Chapter 3
Writing ODE Files for Differential Equations

3.1 Introduction

XPPAUT is equipped with a powerful parser that has many useful functions and tools that make it easy to solve problems that are related to dynamical systems. Furthermore, there are many tricks that you can use to solve very complicated problems. In Chapter 9, I provide a survey of the more obscure tricks. Perhaps the best way to describe the range of ODE files is to give you a number of examples sorted by problem type. Appendix D gives a complete description of the language for the creation of ODE files. Every ODE file is a plain text file. The total length of lines must be less that 1000 characters. Line continuation is done with the \ character as follows:

```
x' = y \
 + z + \
w
```

This is interpreted as `x' = y+z+w`.

Every ODE file consists of declarations of the equations that you want to solve, the parameters involved, and any user-defined functions that you will need. Every ODE file must end with the `done` statement. Other than that, the form of the file is pretty flexible. **Note.** The name of anything that is user defined in *XPPAUT* must be fewer than 10 characters. The allowable characters are the letters of the alphabet, numbers, and the underscore symbol "_". Other symbols are not recommended, since they may be confused with symbols that have meaning to *XPPAUT*. Do not start names with numbers.

Most of the input that you type into an ODE file is pretty much as you'd expect and similar to how you would write it on paper. The main exceptions to this are (i) you must use the `*` symbol between items that are to be multiplied, and (ii) *XPPAUT* is case insensitive.

All the examples here and in the rest of the book are available from the *XPPAUT* home page so that you need not type them in. I also encourage you to run the example files and put in some initial conditions or parameters so that you become familiar with simulation. In many of the examples, I suggest some interesting things to explore.

3.2 ODEs and maps

The easiest kinds of equations to type in are ODEs or maps. You type these in almost exactly the same way you would write them on paper. There are two different ways to declare a differential equation. Consider the following ODE:

$$\frac{dx}{dt} = -x.$$

Then the corresponding line in an ODE file will be either

```
x' = -x
```

or

```
dx/dt = -x
```

I will generally use the former since it requires less typing. (**Note.** The spaces are not required and are just for readability.)

There are two different ways to define a map. Consider

$$x_{n+1} = x_n/2.$$

You could write this in two ways:

```
x' = x/2
```

or

```
x(t+1) = x/2
```

Both are interpreted the same way. Again, I will use the former, as it requires less typing.

Higher order differential equations and maps must be rewritten as systems of first order equations. Thus, the classic damped spring,

$$\frac{d^2x}{dt^2} + f\frac{dx}{dt} + k(x-l) = 0,$$

would be written as

$$\frac{dx}{dt} = v, \qquad \frac{dv}{dt} = -fv - k(x-l)$$

and the corresponding ODE file is written as

```
# the spring
x'=v
v'=-f*v-k*(x-l)
par f=0,l=1,k=1
done
```

where I have assigned some arbitrary values to the parameters as well as a comment.

Consider the delayed logistic map,

$$x_{n+1} = rx_n(1 - x_{n-1}),$$

which is a second order map. To solve this in *XPPAUT* we rewrite it as a system of two first order maps,

$$x_{n+1} = y_n, \qquad y_{n+1} = x_{n+2} = rx_{n+1}(1 - x_n) = ry_n(1 - x_n).$$

Thus the ODE file is written as

```
# delayed logistic
x(t+1)=y
y(t+1)=r*y*(1-x)
par r=2.1
init y=.25
@ total=200
done
```

I have added an initial condition statement for y. I have also set the number of iterations to 200 instead of the default of 20. The "t+1" tells *XPPAUT* that the file should be considered a model with discrete dynamics so that the method of solving the equations is automatically set to `discrete`.

Alternatively, I could save typing by also writing this file as

```
# delayed logistic
x'=y
y'=r*y*(1-x)
par r=2.1
init y=.25
@ meth=discrete,total=200
done
```

(Here, since I don't use the construction t+1, I have to tell *XPPAUT* that the model is discrete. *XPPAUT* defaults to a continuous differential equation.) Start up *XPPAUT* with this file (e.g., `xpp delaylog.ode`). Solve the equations by clicking `Initialconds Go` (**I G**). Then expand the viewing window by clicking on `Window Window` and changing `X Hi` to 200. Change the parameter r, making it smaller or larger, and watch what happens. If r is too big, you will see that solutions blow up (i.e., they go out of bounds).

3.2.1 Nonautonomous systems

XPPAUT uses `t` to denote the independent variable. For differential equations and other continuous dynamical systems, `t` represents continuous time. For maps, it always represents a natural number 0, 1, 2, ... Consider the forced Duffing equation,

$$\frac{d^2x}{dt^2} + f\frac{dx}{dt} + ax(x^2 - 1) = c\cos\omega t.$$

As above, we write this as a pair of first order equations. The resulting ODE file `duffing.ode` is written as

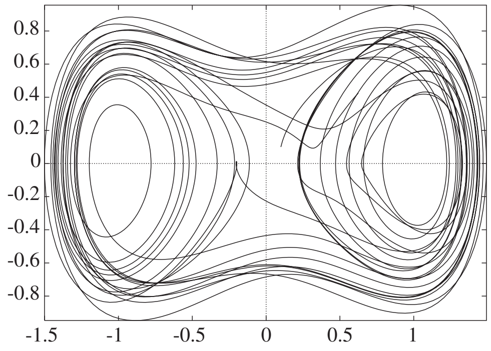

Figure 3.1. *Phase-plane of the forced Duffing equation.*

```
# forced duffing equation
x'=v
v'=a*x*(1-x^2) -f*v+c*cos(omega*t)
par a=1,f=.2,c=.3,omega=1
init x=.1,v=.1
done
```

Try running this equation. Change the total integration time to 200 by clicking on `nUmerics Total` and entering 200. Then click on `Esc` and integrate the equations with `Initialconds Go`. Click on `Window Fit` to fit the entire time series into the window. Create a new window by clicking on `Makewindow Create`. Stretch it out to make it bigger. Then make a plot of the phase-plane by clicking the little boxes next to the variables `x` and `v` in the **Initial Data Window** and then clicking on the `xvsy` button in the **Initial Data Window**. You should see something like Figure 3.1.

In a similar vein, we can write nonautonomous maps in the same manner. For example, the logistic equation with slowly increasing carrying capacity,

$$x_{n+1} = rx_n\left(1 - \frac{x_n}{1+an}\right),$$

becomes

```
# time dependent carrying capacity
par a=.01,r=1.8
init x=.1
x(t+1)=r*x*(1-x/(1+a*t))
@ meth=discrete,total=200
done
```

3.3 Functions

XPPAUT comes with all the usual functions you would expect with the standard names (see Table 3.1). In addition, there are a number of less standard functions that you can use in the program.

`sin(x)`	$\sin(x)$	`cos(x)`	$\cos(x)$	`tan(x)`	$\tan(x)$
`atan2(x,y)`	$\tan^{-1}(y/x)$	`asin(x)`	$\sin^{-1}(x)$	`acos(x)`	$\cos^{-1}(x)$
`atan(x)`	$\tan^{-1}(x)$	`sinh(x)`	$\sinh(x)$	`cosh(x)`	$\cosh(x)$
`tanh(x)`	$\tanh(x)$	`x**y,x^y`	x^y	`exp(x)`	e^x
`abs(x)`	$\|x\|$	`ln(x)`	$\ln(x)$	`log(x)`	$\ln(x)$
`log10(x)`	$\log_{10}(x)$	`sqrt(x)`	$\sqrt{x}$	`max(x,y)`	$\max(x,y)$
`min(x,y)`	$\min(x,y)$	`sign(x)`	$x/\|x\|$	`heav(x)`	if $x \geq 0$ then 1 else 0
`flr(x)`	$\mathrm{int}(x)$	`erf(x)`	$\mathrm{erf}(x)$	`mod(x,y)`	x modulo y
`erfc(x)`	$1-\mathrm{erf}(x)$	`x & y`	x AND y	`bessely(n,x)`	$Y_n(x)$
`x | y`	x OR y	`not(x)`	$\tilde{}x$	`besselj(n,x)`	$J_n(x)$

Table 3.1. *Names of standard XPPAUT functions.*

The following logical functions take values of 1 or 0, depending on the truth of the (in)equality, `x>y`, `x<y`, `x==y`, `x<=y`, `x>=y`, `x!=y`. The last means x NOT EQUAL y. (For all these logical operations, it is best to surround the right and left sides by parentheses.) Here are the remaining simple functions:

- `ran(x)` produces a uniformly distributed random number between 0 and x.
- `normal(x,s)` produces a normally distributed random number with mean x and standard deviation s.
- `if(x)then(y)else(z)` returns y if x is true; otherwise it returns z.
- `sum(x,y)of(z)` produces $\sum_{i'=x}^{i'=y} z$ where z is some expression involving the index `i'`. For example, `sum(0,10)of(i')` evaluates to 55. The summation variable is always `i'`.

There are other complicated functions that act directly on variables; we will describe them shortly.

3.3.1 User-defined functions

It is easy to define functions of up to nine variables in *XPPAUT*. A function is defined with a statement such as

```
f(x,y) = x/(x+y)
```

Functions can depend on other functions, but be careful of recursive definitions, as *XPPAUT* does not check for this and will unceremoniously crash if such a function is called. (A legitimate recursive function will not cause a crash, e.g., one that stops, such as `fac(x)=if(x>0)then(x*f(x-1))else(1)`.) For example, here is an ODE file, `wc.ode`, for the Wilson–Cowan equations:

```
# the wilson-cowan equations
u'=-u+f(a*u-b*v+p)
v'=-v+f(c*u-d*v+q)
f(u)=1/(1+exp(-u))
par a=16,b=12,c=16,d=5,p=-1,q=-4
@ xp=u,yp=v,xlo=-.125,ylo=-.125,xhi=1,yhi=1
done
```

Note that, in the ODE file, I have defined the logistic function $f(x)$ and also included a line so that, when the program is run, the graphics window displays the (u, v) phase-plane. This is done on the line beginning with the @ symbol. Here, the statement `xp=u` says that the variable to be plotted on the x-axis is u, `yp=v` says to plot v on the y-axis, and the remaining statements set up the window size. Fire this up and look at the phase-plane as you change parameters. For example, draw the nullclines by clicking `Nullclines New`. Try choosing different initial conditions with the `Initialconds Mice` command. Vary the parameter `a` by making it lower. Figure out at what point the limit cycle disappears. To automate this, attach the parameter `a` to one of the parameter sliders. Give it a range between 0 and 20.

3.4 Auxiliary and temporary quantities

In *XPPAUT*, the values of every one of the variables that define the dynamical system are available for plotting, storing, and manipulating. However, sometimes there are other quantities that you would like to see, such as the total energy of a system or one of the currents in a model for a cell membrane. *XPPAUT* allows you to define these within an ODE file so that they are accessible for plotting, etc. This is done by writing a line that starts with `aux` followed by the definition of the quantity, e.g.,

```
aux stuff=x+y*sin(t)
```

`stuff` will now be plottable. Let's take as an example the pendulum which satisfies

$$ml^2\ddot{\theta} = -mgl\sin\theta,$$

where l is the length, g is gravity, and m is the mass of the pendulum. θ is the angle of the pendulum with respect to the vertical axis. The kinetic energy is $K = m(l\dot{\theta})^2/2$ and the potential energy is $P = mgl(1 - \cos\theta)$. Thus, the total energy is

$$E = m(l\dot{\theta})^2/2 + mgl(1 - \cos\theta).$$

Let's write an ODE file for the pendulum and make the energy a plottable quantity. As usual, we want to first make the equation a system of two first order equations,

$$\dot{\theta} = v, \qquad \dot{v} = -(g/l)\sin\theta.$$

The ODE file is written as

```
# the undamped pendulum
theta'=v
v'=-(g/l)*sin(theta)
par g=9.8,l=2,m=1
aux E=m*((l*v)^2/2+g*l*(1-cos(theta)))
done
```

Now if you run this, the energy will be accessible. You will find that it is constant along solutions to the differential equation—a hallmark of frictionless mechanical systems. Try different initial conditions. The energy is always conserved. In *XPPAUT*, auxiliary quantities that are made available for plotting, etc. are called **auxiliary variables**. Rewrite the above ODE file as follows to incorporate a linear friction term with magnitude f:

$$\dot{\theta} = v, \qquad \dot{v} = -(g/l)\sin\theta - fv.$$

Run this, plot the energy for $f = 0$ and $f = 0.1$, and notice how the energy dissipates to 0 when there is friction.

3.4.1 Fixed variables

If you have a particularly complicated equation with quantities that are used repeatedly, then it is often useful, for both computational ease and readability, to define them as temporary quantities. Say the quantity is called `z`. Then you just include a statement such as

```
z=x+y*sin(t)
```

and that quantity is defined and can be used in the ODE. For example, consider the nonlinear oscillator

$$\frac{dx}{dt} = x(a-R) - y(1+qR), \qquad \frac{dy}{dt} = y(a-R) + x(1+qR),$$

where $R = x^2 + y^2$. We should define R in the ODE file and then write the equations as

```
# standard nonlinear oscillator
R=x^2+y^2
x'=x*(a-R)-y*(1+q*R)
y'=y*(a-R)+x*(1+q*R)
par a=1,q=1
done
```

You can have many such temporary quantities and they also can depend on each other. In *XPPAUT* these are called **fixed variables**.

Important notes

- Auxiliary variables are not known to *XPPAUT* and so you cannot use them in any formulas.
- Fixed variables are not accessible to the user—they cannot be plotted, but can be used in formulas.
- The order of definition for fixed variables is important, as they are evaluated in the order in which they are defined. Thus, if you define a quantity `x` that depends on another quantity `y`, then you should define `y` first or you will get unintended results.

Just remember that, if you want to use a variable in a formula, make it a fixed variable. If it is just an auxiliary quantity you want to plot, make it an auxiliary variable. You can use fixed variables in definitions of auxiliary variables but not vice versa.

In the oscillator example above, we could make the fixed variable `R` available for plotting by adding a statement

```
aux myr=R
```

so that `myr` is the user-accessible version of `R`.

3.4.2 Exercises

1. Write an ODE file for the Fibonacci recurrence

$$f_{n+1} = f_n + f_{n-1}, \qquad f_0 = f_1 = 1.$$

2. Write an ODE file for the Lotka–Volterra equations

$$\frac{dx}{dt} = ax - bxy, \qquad \frac{dy}{dt} = -cy + dxy$$

with positive parameters a, b, c, d. Track the quantity

$$Q = a \ln|y| + c \ln|x| - by - dx$$

with initial data $x = y = \frac{1}{2}$.

3. Write a differential equation file for the Morris–Lecar equations defined as follows:

$$C\frac{dV}{dt} = -g_L(V - E_L) - g_{Ca}m_\infty(V)(V - E_{Ca}) - g_K w(V - E_K) + I,$$

$$\frac{dw}{dt} = \phi\frac{w_\infty(V) - w}{\tau_w(V)},$$

$$m_\infty(V) = 1/(1 + \exp(-(V - V_1)/V_2)),$$

$$w_\infty(V) = 1/(1 + \exp(-(V - V_3)/V_4)),$$

$$\tau_w(V) = 1/\cosh((V - V_3)/(2V_4)),$$

where $\phi = 0.8$, $g_K = 8$, $g_{Ca} = 4.4$, $g_L = 2$, $C = 1$, $E_K = -84$, $E_{Ca} = 120$, $E_L = -60$, $V_1 = -1.2$, $V_2 = 9$, $V_3 = 2$, $V_4 = 15$, and $I = 90$. Integrate them and analyze them along the lines that you did in Chapter 2 for the Fitzhugh–Nagumo equations. Look in the (V, w) phase-plane and compute a coexistent periodic solution and a stable fixed point. (If the ODE file is too hard for you to figure out, then, as a last resort, download the ODE file `mlex.ode`.)

4. Write the third order system

$$x''' + ax'' + bx' + cx' = x^2$$

as a system of first order equations. (**Hint:** let $y_1 = x$, $y_2 = x'$, and $y_3 = x''$. Try to write an equation for $y_3' = x'''$.) Write an ODE file for this with parameters $a = 1$,

$b = 2$, $c = 3.7$, and initial data $x = 1$, $x' = 0$, and $x'' = 0$. You will need to integrate this for a long time to see the chaotic orbit. Set the `maxstor` option to something like 20,000 (by adding the line `@ maxstor=20000` somewhere in your ODE file) and set the total time to 400. You may want to look at $x(t)$ versus $x'(t)$ (y_1 versus y_2).

5. Write an ODE file for the Rossler attractor,

$$x' = -y - z,$$

$$y' = x + ay,$$

$$z' = bx - cz + xz,$$

with parameters $a = .36$, $b = 0.4$, and $c = 4.5$ and initial data $x = 0$, $y = -4.3$, and $z = 0$. Set the total integration time to 200. Make a three-dimensional plot by clicking on the little boxes next to the three variables in the **Initial Data Window**, and then click on the `xvsy` button in the **Initial Data Window**.

6. Write and simulate the following differential equation which has three limit cycles:

$$x' = xf(r) - y,$$

$$y' = yf(r) + x,$$

$$f(r) = (1 - r)(2 - r)(3 - r),$$

$$r = \sqrt{x^2 + y^2}.$$

Set the viewing window to $[-4, 4] \times [-4, 4]$. Can you create a differential equation with five limit cycles?

7. As a bonus problem, run some of the examples in the text and some that you have written yourself to see what they do.

3.5 Discontinuous differential equations

3.5.1 Integrate-and-fire models

Consider the integrate-and-fire model for a neuron,

$$\frac{dV}{dt} = I - V,$$

with the "reset" condition that each time V hits some value V_T it is reset to 0. This leads to an oscillation when $I > V_T$. Often these models are coupled to each other through "alpha" functions, $\alpha(t) = b^2 t \exp(-bt)$. Thus, the pair satisfies

$$\frac{dV_1}{dt} = I - V_1 + gs_2(t), \qquad \frac{dV_2}{dt} = I - V_2 + gs_1(t),$$

where, each time t^*, V_j crosses V_T, it is reset to 0 and the quantity $\alpha(t - t^*)$ is added to s_j. *XPPAUT* has a mechanism called **global flags** for incorporating such discontinuities. These are quantities that the integrator checks for zero crossings. If a zero crossing occurs, then the variables can be updated and thus discontinuously changed. The integrators are restarted so that even the adaptive step solvers will work fine. Here is the ODE file for a single integrate-and-fire oscillator, `iandf1.ode`:

```
# integrate and fire model
v'=-v + I
global 1 v-vt {v=0}
par I=1.2,vt=1
@ dt=.01,ylo=0
done
```

The new line is `global 1 v-vt {v=0}`. This asserts that there is a global flag to check. The general syntax for flags is

```
global sign condition {event1;event2;...}
```

The `condition` is evaluated at each time step. If the `sign` is 1 and the condition changes sign from negative to positive, or if the `sign` is -1 and the condition changes sign from positive to negative, then each of the events is done. If the `sign` is 0, then the event occurs only if the condition is identically zero. Events always have the form `x=expr`, where `x` is one of the *variables* and `expr` is some expression. Thus, in the above ODE file, if $V - V_T$ changes from negative to positive (i.e., V crosses threshold from below) then V is reset to 0. The line `@ dt=.01,ylo=0` sets the time step to be 0.01 and the minimum value of the y-axis to be zero. Run this file and integrate the equations, and you will see a nice oscillation. Move the mouse inside the graphics window and hold the left button down while moving the mouse. At the bottom of the main window, the coordinates will be shown. You can use this to compute the period of the oscillation, which should be about 1.8.

Let's write an ODE file for the coupled system. The question is how to implement the alpha function. Since $b^2 t \exp(-bt)$ is a solution to

$$s'' + 2bs' + b^2 s = 0, \quad s(0) = 0, \quad s'(0) = b^2,$$

this suggests letting s_j evolve according to this equation with the condition that each time the voltage crosses threshold, $s_j'(t)$ is incremented by b^2. Here is the corresponding ODE file, `iandf2.ode`:

```
# two integrate and fire models coupled with alpha functions
#  b^2*t exp(-b*t)
#  we solve these by solving a 2d ode
v1' = -v1 + i1 + g*s2
v2' = -v2 + i2  + g*s1
s1'=s1p
s1p'=-2*b*s1p-b*b*s1
s2'=s2p
s2p'=-2*b*s2p-b*b*s2
```

```
#
init v1=.1
par i1=1.1,i2=1.1,vt=1,g=.2
par b=5
global 1 v1-vt {v1=0;s1p=s1p+b*b}
global 1 v2-vt {v2=0;s2p=s2p+b*b}
@ dt=.01,total=100,transient=80
@ xlo=80,xhi=100,ylo=0,nplot=2,yp2=v2
done
```

Here are some comments on the ODE file:

1. We rewrite the second order equations for s as a pair of first order ODEs.

2. We initialize $V_1 = .1$ so that it is not identical to V_2 to break the symmetry.

3. Each time V_1 crosses V_T from below, V_1 is reset to 0 and s_1' (`s1p`) is incremented by b^2. A similar condition is set for V_2.

4. I have set a number of options so that the user doesn't have to worry about them and can just run the integration. I have set the integration time step to 0.01, the total integration to 100 time units, and I will start storing data only after a transient of 80 time units. I set the low and high limits of the x-axis to 80 and 100, respectively. I set the lower limit of the y-axis to 0. I tell *XPPAUT* to plot two curves in the window and tell it that the second curve to plot is V_2. (Note that, by default, the x component of a plot is time t, and the y component is the first variable defined in the file.)

Run this equation in *XPPAUT* and note that both V_1 and V_2 are synchronized. Change the initial conditions however you want (but make sure that $V_j(0) < 1$) and the solution will always go to synchrony. Change b to 1 and integrate again. The oscillations alternate; synchrony is not stable. Now change b back to 5 and change g to $-.2$. Integrate again. Notice that synchrony is unstable. Now, change b to 1 again. Set all variables to 0 except V_1. Set $V_1 = .1$ and integrate. You should see synchronous oscillations. Set $V_1 = .4$ and integrate; the oscillations alternate. This result was proven by van Vreeswijk, Abbott, and Ermentrout [39]. (See Figure 3.2.)

3.5.2 Clocks: Regular and irregular

There are many other examples of models that involve discontinuous right-hand sides. The simple clock is an excellent example. We model a pendulum clock as follows. It consists of a decaying sinusoid that receives a kick each time it reaches its maximum on the left-hand side. Let x be the angle of the pendulum and y be the velocity. We model the decaying sinusoid as

$$\frac{dx}{dt} = -ax - by, \quad \frac{dy}{dt} = -ay + bx$$

with the kick occurring whenever $y = 0$ and $x < 0$. X receives a kick to the left with magnitude k. Thus when $y(t) = 0$ and $x(t) < 0$, then $x(t) = x(t) - k$. The ODE file is written as

```
# the clock model - a linearly decaying spiral that is
# kicked out
# by a fixed amount
x'=-a*x-b*y
y'=-a*y+b*x
# heres the kicker
global -1 y {x=x-k}
par a=.1,b=1,k=.5
init x=.5,y=0
# change some XPP parameters to make a nice 2D phase-plane
@ total=150,xlo=-1.5,ylo=-1,xhi=1.5,yhi=1,xp=x,yp=y
done
```

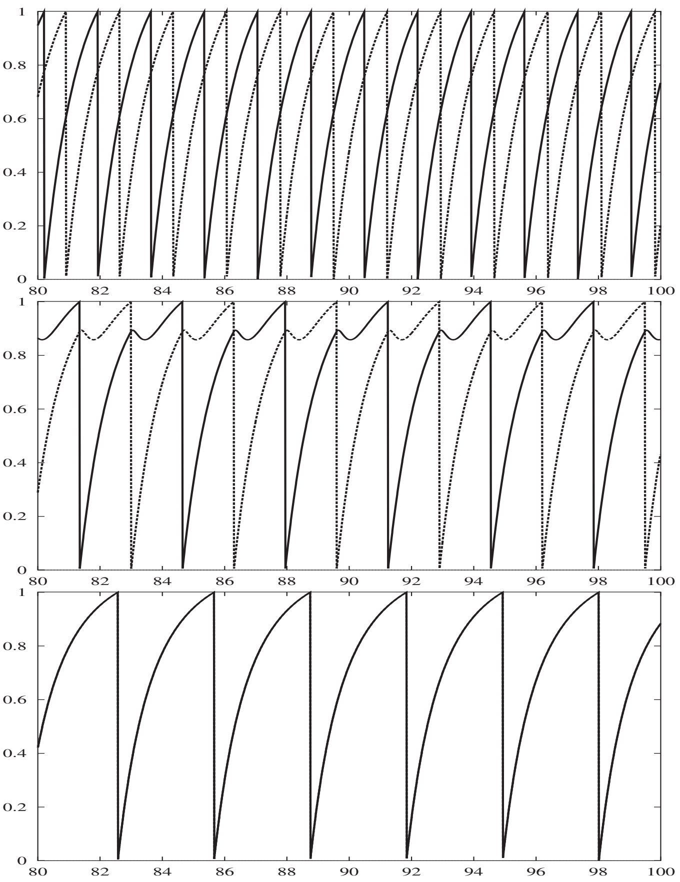

Figure 3.2. *A pair of integrate-and-fire neurons coupled with alpha-function synapses. With slow excitatory coupling, synchrony is unstable; with fast inhibition, synchrony is also unstable; with slow inhibition, synchrony is stable.*

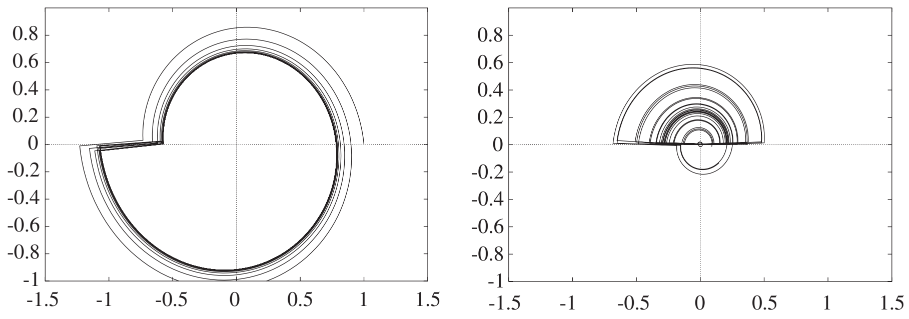

Figure 3.3. *A kicked clock and its chaotic brother.*

As usual, I have set some internal parameters so that you are looking at a phase-plane projection. Let's take a look at the `global` declaration. The linear equation is a spiral source and, since $b > 0$, the flow is counterclockwise. (Without the kick, this is a linear equation, but since the kick is dependent on the values of the variable, the ODE is nonlinear.) When $y = 0$ and $x < 0$ the variable y crosses 0 from positive to negative. Thus we want to use `-1` in the `global` statement: this means crossing from above to below. When this happens, x is kicked backwards by the amount k. Run this program. Change the initial conditions—note that no matter what you choose, the solutions always converge to a stable periodic solution. Let's add another window and plot x and y as a function of t. Click on `Makewindow Create` (**M C**). The main window will be repeated. Note the little rectangle in the upper left corner. This indicates that this view is the active view. All new plotting and changes in the graphics will occur in this window. If you want the main window to be active, simply click in it. For now, make the little window active and stretch it out a bit. In the **Initial Data Window**, click on the little boxes next to the `x,y` variables and then click on the button `xvst`. The result of this is that x and y will be plotted with x white and y red. Click on `Window/zoom Zoom in` (**W Z**) to zoom in on this and pick a few cycles so that you can see it better by clicking with the mouse on the little window and releasing the mouse after choosing a suitable area. If you mess up, click on `Window/zoom Fit` (**W F**) which will resize the window to contain all the data. The period of this clock is about 6.3.

A chaotic clock

We can make this little clock chaotic and in so doing introduce some additional features of *XPPAUT*. In fact, it is easy to prove that the resulting system is chaotic. Here is what to do. Change a from 0.1 to -0.1 and change the kick k from 0.5 to -0.5. Using the initial conditions $x = 0.5$ and $y = 0$, you should see a chaotic solution as in Figure 3.3. You can prove that this solution is chaotic (this is left as an exercise). The key point is that, since $a < 0$, the damping is negative so that nearby trajectories will exponentially diverge. The kick to the right keeps the solutions bounded. However, if you start with x large enough, then the solutions will grow without bound. You can also prove that there is an unstable periodic solution with initial conditions at about $(x, y) = (0.7831, 0)$. Any initial data inside this unstable periodic will converge to the chaotic attractor. Integrate using as initial conditions the values at the end of the last integration (click on `Initialconds Last` (**I L**)). Now let's look at the maximal Liapunov exponent, a measure of chaos. We haven't gotten much data, but can try it anyway. Click on `nUmerics stocHastic Liapunov` (**U H L**) and, after a brief moment, a window will appear, showing you an approximation of the maximal exponent. The approximation is not very good (the actual value is 0.1) but if you integrate longer, you will get a better value. (Try this: In the numerics menu, change `Total` to 1000. Change `nout` to 5. Integrate. Recompute the Liapunov exponent and it will be much closer to 0.1.) Another quantity that is indicative of chaos is the power spectrum. To get this, click on `nUmerics stocHastics Power` (**U H P**) and choose y as the variable to transform. In the **Data Viewer** the two columns formerly occupied by x and y are now filled with the power and the phase. (That is, the FFT returns the cosine and sine coefficients of the spectrum, (c, s); the power, $p = \sqrt{c^2 + s^2}$, is the magnitude; and the phase, $\phi = \arctan(s/c)$, is the angle.) The column occupied by t is replaced by the frequency. Click on `Escape` to leave the `nUmerics` menu. You can plot the spectrum by graphing `x` against `t`. Note the broadband frequency. There are solitary peaks at about 318, 636, and so on corresponding to the unstable periodic orbit. You can return to the original orbit by clicking on the `nUmerics Stochastic Data` (**U H D**) entry and then pressing `Esc` to return to the main menu.

There are a number of other interesting ways to analyze chaotic data. We will return to these at a later time.

3.5.3 The dripping faucet

One of the classic examples of chaos in a physical system is the dripping faucet. Bob Shaw [30] collected data on the interval between drips of the faucet. He found through a variety of analyses that the process was chaotic, and he simulated it on an analog computer. The model consists of a mass on a spring. The mass grows linearly and, when the spring extends beyond a certain limit, a fraction of the mass drops off. The fraction is proportional to the velocity. The equations are

$$\frac{d}{dt}\left(m\frac{dx}{dt}\right) = mg - \left(\mu\frac{dx}{dt} + kx\right), \qquad \frac{dm}{dt} = f,$$

with the condition that if $x(t)$ crosses 1, then the mass is decreased by an amount that depends on the velocity, $v = dx/dt$. Thus, $m(t) \to (1 - hv(t))m(t)$. The ODE file for this model is written

```
# faucet.ode
x'=v
v'=g-(k*x+(f+mu)*v)/m
m'=f
global 1 x-1 {m=max(m-h*m*v,m0)}
par f=.4,g=.32,h=4,m0=.01,mu=.6,k=1
init x=0,v=0,m=1
@ total=200
done
```

The first three equations are straightforward enough; we have just rewritten the second order system into a pair of first order systems and used the fact that $(mv)' = m'v + mv'$. The `global` declaration states that, when x crosses 1 from below, we decrement m by hmv. The additional function `max(x,m0)` ensures that the mass never falls below some minimum value or becomes negative. Run this and look in the `(x,v)` phase-plane to see something very similar to the Rossler attractor (see exercise 5 in the previous section). Play with the parameter f and see a variety of different solutions and bifurcations.

3.5.4 Exercises

1. Write a differential equation for John Tyson's cell growth model,

$$u' = k_4(v - u)(a + u^2) - k_6 u,$$
$$v' = k_1 m - k_6 u,$$
$$m' = bm,$$

   where $k_4 = 200$, $k_1 = 0.015$, $k_6 = 2$, $a = 0.0001$, and $b = 0.005$. m is the mass and, when the variable u drops **below** 0.2, the cell divides and $m = m/2$. Start with $u(0) = .0075$, $v = 0.48$, and $m = 1$. Integrate this for a total of 1000 with a time step of 0.25 and using the `DorPri(8)` integrator. Plot the mass as a function of time. Notice that, after a few transients, it goes to a periodic solution.

2. Consider the constant planar vector field

$$x' = 1, \qquad y' = a.$$

   If this is integrated on a torus, that is, each time $x = 1$ (respectively, $y = 1$), x (respectively, y) is reset to 0. If a is irrational, the trajectories densely fill the unit square representing the unfolded torus. For the first part of this exercise, verify this with a numerical simulation. Plot the phase-plane for the unit square and use the `Graphics Edit curve 0` to change the `Linetype` to 0 so that only dots are drawn. Use the `global` declaration to reset x and y.

   Curiously enough, if you solve this equation on a Klein bottle instead of a torus, all solutions are periodic! How do you make a Klein bottle phase-space? Recall that to create a Klein bottle, you must reverse the orientation along one of the edges of the square. Thus, when $y = 1$, reset y to 0 *and* set $x = 1 - x$, which flips its orientation. Make an *XPPAUT* file that does this and verify that all solutions are periodic. Prove your answer.

3. Put a lemon seed in a glass of soda. It will begin to sink, but as it sinks, bubbles accumulate on it making it more buoyant. It will slow down and float toward the surface. When it hits the surface, the bubbles pop and it begins to sink again. We can model this in a manner similar to the dripping faucet. As the seed sinks, it accumulates bubbles which have considerably less density than the soda. Eventually the volume fraction of the bubbles overcomes the seed density and the seed floats up.

The difference between the seed density and the density of the soda water provides the driving force. We assume that the soda is sufficiently viscous so that we can ignore inertial effects. Once the seed hits the surface, the bubbles pop, more so if the velocity is high. Let V_0 denote the volume of the seed with no bubbles and let $d > 1$ be the density with the density of the soda water equal to 1. Let $V(t)$ be the volume of bubbles accumulated and we will let their density be 0. The density of the seed is

$$d_s = d\frac{V_0}{V_0 + V(t)}.$$

The velocity of the seed is

$$x' = k(d_s - 1)Q(\epsilon - x).$$

The function Q prevents the seed from rising above the surface and thus it is kept below ϵ. We will use the step function for Q. The accumulation of bubbles is

$$V' = c,$$

where c is just a constant. Let $f(\dot{x})$ denote the fraction of bubbles that remains once the seed hits the surface. Then when $x = 0$, we replace $V(t)$ by $f(\dot{x})V(t)$. A linear f seems to be a good approximation:

$$f(\dot{x}) = \max(f_0 - f_1\dot{x}, 0).$$

Try writing an ODE file and simulating it. I made a nice seed oscillator with $\epsilon = 0.1$, $f_0 = 0.05$, $k = V_0 = 1$, $d = 2$, and $c = 0.1$.

4. **Project idea.** The drinking duck is a popular toy which consists of a glass tube filled with freon and decorated to look like a duck. When the duck's head is moistened, evaporative cooling causes the freon to rise up to the duck's head and this makes the duck swing down into a waiting glass of water. This action moistens the duck's head and also causes the freon to drop back to the duck's belly. He swings back and the process repeats. Model this as a pendulum with two weights. The head weight grows slowly proportional to the angular velocity. When the head drops, the weight is reset to the bottom and the process begins anew.