, how can we see the map? First, let's freeze this diagonal line of points since, with maps, it is always nice to plot the line $y = x$. Click on `Graphics Freeze Freeze` and then click on `Ok`. The diagonal dots become a solid line. *Finally*, we can plot the curve z_{n+1} against z_n. *XPPAUT* has a command called `rUelle` named after the famous mathematician, David Ruelle, who proved that you could reconstruct the dynamics of arbitrary systems from the time series of one variable $x(t)$ by looking at the points $(x(t), x(t-\tau_1), \ldots, x(t-\tau_n))$. Thus, we will look at (z_{n-1}, z_n). We thus want to shift the values on the x axis up by one. Click on `nUmerics rUelle` (**U U**) and change the `X-axis shift` to `1`. Click `Escape` to exit the numerics menu and click on `Restore` to restore the plot. You should see a cusp-like curve that is almost one-dimensional. It intersects the diagonal at around $z = 38.5$, and this corresponds to a periodic point. If you are reasonably good at guessing formulas for graphs from their shape, you might try to approximate this curve by some function of z. Here is my guess:

$$f_{lorenz}(z) = 27 + 18\exp(-|z - 37.3|/5). \tag{4.5}$$

In any case, we see that the Lorenz equations can be well approximated by a one-dimensional map. How do we find other periodic points? If we change the `X-axis shift` to `2`, for example, we will be plotting z_{n-2} against z_n so that intersections with the diagonal are period 2 and period 1 points. Try this and you will find two period 2 points (one period 2 orbit). Now for the final blow, let's find the period 3 points. Change the `X-axis shift` to 3 in the Ruelle plot and redraw the graph. You should see six intersections in addition to the trivial one corresponding to two period 3 orbits. Figure 4.13 shows the numerically computed map and the second and third iterates. In Chapter 9 (section 9.6.4) we present a different way to make Poincaré maps which can be significantly faster.

Through three lines of evidence, (i) sensitivity to initial conditions, (ii) complex spectrum, and (iii) period 3 orbits, you can be reasonably sure that the Lorenz equations are chaotic. In fact, there are only a few rigorous results on this system.

4.6.3 Exercises

1. Explore the simple map f_{lorenz} (equation (4.5)) and verify that the maximal Liapunov exponent is about 0.64 so that it is "not as chaotic" as the real Lorenz system. Find values for the period 1, 2, and 3 orbits. Integrate this for 4000 iterates and try to show that there are six different period 5 orbits.

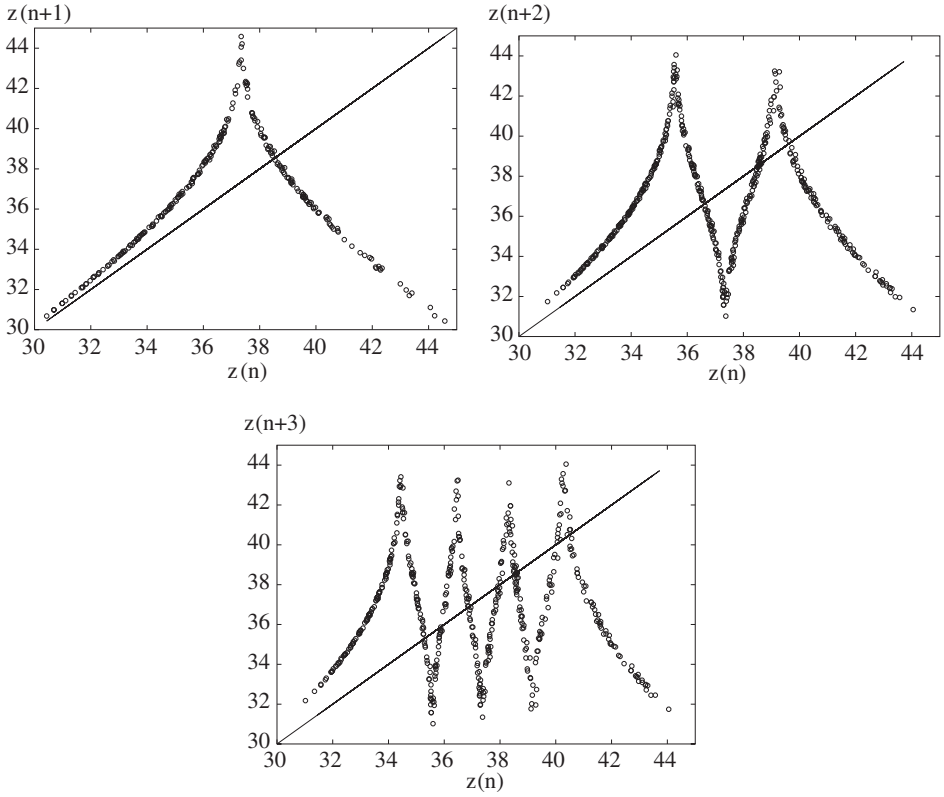

Figure 4.13. *Poincaré map for the Lorenz attractor looking at successive values of the maximum of z. Period 1, 2, and 3 points are illustrated.*

2. Analyze the Rossler equation

$$x' = -y - z,$$
$$y' = x + ay,$$
$$z' = bx - cz + xz,$$

where $a = 0.36, b = 0.4, c = 4.5$ with initial data $(x, y, z) = (0, -4.3, -0.054)$. First integrate it for a total of 200 with a time step of 0.1. Look at the time series and various phase-space projections. Look at the equation in three dimensions. (Note that the `Window/zoom Fit` is invaluable here since this will compute the scales for you!) Compute the Liapunov exponent. Compute the power spectrum. You should see a big peak and some side peaks, but there are many other frequencies. However, it is not nearly as chaotic as the Lorenz system, as evidenced by the dominance of certain frequencies and the Liapunov exponent that is close to 0.

Compute a Poincaré map for this attractor as follows. Instead of selecting `Max/min` choose `Section`. Choose `X` as the variable, `0` as the section, and +1 as the direction.

A point will be stored each time that x crosses 0 with a positive derivative. Integrate for a total of 2000. Plot y_{n-1} against y_n using the Ruelle plot. The resulting map is essentially one-dimensional and looks like an upsidedown logistic map. Find period 2 and period 3 points. For fun, simulate the map

$$y_{n+1} = -6 + 0.65(y_n + 3.5)^2$$

with $y_0 = -5$ which is a reasonable fit to the true map. Find the period 1, 2, and 3 points for this.

3. Simulate the Chua circuit whose *XPPAUT* file is given below:

```
# chuas scroll chaos
x'=a*(y-if(x>=1)then(m1*x+(m0-m1))\
          else(if((x+1)>0)then(m0*x)else(m1*x-(m0-m1))))
y'=x-y+z
z'=-b*y
par a=9,b=14.28
par m0=-0.1428,m1=0.2856
init x=.1,y=.1,z=.1
@ total=200
done
```

Plot it in three dimensions and see why it is called scroll chaos. Compute the spectrum and the Liapunov exponent. Take a few different Poincaré sections; see if you can find one which reveals a one-dimensional map lurking beneath.

4. Chaos occurs in two-dimensional systems as well if they are periodically forced. The classic example is the forced Duffing equation which we will write as

$$\ddot{x} + x(x^2 - 1) + f\ddot{x} = a\cos t$$

We can rewrite this as a system

$$x' = v, \qquad v' = -fv + a\cos(t) - x(x^2 - 1).$$

Write an ODE file. Set $a = .3$ and $f = .25$. Integrate this for a total of 200 and look at it in the (x, v) plane. Window the plane $[-1.5, 1.5] \times [-1, 1]$. Now, we will take a Poincaré map with respect to t, plotting a point every time t hits multiples of 2π. If you choose your section variable to be t, then *XPPAUT* assumes it is periodic and plots a point every time t is a multiple of the section value. Thus, to get a map, choose the Poincaré map option, choose `Section`, choose `T` as the variable, and type in `6.283185307` for the section. Change the total time to integrate to 10,000. Then in the main menu, change the linetype to 0 so dots are pointed (`Graphics stuff Edit`) and choose 0 as the graph to edit. Finally, turn off the axes by clicking `Graphics stuff aXes opt` and change both `X-org` and `Y-org` from 1 to zero. Now run the simulation. You will see a beautiful folded map. This structure arises due to something called a Smale horseshoe that can be found in this system. The

horseshoe was one of the first examples of rigorously proven chaos in a dynamical system. If you let the friction f and the forcing a be small in the forced Duffing equation, it is possible to rigorously prove that there is a Smale horseshoe for the system by simply finding some zeros of a certain integral! This is one of the few general methods we have of rigorously proving chaos in a dynamical system.

A simple map that has a similar folded structure is the Henon map,

$$x_{n+1} = 1 - ax_n^2 + y_n, \quad y_{n+1} = jx_n,$$

which is a two-dimensional map. Here is the corresponding ODE file, `henon.ode`:

```
# the henon map
x'=1-a*x^2+y
y'=j*x
par a=1.4,j=.3
init x=.316,y=.206
@ xp=x,yp=y,xlo=-1.3,xhi=1.3,ylo=-.4,yhi=.4
@ total=20000,meth=discrete,lt=0,maxstor=40000
done
```

It is set up to iterate 20,000 times. I have to tell *XPPAUT* to increase the storage, `maxstor=40000`, so that we can keep all the points. I have also told *XPPAUT* to use linetype 0. Try this and then zoom into the folded parts. You will see that it is like pastry all folded up tighter and tighter.

5. The Field–Worfolk equations arise as a four-dimensional normal form for certain symmetric bifurcations. The equations are

$$\begin{aligned} x' &= (\lambda + ar^2 + by^2 + cz^2 + dw^2)x + eyzw, \\ y' &= (\lambda + ar^2 + bz^2 + cw^2 + dx^2)y - exzw, \\ z' &= (\lambda + ar^2 + bw^2 + cx^2 + dy^2)z + exyw, \\ w' &= (\lambda + ar^2 + bx^2 + cy^2 + dz^2)w - exyz, \end{aligned}$$

where $r^2 = x^2 + y^2 + z^2 + w^2$ and a, b, c, d, e are parameters. The parameter λ is the bifurcation parameter. The equations are notable because for some values of $a - e$, as λ crosses 0, there is bifurcation to instant chaos. Try $a = -1$, $b = 0.1$, $c = -.05$, $d = .015$, $e = .55$, and vary λ. Use initial conditions $x = y = w = 0.1$ and $z = 0.2$. You should integrate these up to $t = 4000$. Here is a possible *XPPAUT* file:

```
# field-worfolk equations
x' = (l+a*r+b*y^2+c*z^2+d*w^2)*x+e*y*z*w
y' = (l+a*r+b*z^2+c*w^2+d*x^2)*y-e*x*z*w
z' = (l+a*r+b*w^2+c*x^2+d*y^2)*z+e*x*y*w
w' = (l+a*r+b*x^2+c*y^2+d*z^2)*w-e*x*y*z
r=x^2+y^2+z^2+w^2
par l=1,a=-1,b=.1,c=-.05,d=-.1,e=1
```

```
set fw1 {l=1,a=-1,b=.1,c=-.05,d=-.1,e=1}
set fw2 {l=1,a=-1,b=.1,c=-.05,d=.015,e=.55}
set fw3 {l=1,a=-1,b=.1,c=-.085,d=.005,e=.572837}
init x=.1,y=.1,z=.2,w=.1
@ maxstor=20000,dt=.5,meth=qualrk,tol=1e-5,total=4000
done
```

Try the second set of parameters, `fw2`.

6. As a final exercise, consider the simplest equation (that I know of) that has chaotic behavior and approaches it through a series of period-doubling bifurcations:

$$\begin{aligned} x' &= y, \\ y' &= z, \\ z' &= -cx - by - az + x^2. \end{aligned}$$

Choose $a = 1$, $b = 2$, and let c vary from 3 to 3.5. This system is very similar to the Rossler attractor.