A quick tour of XPP

March 29, 2010

NOTES: Menu commands will appear like this `Command` and single letter keyboard shortcuts will appear like this: **A**. *Do not use the CapsLock key*; all shortcuts are lower case. Every command can be accessed by a series of keystrokes. To make sure key clicks are interpreted correctly, click on the title bar of the window for which the shortcut is intended.

1 Ordinary differential equations.

1.1 Creating the ODE file.

Consider the simple linear differential equation:

$$\begin{aligned} \frac{dx}{dt} &= ax + by \\ \frac{dy}{dt} &= cx + dy \end{aligned} \tag{1}$$

where a, b, c, d are parameters. We will explore the behavior of this two-dimensional system using *XPPAUT* (even though it is easy to obtain a closed form solution). To analyze a differential equation using *XPPAUT* , you must create an input file that tells the program the names of the variables, parameters, and the equations. By convention, these files have the file extension `ode` and I will call them ODE files. Here is an ODE file for (1):

```
# linear2d.ode
#
# right hand sides
x'=a*x+b*y
y'=c*x+d*y
#
# parameters
par a=0,b=1,c=-1,d=0
#
# some initial conditions
init x=1,y=0
#
```

```
# we are done
done
```

I have included some comments denoted by lines starting with #; these are not necessary but can make the file easier to understand. The rest of the file is fairly straightforward (I hope). The values given to the parameters are optional; by default they are set to zero. The `init` statement is also optional. The minimal file for this system has 4 lines:

```
x'=a*x+b*y
y'=c*x+d*y
par a,b,c,d
done
```

Of course all parameters are set to zero as are the initial conditions. Use a text editor to type in the first file exactly as it is shown. Name the file `linear2d.ode` and save it. That's it - you have written an ODE file. The minimal steps are:

- Use an editor to open up a text file.
- Write the differential equations in the file; one per line.
- Use the `par` statement to declare all the parameters in your system. Optionally define initial conditions with the `init` statement.
- End the file with the statement `done`
- Save and close the file.

NOTE The equation reader is case-insensitive so that `AbC` and `abC` are treated as identical.

WARNING In statements declaring initial conditions and parameters, **do not ever** put spaces between the variable and the "=" sign and the number. *XPPAUT* uses spaces as a delimiter. Always write `a=2.5` and **never** write `a = 2.5`.

1.2 Running the program

Run *XPPAUT* by typing

```
xpp linear2d.ode
```

Replace `xpp` with whatever you have decided to call the executable with all the desired command line options. (*If you are using winpp, click on the* **winpp** *icon; then choose the file from the file selection dialog box.*)

Six windows will appear on the screen or they may be iconified (depending on the command line options.) (If any of the windows appear "dead" or blank, iconify them manually and then uniconify them. Next time run *XPPAUT* without the `-iconify` command line option.

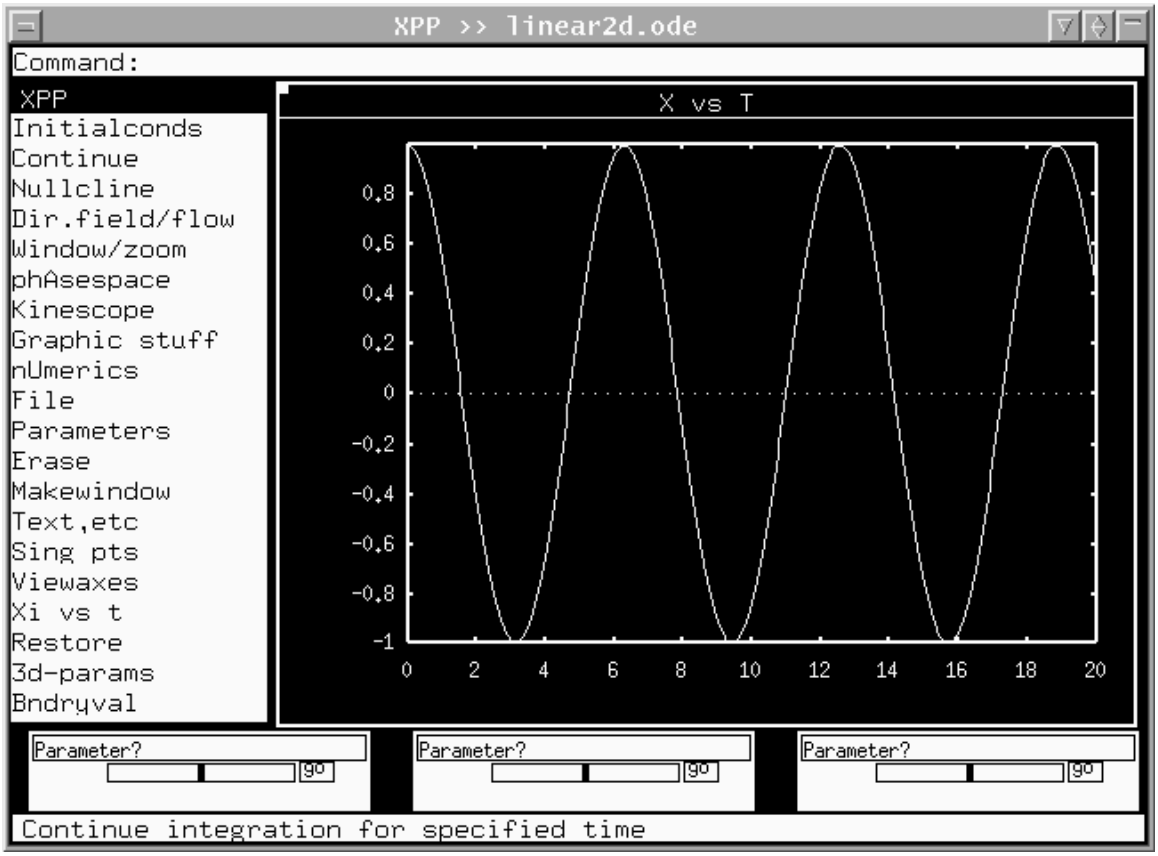

Figure 1: The main *XPPAUT* window

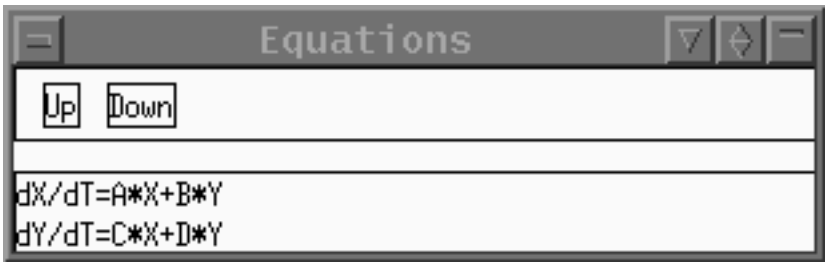

Figure 2: The Equation Window.

1.3 The main window.

The **Main Window** contains a large region for graphics, menus, and various other regions and buttons. It is illustrated in Figure 1. Commands are given either by clicking on the menu items in the left column with the mouse or tapping keyboard shortcuts. After a while, as you become more used to *XPPAUT* , you will use the keyboard shortcuts more and more. I will tell you the full commands and the keyboard shortcuts. In general, the keyboard shortcut is the first letter of the command unless there is ambiguity (such as `Nullcline` and `nUmerics`) and then, it is just the capitalized letter (**N** and **U** respectively). Unlike Windows keyboard shortcuts, the letter key alone is sufficient and it is not necessary to press the Alt key at the same time. The top region of the **Main Window** is for typed input such as parameter values. The bottom of the **Main Window** displays information about various things as well as a short description of the highlighted menu item. The three little boxes with the words `parameter` are sliders to let you change parameters and initial data.

In addition to the **Main Window** , there are several other windows that appear. The **Equation Window**, shown in figure 2 allows you to see the differential equations that you are solving. We will describe the other windows as the tutorial progresses.

1.3.1 Quitting the program.

To exit *XPPAUT* , click `File` `Quit` `Yes` (**F** **Q** **Y** .)

1.4 Solving the equation, graphing, and plotting.

Here, we will solve the ODE, use the mouse to select different initial conditions, save plots of various types, and create files for printing.

Computing the solution. In the **Main Window** , you should see a box with axes numbers. The title in the window should say `X vs T` which tells you that the variable `X` is along the vertical axis and `T` along the horizontal. The plotting range is from 0 to 20 along the horizontal and -1 to 1 along the vertical axis. When a solution is computed, this view will be shown. Click on `Init Conds`

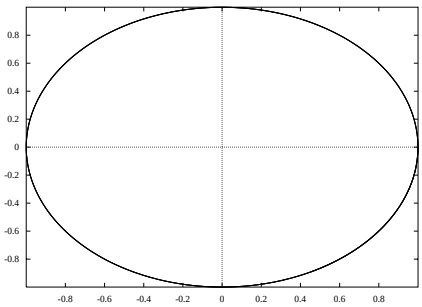

Figure 3: Phase plane for the linear 2d problem.

Go (I G) in the **Main Window**. A solution will be drawn followed by a beep. As one would expect given the differential equations, the solution looks like a few cycles of a cosine wave.

Changing the view. To plot Y versus T instead of X, just click on the command Xi vs t X) and choose Y by backspacing over X, typing in Y and typing **Enter**.

Many times, you may want to plot a phase-plane instead, that is X vs Y. To do this, click on Viewaxes 2D (V 2) and a dialog box will appear. Fill it in as follows:

X-axis: X	Xmax: 1
Y-axis: Y	Ymax: 1
Xmin: -1	Xlabel:
Ymin: -1	Ylabel:

Click on OK when you are done. (Note that you could have filled in the labels if you had wanted, but for now, there is no reason to.) You should see a nice elliptical orbit in the window. This is the solution in the phaseplane (cf Fig. 3).

Shortcuts. There is a very simple way to view the phaseplane or view variables versus time. Look at the **Initial Data Window** (Fig 4). You will see that there are little boxes next to the variable names. Check the two boxes next to X and Y. Then at the bottom of the **Initial Data Window**, click the XvsY button. This will plot a phaseplane and automatically fit the window to contain the entire trajectory. This is a shortcut and does not give you the control that the menu command does. (For example, the window is always fit to the trajectory, and no labels are added or changed. Nor can you plot auxiliary quantities with this shortcut.) To view one or more variables against time, just check the variables you want to plot (up to 10) and click on the XvsT button in the **Initial Data Window**.

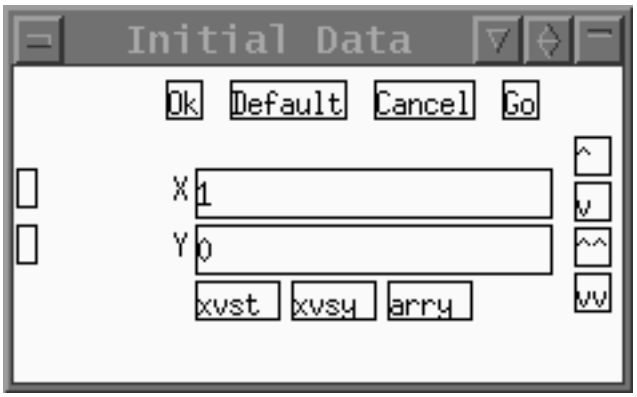

Figure 4: The initial conditions window.

You should have a phaseplane picture in the window. (If not, get one following the above instructions or using the shortcut.) Click on Init Conds Mouse (**I** **M**.) Use the mouse to click somewhere in the window. You should see a new trajectory drawn. This, too, is an ellipse. Repeat this again to draw another trajectory. If you get tired of repeating this, try Init Conds mIce (**I** **I**) which, being "mice" is many mouses. Keep clicking in the window. When you are bored with this, click either outside the window or tap the escape key, **Esc**.

Click on Erase and then Restore (**E** **R**). Note that all the trajectories are gone except the latest one. *XPPAUT* only stores the latest one. There is a way to store many of them, but we will not explore that for now.

Printing the picture.

XPPAUT does not directly send a picture to your printer. Rather, it creates a postscript file which you can send to your printer. If you don't have postscript capabilities, then you probably will have to use the alternate method of getting hardcopy. (Note that Word supports the import of PostScript and Encapsulated PostScript, but can only print such pictures to a postscript printer. You can download a rather large program for Windows called GhostView which enables you to view and print postscript on nonpostscript printers. Linux and other UNIX distributions usually have a PostScript viewer included.)

Here is how to make a PostScript file. Click on Graphics Postscript (**G** **P**) and you will be asked for three things: (i) Black and White or Color (ii) Landscape or Portrait; (iii) and the Fontsize for the axes. Accept all the defaults for now by just clicking Enter. Finally, you will be asked for a filename. The File Selector box is shown in the figure 5. You can: move up or down directory trees by clicking on the <>; choose files by clicking on them; scroll up or down by clicking on the up/down arrows on the left or using the arrow keys and the PageUp/PageDown keys on the keyboard; change the wild card; or type in a

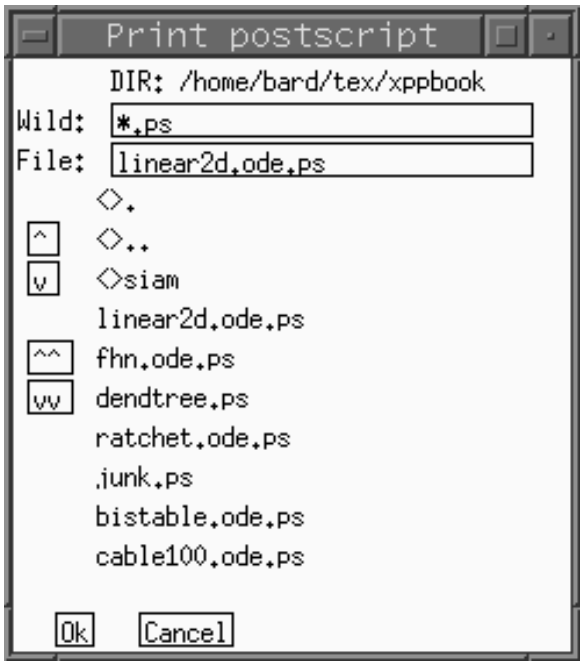

Figure 5: File selector.

filename. For now, you can just click on `Ok` and a postscript plot will be created and saved. The file will be called `linear2d.ode.ps` but you can call it anything you want.

Once you have the postscript file, you can type

```
lpr filename
```

on UNIX. In Windows, if your computer is hooked up to a postscript printer, then this usually does the trick:

```
copy filename lpt1:
```

Other ways to get hardcopy.

Another way to get hardcopy which you can import into documents is to grab the image from the screen. In Windows, click on `Alt+PrtSc` after making the desired window active. For some servers such as the MiX server, this will grab the entire X desktop window. Paste this into the MSPaint accessory and then use the tools in Paint to cut out what you want. Alternatively, you can download a number of programs that let you capture areas of the screen. In the UNIX environment, you can capture a window using `xv`, an excellent utility that is free and available for most UNIX versions. All of the screenshots in this tutorial were captured with `xv`. Finally, you can capture the screen (or a series of screen images) with the `Kinescope` | `Capture` command and then write these to disk

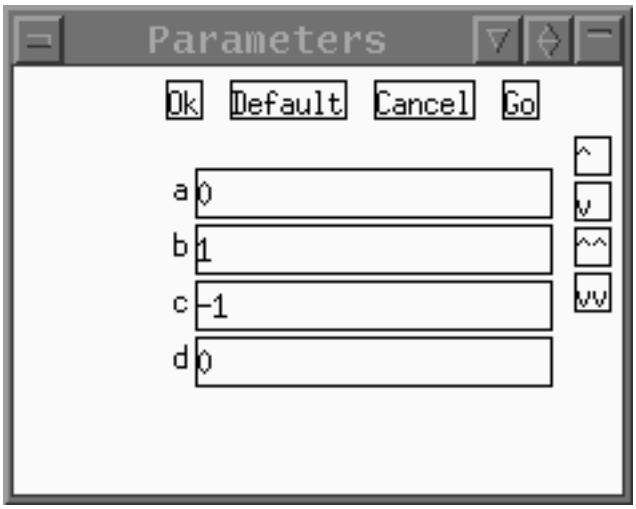

Figure 6: The parameter window.

with the Kinescope Save command. This produces a series of GIF files that are useable by many software packages.

Getting a good window. If you have computed a solution and don't have a clue about the bounds of the graph, let *XPPAUT* do all the work. Click on Window/zoom (F)it and the window will be resized to a perfect fit. The shortcut is **W** **F** and you will likely use it a lot!

2 Changing Parameters and Initial Data.

There are many ways to vary the parameters and initial conditions in *XPPAUT*. We have already seen how to change the initial data using the mouse. This method works for any n-dimensional system as long as the current view is a phaseplane of two variables. Here are two other ways to change the initial data:

- From the main menu click on Init Conds New and manually put them in at the prompts. You will be prompted for each variable in order. (For systems with hundreds of variables, this is not a very good way to change the data!)
- In the **Initial Data Window** , you can edit the particular variable you want to change. Just click in the window next to the variable and edit the value. Then click on the Go button in the **Initial Data Window**. If there are many variables, you can use the little scroll buttons on the right to go up and down a line or page at a time. If you click the mouse in the text entry region for a variable, you can use the **PageUp** etc keys to

move around. Clicking **Enter** rolls around in the displayed list of initial conditions. The Default button returns the initial data to those with with the program started. If you don't want to run the simulation, but have set the initial data, *you must* click on the Ok button in the **Initial Data Window** for the new initial data to be recognized.

There are many ways to change parameters as well. Here are three of them:

- From the **Main Window** , click on Parameters. In the command line of the **Main Window** , you will be prompted for a parameter name. Type in the name of a parameter that you want to change. Click on Enter to change the value and Enter again change another parameter. Click on Enter a few times to get rid of the prompt.
- In the **Parameter Window** (shown in figure 6), type in values next to the parameter you want to change. Use the scroll buttons or the keyboard to scroll around. As in the **Initial Data Window** , there are four buttons across the top. Click on Go to keep the values and run the simulation; click on Ok to keep the parameters withou running the simulation. Click on Cancel to return to the values since you last pressed Go or Ok. The Default button returns the parameters to the values when you started the program.
- Use the little sliders (Fig 7). We will attach the parameter d to one of the sliders. Click on one of the unused parameter sliders. Fill in the dialog box as follows:

Parameter: d
Value: 0
Low: -1
High: 1

and click Ok. You have assigned the parameter d to one of the sliders and allowed it to range between -1 and 1. Grab the little slider with the mouse and move it around. Watch how d changes. Now click on the tiny go button in the slider. The equations will be integrated. Move the slider some more and click on the go button to get another solution.

3 Looking at the numbers - the Data Viewer.

In addition to the graphs that *XPPAUT* produces, it also gives you access to the actual numerical values from the simulation. The **Data Viewer** shown in Figure 8 has many buttons, some of which we will use later in the book. The main use of this is to look at the actual numbers from a simulation. The

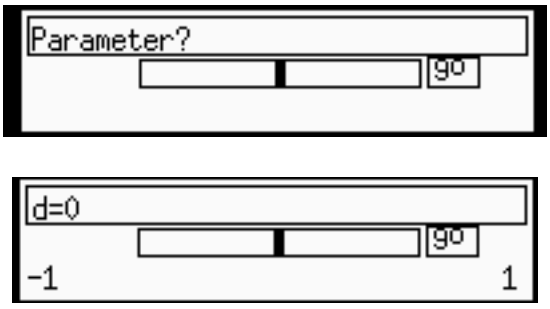

Figure 7: Top: Unused parameter slider. Bottom: Used parameter slider.

independent variable occupies the left-most column and the dependent variables filling in the remaining windows. Click on the top of the **Data Viewer**to make it the active window. The arrow keys and the **PageUp**, **PageDown**, **Home**, and **End** keys (as well as their corresponding buttons) do all the obvious things. Left and right keys scroll horizontally - a useful feature if you have many variables. I mention three buttons of use:

`Find` brings up a dialog box prompting you for the name of a column and a value. If you click on `Ok`, *XPPAUT* will find the entry that is closest and bring that row to the top. You can find the maximum and minimum, for example, of a variable.

`Get` loads the top line of the **Data Viewer** as initial data.

`Write` writes the entire contents of the browser to a text file that you specify.

4 Saving and restoring the state of *XPPAUT* .

Often you will have a view, a set of parameters, and initial data that you want to keep. You can save the current state of *XPPAUT* by clicking on `File` `Write set` (**F** **W**) in the **Main Window**. This bring up a file selection box. Type in a filename – the default extension is `.set`. The resulting file is an ASCII file that is human and computer readable. The first and last few lines look like:

```
## Set file for linear2d.ode on Fri Aug  4 13:53:31 2000
2   Number of equations and auxiliaries
4   Number of parameters
# Numerical stuff
1    nout
40    nullcline mesh

.......
```

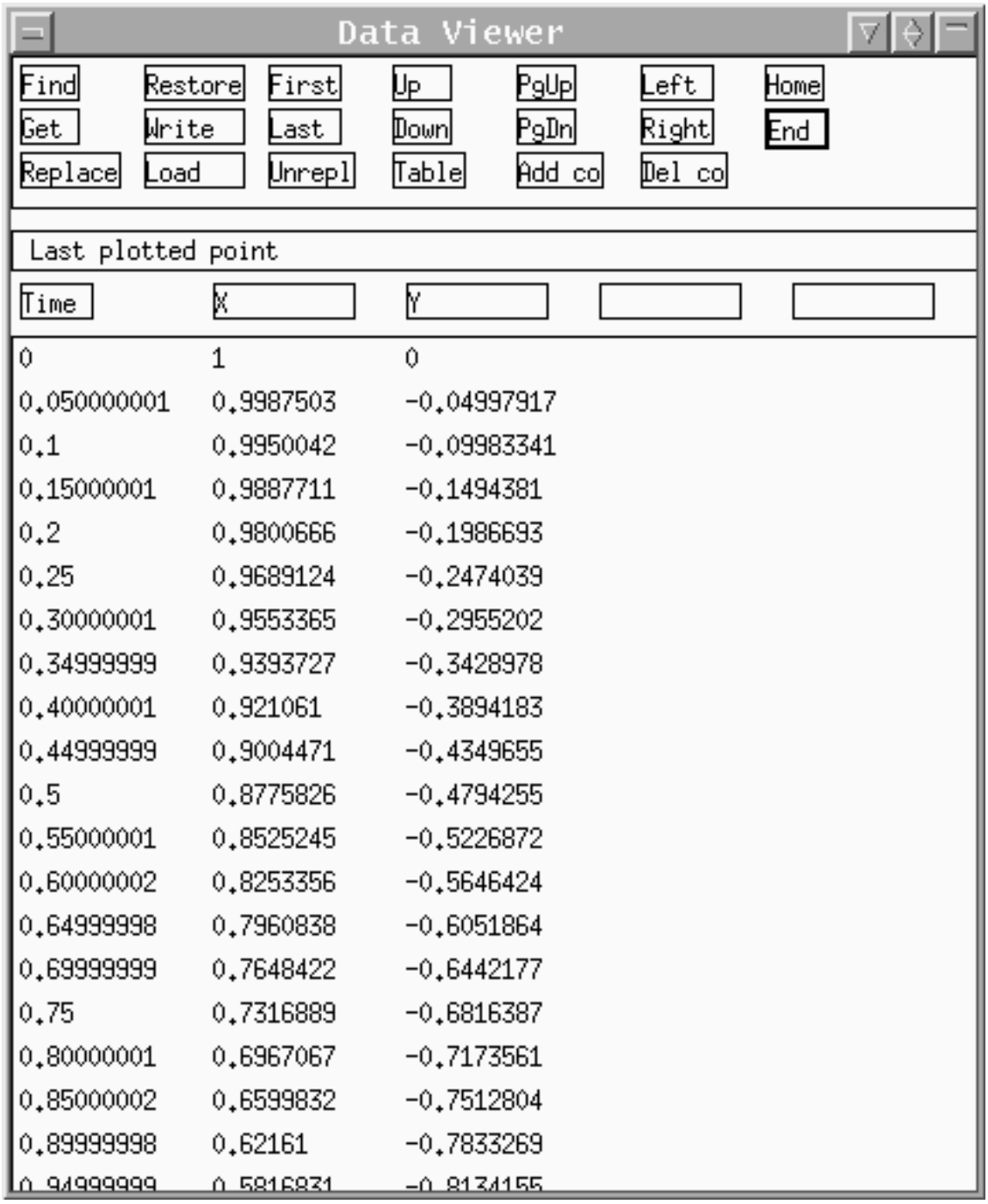

Figure 8: The Data Viewer.

```
RHS etc ...
dX/dT=A*X+B*Y
dY/dT=C*X+D*Y
```

Once you quit *XPPAUT* , you can start it up again and then use the File Read set to load up the parameters etc that you saved.

Now you should quit the program. We will look at a nonlinear equation next, find fixed points, and draw some nullclines and direction fields. To quit click on File Quit Yes (**F** **Q** **Y**).

4.0.1 Command summary

Initialconds Go computes a trajectory with the initial conditions specified in the **Initial Data Window** . (**I** **G**)

Initialconds Mouse computes a trajectory with the initial conditions specified by the mouse. Initialconds m(I)ce lets you specify many initial conditions. (**I** **M** or **I** **I**)

Erase erases the screen.(**E**)

Restore redraws the screen. (**R**)

Viewaxes 2D lets you define a new 2D view. (**V** **2**)

Graphic stuff Postscript allows you to create a postscript file of the current graphics. (**G** **P**)

Kinescope Capture allows you to capture the current view into memory and Kinescope Save writes this to disk.

Window/zoom (F)it fits the window to include the entire solution. (**W** **F**).

File Quit exits the program. (**F** **Q**)

File Write set saves the state of *XPPAUT* . (**F** **R**).

File Read set restores the state of *XPPAUT* from a saved `.set` file. (**F** **R**).

4.1 A nonlinear equation.

Here we want to solve a nonlinear equation. We will choose a planar system since there are many nice tools available for analyzing two-dimensional systems. A classic model is the Fitzhugh-Nagumo equation which is used as a model for nerve conduction. The equations are:

$$\begin{aligned} \frac{dV}{dt} &= I + V(1-V)(V-a) - w \\ \frac{dw}{dt} &= \epsilon(V - \gamma w) \end{aligned} \tag{2}$$

with parameters I, a, ϵ, γ. Typical values are $a = .1, I = 0, \epsilon = .1$, and $\gamma = 0.25$. Let's write an ODE file for this:

```
# Fitzhugh-Nagumo equations
v'=I+v*(1-v)*(v-a) -w
w'=eps*(v-gamma*w)
par I=0,a=.1,eps=.1,gamma=.25
@ xp=V,yp=w,xlo=-.25,xhi=1.25,ylo=-.5,yhi=1,total=100
@ maxstor=10000
done
```

We have already seen the first four lines: (i) lines beginning with a # are comments, (ii) the next two lines define the differential equations and (iii) the line beginning with `par` defines the parameters and their default values. The penultimate line beginning with the `@` sign is a directive to set some of the options in *XPPAUT* . These could all be done within the program, but this way everything is all set up for you. Details of these options are found in the user manual. For the curious, these options set the x-axis (`xp`) to be the `V` variable, the y-axis (`yp`) to be the `w` variable, the plot range to be [-.15,1.25]x[-.5,1], and the total amount of integration time to be 100. The last option `@ maxstor=10000` is a very useful one. *XPPAUT* allocates enough storage to keep 4000 time points. You can make it allocate as much as you want with this option. Here I have told *XPPAUT* to allocate storage for 10000 points. Type this in and save it as `fhn.ode`.

4.2 Direction fields.

Run this file by typing `xpp fhn.ode`. The usual windows will pop up. One of the standard ways to analyze differential equations in the plane is to sketch the *direction fields*. Suppose that the differential equation is:

$$x' = f(x,y) \qquad y' = g(x,y).$$

The phaseplane is divided into a grid and at each point in the grid, (x, y), a vector is drawn with (x, y) as the base and $(x + sf(x, y), y + sg(x, y))$ as the terminal point where s is a scaling factor. This so-called direction field gives

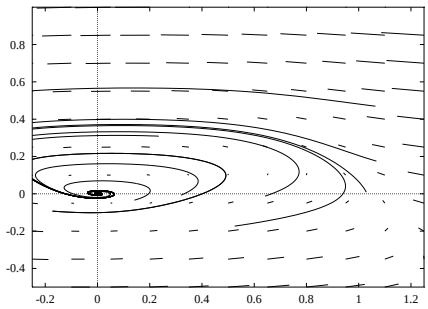

Figure 9: Direction fields and some trajectories for the Fitzhugh-Nagumo equations.

you a hint about how trajectories move around in the plane. *XPPAUT* lets you quickly draw the direction field of a system. Click on Dir.field/flow (D)irect Field (D D) and then accept the default of 10 for the grid size by clicking Enter. A bunch of vectors will be drawn on the screen, mainly horizontal. They are horizontal because ϵ is small so that there is little change in the w variable. The length of the vectors is proportional to the magnitude of the flow at each point. At the head of each vector is a little bead. If you want to have pure direction fields which don't take into account the magnitude of the vector field, just click on Dir.field (S)caled Dir. Fld (D S) and use the default grid size. (I prefer pure direction fields, but this is a matter of taste.)

Click on Initialconds m(I)ce and to experiment with a bunch of different trajectories. Note how the vectors from the direction field are tangent to the trajectories. See figure 9.

4.3 Nullclines and fixed points.

A powerful technique for the analysis of planar differential equations and related to the direction fields is the use of *nullclines*. Nullclines are curves in the plane along which the rate of change of one or the other variable is zero. The x-nullcline is the curve where $dx/dt = 0$, that is, $f(x, y) = 0$. Similarly the $y-$nullcline is the curve where $g(x, y) = 0$. The usefulness of these curves is that they break the plane up into regions along which the derivatives of each variable have a constant sign. Thus, the general direction of the flow is easy to determine. Furthermore, anyplace that they intersect represents a fixed point of the differential equation.

XPPAUT can compute the nullclines for planar systems. To do this, just click on Nullcline New (N N). You should see two curves appear; a red one representing the V-nullcline and a green one representing the W-nullcline. The green one is a straight line and the red is a cubic. They intersect just once:

there is a single fixed point. Move the mouse into the phaseplane area and hold it down as you move it. At the bottom of the **Main Window** you will see the x and y coordinates of the mouse. The intersection of the nullclines appears to be at (0,0). Figure 10 shows a printout along with a representative trajectory when the parameter $I = 0.3$.

The stability of fixed points is determined by linearizing about them and finding the eigenvalues of the resulting linear matrix. *XPPAUT* will do this for you quite easily. *XPPAUT* uses Newton's method to find the fixed points and then numerically linearizes about them to determine stability. To use Newton's method, a decent guess needs to be provided. For planar systems, this is easy to do – it is just the intersection of the nullclines. In *XPPAUT* fixed points and their stability are found using the `Sing pts` command as "singular points" is a term sometimes used for fixed points or equilibrium points. Click on `Sing pts` `Mouse` (**S** **M**) and move the mouse to near the intersction of the nullclines. Click the button and a message box will appear on the screen. Click on `No` since we don't need the eigenvalues. A new window will appear that contains information about the fixed points. The stability is shown at the top of the window. The nature of the eigenvalues follows:**c+** denotes the number of complex eigenvalues with positive real part; **c-** is the number of complex eigenvalues with negative real part; **im** is the number of purely imaginary eigenvalues; **r+** is the number of positive real eigenvalues; and **r-** is the number of negative real eigenvalues. Recall that a fixed point is linearly stable if all of the eigenvalues have negative real parts. Finally, the value of the fixed points is shown under the line. As can be seen from this example, there are two complex eigenvalues with negative real parts: the fixed point is (0,0). (*XPPAUT* reports a very small nonzero fixed point due to numerical error.) Integrate the system using the mouse, starting with initial conditions near the fixed point. (In the **Main Window** , tap **I** **I**.) Note how solutions spiral into the origin as is expected when there are complex eigenvalues with negative real parts.

For nonplanar systems of differential equations, you must provide a direct guess. Type your guess into the **Initial Data Window** and click on `Ok` in the **Initial Data Window** . Then from the **Main Window** , click on `Sing Pts` `Go` (**S**,`G`).

Change the parameter `I` from 0 to 0.4 in the **Parameter Window** and click on `Ok` in the **Parameter Window** . In the **Main Window** , erase the screen and redraw the nullclines: `Erase` `Nullclines` `New` (**E** **N** **N**). The fixed point has moved up. Check its stability using the mouse (`Sing pts` `Mouse`). The fixed point should be (0.1,0.4). Use the mouse to choose a bunch of initial conditions in the plane. All solutions go to a nice limit cycle. That is, they converge to a closed curve in the plane representing a stable periodic solution.

Let's make a nice picture that has the nullclines, the direction fields and a few representative trajectories. Since *XPPAUT* only keeps the last trajec-

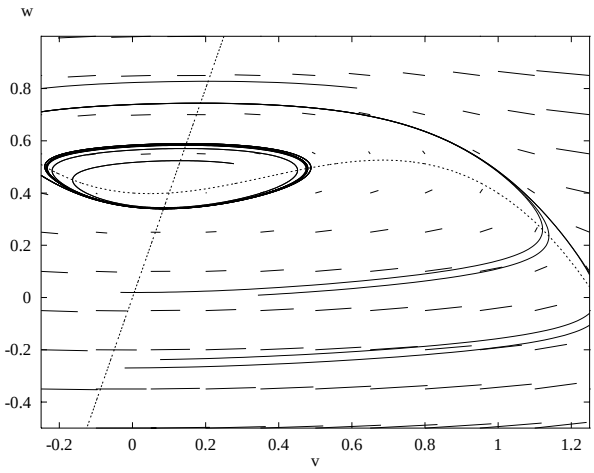

Figure 10: Nullclines,direction fields, trajectories for $I = 0.4$ in the Fitzhugh-Nagumo equations.

tory computed, we will "freeze" the solutions we compute. You can freeze trajectories automatically or one at a time. We will do the former. Click on `Graphic stuff` `(F)reeze` `(O)n freeze` (**G** **F** **O**) to permanently save computed curves. Up to 26 can be saved in any window. Now use the mouse to compute a bunch of trajectories. Draw the direction fields by clicking `Dir.field/flow` `(D)irect Field` (**D** **D**). Finally, lets label the axes. Click on `Viewaxes` `2D` (**V** **2**) and the 2D view dialog will come up. Change nothing but the labels, (the last two entries) and put `V` as the **Xlabel** and `w` as the **Ylabel**. Click on `Ok` to close the dialog. Finally, since the axes are confusing in the already busy picture, click on `Graphic stuff` `aXes opts` (**G** **X**) and in the dialog box change the 1's in the entries **X-org(1=on)** and **Y-org(1=on)** to 0's to turn off the plotting of the X and Y axes. Click `Ok` when you are done. Now create a postscript file, `Graphic stuff` `(P)ostscript` (**G** **P**) and accept all the defaults. Name the file whatever you want and click on **Ok** in the file selection box. Figure 10 shows the version that I have made. Your's will be slightly different. If you want to play around some more; turn off the automatic freeze option : `Graphic stuff` `Freeze` `Off freeze` (**G** **F** `O`); and delete all the frozen curves: `Graphic stuff` `Freeze` `Remove all` (**G** **F** **R**).

4.4 Command Summary.

`Nullcline` `New` draws nullclines for a planar system. (**N** **N**)

Dir.field/flow | (D)irect Field draws direction fields for a planar system. (D D)

Sing pts | Mouse computes fixed points for a system with initial guess specified by the mouse. (S M)

Sing pts | Go computes fixed points for a system with initial guess specified by the current initial conditions. (S G)

Graphic stuff | Freeze | On Freeze will permanently keep computed trajectories in the current window. (G F O)

Graphic stuff | Freeze | Off Freeze will toggle off the above option. (G F O)

Graphic stuff | Freeze | Remove all delete all the permanently stored curves. (G F R)

Graphic stuff | aXes opts lets you change the axes. (G X).

Viewaxes | 2D allows you to change the 2D view of the current graphics window and to label the axes. (V 2D).

4.5 The most important numerical parameters.

XPPAUT has many numerical routines built into it and thus there are many numerical parameters that you can set. These will be dealt with in subsequent sections of the book where necessary. However, the most common things you will want to change are the total amount of time to integrate and the step size for integration. You may also want to change the method of integration from the default fixed step Runge-Kutta algorithm. To alter the numerical parameters, click on nUmerics (U) which produces a new menu. This is a top level menu so you can change many things before going back to the main menu. To go back to the main menu, just click on the [Esc]-exit or tap Esc. There are many entries on the numerics menu. The following four are the most commonly used:

Total sets the total amount of time to integrate the equations. (Shortcut: T)

Dt sets the size of the timestep for the fixed step size integration methods and sets the output times for the adaptive integrators. (Shortcut: D)

Nout sets the number of steps to take before *plotting* an output point. Thus, to plot every fourth point, change `Nout` to 4. For the variable step size integrators, this should be set to 1.

Method sets the integration method. There are currently 13 available. (Shortcut: M). They are described in the user manual.

When you are done setting the numerical parameters, just click on Esc-exit or tap the Esc key.

5 Partial differential equations

XPPAUT doesn't have any way to solve PDEs other tahn by discretizing space and producing a series of ODEs using the method of lines. However, one does not have to write all the differential equations down, one at a time. There are ODE file shortcuts which make this easy to do. There is also a nice way of plotting the space-time behavior of a one-dimensional PDE. I will go through one quick example here. Consider the PDE:

$$\begin{aligned} \frac{\partial V}{\partial t} &= f(V,w) + D\frac{\partial^2 V}{\partial x^2} \\ \frac{\partial w}{\partial t} &= g(V,w) \end{aligned}$$

where f, g are the kinetics for the Fitzhugh-Nagumo model or some other model. For simplicity, I assume Neumann boundary conditions. This system can be discretized with the method of lines yielding a system of ODEs:

$$\begin{aligned} V_0' &= f(V_0,w_0) + d(V_1 - V_0) \\ V_j' &= f(V_j,w_j) + d(V_{j+1} - 2V_j + V_{j-1}) \quad j = 1,\ldots,N-1 \\ V_N' &= f(V_N,w_N) + d(V_{N-1} - V_N) \\ w_j' &= g(V_j,w_j) \quad j = 0,\ldots,N \end{aligned}$$

We will use the Fitzhugh-Nagumo kinetics and make an ODE file of the discretized system:

```
# fitzhugh-nagumo action potential
f(v,w)=v*(1-v)*(v-a)-w+i
g(v,w)=eps*(v-gam*w)
par a=.05,i=0,eps=.01,gam=.2
par d=.5
v0'=f(v0,w0)+d*(v1-v0)
v[1..49]'=f(v[j],w[j])+d*(v[j+1]-2*v[j]+v[j-1])
v50'=f(v50,w50)+d*(v49-v50)
w[0..50]'=g(v[j],w[j])
@ total=200,dt=.25,meth=qualrk,tol=1e-6
@ xhi=200,yp=v20
done
```

XPPAUT actually expands this to 100 differential equations and the variables are named v0,w0,v1,w1 and so on up to v50,w50. You must always use the

letter "j" for the index. We have told *XPPAUT* to use a quality stepsize (adaptive) Runge-Kutta routine with an outpu stepsize of 0.25. We integrate the equations for 200 time units. We plot `v20` so that the appearance of an action potential down line will be clear.

Run *XPPAUT* with this file. Now we will give some initial conditions. Rather than type them in one by one, we will define $V_j(0)$ by a formula. Click on `Initial conds` `formUla`. When prompted for the variable type in `v[0..50]` and type in `heav(5-[j])` for the formula. Then tap the Enter key a few times. Note that the index is referred to as `[j]` in the formula rather than just `j`. You should see an action potential appear on the screen. Click on `Graphic stuff` `Add curve` and choose `V40` for the y-axis and color 7 (green) for the color. The potentials of the 20th and 40th points will appear. In the **Initial Data Window** , click on the box next to `V0`. Scroll down and click on the box next to `V50`. Now click on the button labeled `arry`. A new window will appear with the space-time plot of the potential. You can fool around with the parameters for this plot by clicking on the edit box in the window.

As a last bit of analysis, we can look at the spatial profile at a fixed point in time. To do this, we will transpose all the space time data so that the 51 columns of the potential at a particular point in time become 51 rows in the second column; the first column will hold the indices. Click on `File` Transpose. Edit the dialog box so that `NCols=51` (the number of columns); `Row 1=300` (the output time step is 0.25 so row 300 represents t=75). Click `OK` to complete the transpose. (You can undo this by clicking on `File` `Transpose` and then clicking `Cancel` in the dialog box.) Once you have transposed the data, just plot `V0` versus time. This is the spatial profile at t=75.

6 Stochastic equations

XPPAUT has many features useful for stochastic modeling. In particular, it can simulate Brownian motion and continuous Markov processes. Before turning to an sample ratchet and a sodium channel simulation, I first create an *XPPAUT* file for exercise 12.03. This is a ratchet that moves between -1 and 1 and is not allowed to exit the boundaries. It is easiest to treat this in discrete time. `normal(0,1)` is a function which produces a normally distributed random number with mean 0 and standard deviation 1. Thus, the exercise can be written as the simple ODE:

```
par f1=-5,f2=5,h=.1,q=2
@ total=1000,meth=discrete
init x=-1
xp=x+h*.5*(f1*(sign(-x)+sign(x+1))+f2*(sign(x)+sign(1-x)))+sqrt(q*h)*normal(0,1)
x'=max(min(xp,1),-1)
done
```

The statement `meth=discrete` tells *XPPAUT* to threat this as a map rather

than a continuous differential equation. `xp` is the new value of `x` under the random dynamics. However we don't want x to escape the boundaries of ± 1 so when we update the new value of `x` it is constrained by the function `max(min(xp,1),-1)`.

In the next example, we simulate a sodium channel model due to Joe Patlak. The model undergoes the transitions shown in Figure **. The functions $\alpha_m, \beta_m, \alpha_h$ are the usual voltage dependendent functions for the Hodgkin-Huxley equations. *XPPAUT* can simulate a multi-state Markov process by defining a "Markov" variable (which has N states, $0, 1, \ldots, N-1$) and the transition matrix. Each row of the transition matrix is given on a single line following the declaration of the Markov variable. Each entry is contained within the curly brackets, { and }. For example suppose that you had a two state process with transition rates a from 0 to 1 and b from 1 to 0. Then you would write:

```
markov z 2
{0} {a}
{b} {0}
```

Note that you can put anything you want in the diagonals as they are ignored. Here is a complete ODE file for the above process:

```
# two state markov model
par a=.2,b=.3
markov z 2
{0} {a}
{b} {0}
@ total=50,xhi=50,xp=z,yp=z,yhi=1.5,ylo=-.5
# dum'=0
done
```

The last line should be uncommented if your version of *XPPAUT* does not accept this file. Older versions require at least one differential equation. Run this and integrate the equations. See z flip up and down.

Now with this trivial example in mind, we turn to the sodium channel model. Here is the *XPPAUT* file:

```
# model for the hh Na channel
# due to patlack
#
par vhold=-100,vnew=10
par ton=1,toff=16,ena=50
par k1=.24,k2=.4,k3=1.5
v=vhold+heav(t-ton)*heav(toff-t)*(vnew-vhold)
am=.1*(v+40)/(1-exp(-(v+40)/10))
bm=4*exp(-(v+65)/18)
ah=.07*exp(-(v+65)/20)
markov z 5
{0} {3*am} {0} {0} {0}
```

```
{bm} {0} {2*am} {0} {k1}
{0} {2*bm} {0} {am} {k2}
{0} {0} {3*bm} {0} {k3}
{0} {0} {0} {0} {ah}
aux cond=(z==3)
aux ina=(z==3)*(v-ena)/40
aux pot=v
@ meth=euler,dt=.01,total=16
@ yp=ina,ylo=-1.5,xlo=0,xhi=15,bound=100
done
```

The voltage is stepped from a value `vhold` to `vnew`. The Markov process has 5 states. Only state 3 is conducting. Thus, the auxiliary variable `cond=(z==3)` is zero if the channel is not conducting and 1 if it is. The current passed is given by the variable `ina`. Euler's method is used for this since that is usually the best method to use for any stochastic models. Integrate this a few times to see the channel open transiently. This is a transient channel so even at hight potentials it stays on only briefly. The Hodgkin-Huxley equations arise by assuming that there are many independent channels. Since they are assumed to be independent, we can simulate the effect of m channels by just integrating the equations m times and averaging the output. *XPPAUT* does this for you. Click on `nUmerics` `Stochastics` `Compute` to tell *XPPAUT* how many trials. Choose `z` as the variable to range over, `200 steps` with `Start=0` and `End=0`. Then click `Ok`. The equations will be integrated 200 times – this is equivalent to haveing 200 independent channels. Once *XPPAUT* is done this (you can keep track by looking at the bottom), then click on `stocHastics` `Mean` to load the data browser with the mean values of all its rows over the 200 trials. Click on `Escape` to get back to the main menu and then click on `Restore` to see the mean value. This look's very similar to the deterministic solution. Try simulating fewer channels (e.g, 10) and more channels, (e.g. 1000). Why do we have to take such small steps?

As a final example of a stochastic equation, we simulate the "flashing ratchet" model; an asymmetric ratchet which flashes on and off at a particular rate according to a Markov process and subject to simple delta-correlated noise. Here is the model:

$$dx = -zf(x)dt + \sigma d\xi$$

where z is a two state Markov process either off (0) or on (1) and f is just the derivative of an asymmetric potential. The potential should be made periodic. Here is the *XPPAUT* file

```
# ratchet
#  -1 for 0<x<1
#   1/a for 1<x<1+a
ff(x)=if(x<1)then(-1)else(1/a)
f(x)=ff(mod(x,1+a))
par a=.25
```

```
par alpha=.2,beta=.2
par sig=.5
wiener xi
x'=f(x)*z+sig*xi
markov z 2
{0} {alpha}
{beta} {0}
@ total=200,bound=200,meth=euler
done
```

I first define the potential on the interval $[0, 1 + a)$ and then extend it to the whole line modulo $1 + a$. I define a two-state markov process which flips randomly between 0 and 1. The `wiener w` declaration tells *XPPAUT* that this is a normally distrinuted number scaled by the internal time-step `dt`. Thus, if you change `dt`, the standard deviation is scaled accordingly by $\sqrt{dt}$. Integrate the equations for a few times. Next, do 50 simulations and look at the mean trajectory. (Use the `nUmerics` `stocHastic` `Compute` as was done in the channel model.) You will see that there is persistent downward drift. This is what is predicted by theory.