Computational Methods in Biology (Fall 2021)

Instructions for XPPAUT

The free software package **XPPAUT**, written by Bard Ermentrout at the University of Pittsburgh, is a great tool for models consiting of ordinary differential equations. Best of all, it's freeware that can be downloaded from his website, and it is available for any computer platform (even for iPad and iPhone). This short introduction will tell you some of the basic XPP commands using one example. Bard's web site has a tutorial and many files that you can download to learn more and get more practice.

The example problem is a model for positive autoregulation. The differential equation is

$$\frac{dx}{dt} = \beta \cdot Sx \cdot Autoreg - \alpha \cdot x$$

where x is a transcription factor converted into an active state by the signal Sx. The maximum production rate is β and the degradation/dilution rate is α. The positive feedback of x onto itself is through the function Autoreg.

Writing and Reading XPP ode files

The computer code for this model, **pos_auto.ode**, can be downloaded from my website (www.math.fsu.edu/∼bertram/course_software). If you look in this file you will see how easy it is to enter ordinary differential equations (ODEs) and related things like parameter values, functions, and initial conditions. The program starts with a title. This is just a comment, using the comment character %. Next comes a block of code setting the XPP parameters that tell XPP what to do. They say to use the 4th-order Runge-Kutta method to numerically solve the ODE with initial step size of *dt* and maximum step size of *dtmax*. The simulation should go over *total* time units. Parameters *maxstor* and *bounds* are there for technical reasons and you should never have to change them. Paramters *xp* and *yp* are just the names of the variables that should go on the x- and y-axis, respectively. The lower and upper bounds for the axes are *xlo*, *xhigh*, *ylo*, and *yhi*. The final parameter is the most important; it tells xpp whether or not to beep every time a simulation is complete. For the sake of sanity I set *bell*=0 (or *bell*=off in some versions of XPP). If you want to irritate your friends then set *bell*=1 and do a lot of simulations.

After the XPP parameters I have listed the variables/parameters and their meanings. These are just comments to help me, and anyone else, remember what these things are. Commenting computer software is considered good practice, but sadly many programmers don't do it (don't be one of them).

Next is a list of parameters and their values. You can change these values once you have started up XPP. The first is a logic parameter that tells whether to use a Boolean production function (Autoreg would be a Boolean function), or to use a Hill function. Parameters *ton* and *toff* set the time at which the signal Sx is on and off. The α and β parameters are given next, followed by parameters for the Autoreg function. The Kxx parameter is the threshold if a Boolean function is used, or the Hill coefficient if a Hill function is used. In the latter case, nx is the exponent of the Hill function.

The next block of code is the initial condition, or initial value of the variable x. The number of initial conditions must match the number of variables.

The next block sets up the function for the input signal Sx. It uses Heaviside functions, also called step functions. The function *heav(t-t1)* means that if $t - t1 > 0$ then $heav = 1$. Otherwise $heav = 0$. So *pulse(t1,t2) = heav(t-t1)-heav(t-t2)* is a function that equals 0 if $t < t1$, it equals 1 if $t1 < t < t2$, and it equals 0 if $t > t2$. It has the right structure to produce a pulse of input, and the next line of code $Sx =$ *pulse(ton,toff)* does just this. It calls the *pulse* function and sends in *ton* as the time for the beginning of the pulse and *toff* as the time for the end of the pulse.

The next block sets up the autoregulation function. The function *step_up(v,K)* is an "upward" Boolean function that goes from 0 to 1 when $v > K$. (Later we will also use "downward" Boolean functions that go from 1 to 0 when $v > K$.) After the Boolean is the Hill function definition. This is set up so that you can adjust both the Hill coefficient and the exponent by changing parameter values. Finally, the autoreg value is computed, based on whether you specified a Boolean or Hill production function, and on the current value of x and the parameter values Kxx and nx.

The next block of code gives the differential equation, where x' means derivative with respect to time. This is followed by definitions of some auxiliary variables. This is done so that you can plot things that are not variables (x is the only variable). So I made Sx and *autoreg* into auxiliary variables so that they can be plotted versus time.

Running XPP ode files

There are a number of ways to run XPP, depending on the computer platform. You may just click on the name of the XPP application and Open the file pos_auto.ode, or if using unix or linux you would type the command "xppaut pos_auto.ode" or (if the executable is in the Applications folder), "/Applications/xppaut pos_auto.ode". Notice the buttons on the top of the XPP screen. Clicking on these will open other windows that will allow you to do things like change the values of the system parameters or initial conditions.

Once XPP is running you should see a screen with coordinate axes. It should say "X vs T" at the top. To run a simulation (i.e., to solve the differential equation) click on Initialconds and Go. Once you get the hang of this you'll want to type instead

of click. In this case you would type "ig". You should see how x evolves in time. It started with initial value of 8, declined to a steady state of 4, and later declined to 0 following the removal of the signal at $t = 15$. To see what the signal looks like you can open another window by clicking Makewindow and Create or typing "mc". [Note that this new window becomes the active window, as indicated by the small white or black dot in the left corner of the new window.] To change what is plotted in this new window click on Viewaxes and 2D (or type "v2"). Put Sx on the y-axis. While you are at it, you can change Ymax to 2 in the Viewaxes window (since the signal never goes beyond 1). Click on each window and type "r" (Restore) to redraw the curve on the window. You should see on the new window that Sx starts at 1 and then goes to 0 at $t = 15$. You can view the autoregulation function by opening a third window. Follow the same steps as before, but now put *autoreg* on the y-axis.

Now let's see what happens when the initial condition is changed from 8 to 0.5. To do this click on ICs at the top of the screen. Change 8 to 0.5 and click OK. Note that XPP does a funny thing here. For it to accept the change you need to click OK, then press Esc (escape) on your keyboard, and then click OK again (maybe even twice). You will know that the change has been accepted when the little black vertical bar next to the changed number goes away. Don't ask me about this, it's strange. In the future I'll just say "click OK". Now re-run starting from the new initial condition by clicking on a window to make it active, clicking Erase (or typing "e"), and then typing "ig". You should see that x now increases to its steady state at 4, until the signal is turned off.

Now we'll change a parameter value. A fun one to change is nx, the exponent of the Hill function (the production function is a Hill function since $Boolean = 0$). By increasing it to $nx = 4$ we increase the slope of the function near Kxx, making the function more nonlinear (increasing the effect of positive autoregulation). To make the change, click on the Param button at the top of the screen and change nx from 1 to 4, followed by Ok. Run the simulation starting with initial condition $x = 0.5$. Now, instead of increasing, x decreases to 0. Try it again, but with initial condition of 1. Now x increases to a steady state of about 5. How do I know this value? If I click the middle button on my mouse then the coordinates show up in the bottom left corner of the main XPP window. I just locate the cursor over the curve and click. If you don't have a middle button then maybe it is time to consider investing in a mouse with one! You can try other initial conditions and you should find that x always approaches a steady state of either 0 or 5. This is clearly a bistable system. There must be one additional steady state that is unstable. Can you estimate its value?

Next, let's move to a Boolean production function. In the parameter window set $Boolean = 1$ (the value of nx is now irrelevant, since it only applies to the Hill

function). The positive feedback function is now even more nonlinear. Can you see a difference in the dynamics?

One feature that is sometimes helpful is the ability to freeze curves. This will keep them on the screen even after you have erased everything else. Let's see how this can be useful. First, go back to a Hill production function with $nx = 1$ (since this is the default you can just click Default in the parameter window, maybe two or three times until the black bar goes away). From an initial condition of 1 run a simulation. Then click Graphic Stuff, then Freeze, and then again Freeze (or type "gff"). This brings up a box that allows you to enter the color of your frozen curve. This is coded by integers, so just type an integer other than 0 and your curve will be frozen with some color, but you have to type "r" (or click Restore) to see it. Next set $nx = 2$ and re-run. This will trace out a new curve with a higher steady state. Freeze it with a different color. Do it again with $nx = 3$ and again freeze the new curve. Finally, make the production function Boolean. You should see that the change in steady state is now so tiny that you can't tell the difference any more. Apparently, the steady state quickly converges from 4 to 5 as the nx parameter is increased from 1 to higher integer values. How to get rid of frozen curves? Click Graphic stuff, then Freeze and then Delete. This allows you to remove one frozen curve at a time.

Printing out your work

There are two ways you can print out a solution curve. One is to click Graphic stuff and then Postscript. This opens a window that allows you to name the postscript file and put it in the directory of your choice. Another window comes up first, just click Ok here. On the second window that pops up click on the home icon and you will then see your directories.

This method produces output that you can print and show to your close friends. But anyone else will think it is ugly and wonder about your aesthetic values. The second option is to export the data itself, rather than a picture of the graph. This is much more flexible since it allows you to make a figure using any graphing program that you wish to use, so the figures will always look better than with the first option. To do this, click on the Data button at the top of the screen. This brings up the Data Viewer, which shows columns of numbers for the time, variable(s), and auxiliary variables. These columns of numbers are what get saved when you then click on Write. This brings up another box that allows you to save the data in the directory of your choice.

You can make some nice graphs with Matlab. I have put two M-files on my web site (under Course Software) that are templates for making figures with Matlab. The first one (**plot_single.m**) reads in two columns of data from the data file *data.dat*. You can modify the M-file to read in whichever columns you are interested in plotting. The second one (**plot_double.m**) sets up two panels, one on top of the other. You

can plot two columns of data in one panel and two others in the other. Also on my website are two Scilab "sci" function files. The first, (**plot_from_file.sci**) creates a single-panel plot, while (**plot_from_file_double.sci**) makes a double-panel plot. Unlike Matlab, Scilab is free and can be downloaded to work with any operating system.

Quitting XPP

This single command gets its own heading since it is so important (almost as important as Bell Off). To quit XPP elegantly click File Quit Yes, or simply type "fqy".

Negative Autoregulation

It will be helpful to you to modify the positive autoregulation computer program so that it produces negative autoregulation instead. Try this, calling the new program **neg_auto.ode**, and play around with it so that you get the hang of using XPP.