XPP Commands

Bard Ermentrout — Dec 2012

ODE File Format

```
# comment line - name of file, etc
#include filename
d<name>/dt=<formula>
<name>'=<formula>
<name>(t)=<formula>
volt <name>=<formula>
<name>(t+1)=<formula>
x[n1..n2]' = ...[j] [j-1] ... <--  Arrays
%[i1..i2]
u[j]'=...
v[j]'=...
%
markov <name> <nstates> <init>
  {t01} {t02} ... {t0k-1}
  {t10} ...
  ...
  {tk-1,0} ... {tk-1 k-1}

aux <name>=<formula>
!<name>=<formula> <--  parameters defined as formulae
<name>=<formula>
parameter <name1>=<value1>,<name2>=<value2>, ...
wiener <name1>, <name2>, ...
number <name1>=<value1>,<name2>=<value2>, ...
<name>(<x1>,<x2>,...,<xn>)=<formula>
table <name> <filename>
table <name> % <npts> <xlo> <xhi> <function(t)>
global sign {condition} {name1=form1;...}
init <name>=<value>,...
<name>(0)=<value> or <expr> <--  delay initial conditions
bdry <expression>
0= <expression>    <---  For DAEs
solv <name>=<expression> <------ For DAEs
special <name>=conv(type,npts,ncon,wgt,rootname)
       fconv(type,npts,ncon,wgt,rootname,root2,function)
       sparse(npts,ncon,wgt,index,rootname)
       fsparse(npts,ncon,wgt,index,rootname,root2,function)
               fftcon(type,npts,wgt,rootname)
               mmult(ncol,nrow,matrix,rootname)
               fmmult(ncol,nrow,matrix,root1,root2,f)
               findext(type,n,skip,root)
               gill(meth,rxn_list)
               delmmult(n,m,w,tau,root)
               delsparse(m,nc,w,index,tau,root)

export {x1,x2,....} {x1p,x2p,..}
# comments
@ <name>=<value>, ...
```

```
set <name> {x1=z1,x2=z2,...}
only <name1>,<name2>,...
options <filename>
" {z=3,b=3,...} Some nice text   <---  Active comments
done
```

Remarks

- ODEs are put in as `name'=expression` or `dname/dt=expression`. Note you can use this notation for maps as well or use `name(t+1)=expression`
- `name=expression` allows you to use `name` in many locations and to build up more comples expressions. The order in which they are written is the order in which they are evaluated, so don't use an expression using a name that hasn't yet been defined.
- Array expressions such as `x[1..10]'=...[j]...` are expanded as `x1'=...1..` etc. The `[j]` is expanded into a number. You can use some minimal arithmetic as well such as `[j+1],[j-1],[j*2]` etc.
- `x[0..99](0)=sin(2*pi*[j]/100)` is a way to initialize anarray ODE
- `markov <name> <n> <init>` sets up a continuous Markov process with `n` states and whos $n \times n$ transition matrix follows with entries delimited with the braces `{, }`. The diagonal entries should just be 0 as they are ignored. The starting state is given by `<init>`.
- `aux` quantities are extra stuff you might want to plot.
- `parameters` are named quantities that you can change within the program
- `numbers` are named quantities that are invisible to the user and cannot be changed
- `!name=...` defines a named quantity whose value depends on other numbers an parameters, but not variables, as these named quantities are only computed when you change parameters.
- `wiener` defines as set of Wiener processes
- `table` reads in function in the form of a table or you can define it within the ODE file; The file version of a table starts with three numbers, the number of values; the low; and the high; followed by the y-values. The x values are equally spaced from low to high. Tables are treated as functions of one variable with a linear interpolation as the default. If you want a cubic spline then the initial number in the file should have an "s" in front of it and if you want piecewise constant then put an "i" in front of the number. The function form of the table uses the `%` followed by three numbers as with the file format and then a formula using the independent variable `t`. So `table s % 51 -25 25 exp(-abs(t))/2` will produce a tabular function such that `s(5)` would return $\exp(-5)/2$. Other values are interpolated. Tables are very useful for networks.
- While rather clumsy in notation, you can initialize a delay equation via `x(0)=f(t)` to set the values of x for $-\tau \leq t < 0$. Delay initial data is zero by default.
- The `global` declaration includes two special quantities. Inside the braces, if you type `arret=value`, the integration will stop if the `value` is not zero. This way you can stop an integration if a particular event happens. The other declaration is `out_put=value` which will overide the `transient` (see the numerics menu) and allow you to plot (when `value` is nonzero), but only when the events occur.
- In the `global` the `sign` is {`0,1,-1`}. 0 means the condition must be met exactly at the point it was checked. It is a good way to set initial conditions as expressions, eg `global 0 t {x=sin(1)}`
- `bdry <expression>` is a way to set boundary conditions for the boundary value solver. The BVP solver tries to zero the expression. Use the names of your variables for the left end conditions and primed versions for the right ends. (see `gberg.ode`).
- The pair `0= <expression>` amd `solve <name>=<expression>` set up differential algebraic equations. The lines starting with `0=` will solve for the variables in the `solv` lines to make the expressions zero. See `huygens.ode` for an example where three accelerations are solved for to get the dynamics of two pendulums on a cart.
- `export` is used to communicate with a dynamically loaded library compiled from some C code. See `tstdll.ode`.
- `only` is useful for silent mode (no GUI) as this puts out a bunch of data in a file. `only` restricts the output to specified set of variables
- `set` creates a bunch of settings for numerics, variables, parameters that you can call by name within XPP.

- `Active comments` allow you to create a little tutorial where you take care of changing parameters, numerics, etc. You involke this in XPP with the File Printsrc command. See `lecar.ode` for an example.

INTEGRAL EQUATIONS

The general integral equation

$$u(t) = f(t) + \int_0^t K(t, s, u(s))ds$$

becomes

```
u = f(t) + int{K(t,t',u)}
```

The convolution equation:

$$v(t) = \exp(-t) + \int_0^t e^{-(t-s)^2} v(s)ds$$

would be written as:

```
v(t) = exp(-t) + int{exp(-t^2)#v}
```

If one wants to solve, say,

$$u(t) = exp(-t) + \int_0^t (t - t')^{-mu} K(t, t', u(t'))dt'$$

the form is:

```
u(t)= exp(-t) + int[mu]{K(t,t',u}
```

and for convolutions, use the form:

```
u(t)= exp(-t) + int[mu]{w(t)#u}
```

NETWORKS

```
special zip=conv(type,npts,ncon,wgt,root)
```

where `root` is the name of a variable and `wgt` is a table, produces an array `zip` with `npts`:

$$\text{zip}[i] = \sum_{j=-\text{ncon}}^{\text{ncon}} \text{wgt}[j + ncon]\text{root}[i + j]$$

```
special bob=fftconv(type,npts,wgt,root)
```

is similar to the `conv` operation, but uses the FFT to do it. The `type` should be `odd` or `periodic`. The size of the `wgt` table should be either `npts` or `2 npts`. The `sparse` network has the syntax:

```
special zip=sparse(npts,ncon,wgt,index,root)
```

where `wgt` and `index` are tables with at least `npts * ncon` entries. The array `index` returns the indices of the offsets to with which to connect and the array `wgt` is the coupling strength. The return is

```
zip[i] = sum(j=0;j<ncon) w[i*ncon+j]*root[k]
k = index[i*ncon+j]
```

The other two types of networks allow more complicated interactions:

```
special zip=fconv(type,npts,ncon,wgt,root1,root2,f)
```

evaluates as

```
zip[i]=sum(j=-ncon;j=ncon) wgt[ncon+j]*f(root1[i+j],root2[i])
```

and

```
special zip=fsparse(npts,ncon,wgt,index,root1,root2,f)
```

evaluates as

```
zip[i]=sum(j=0;j<ncon) wgt[ncon*i+j]*f(root1[k],root2[i])
k = index[i*ncon+j]
```

Matrix multiplication is also possible:

```
special k=mmult(n,m,w,u)
```

returns a vector `k` of length `m` defined as

```
k(j)=sum(i=0;i<n)w(i+nj)u(i)
```

while

```
special k=fmmult(n,m,w,u,v,f)
```

returns

```
k(j)=sum(i=0;i<n)w(i+nj)f(u(i),v(j))
```

```
special z=findext(type,n,skip,root)
```

finds the extreme values of a list of `n` variables starting at `root` and skipping every `skip` one. `type=1,-1,0` according as to whether you want the max, min, or both. In any case, `z(0), z(2)` are the max and min and `z(1),z(3)` are the index of the max and min. See `kohonen.ode` for an example of this.

```
special k=delmmult(n,m,w,tau,u0)
```

returns the m values

$$k(i) = \sum_{j=0}^{m-1} \mathrm{w}[im+j]u[j](t - \mathrm{tau}[im+j])$$

```
special k=delsparse(m,n,w,l,tau,root)
```

returns the m values

$$k(i) = \sum_{j=0}^{n-1} \mathrm{w}[in+j]u[l(in+j)](t - \mathrm{tau}[im+j])$$

```
special ydot=import(soname,sofun,nret,root,w1,w2,...wm)
```

This is a conveneient way to load a large number of right-hand sides that have been coded in C. See the examples in the cuda folder.

OPTIONS The format for changing the options is:

```
@ name1=value1, name2=value2, ...
```

where `name` is one of the following and `value` is either an integer, floating point, or string. (All names can be upper or lower case).

- QUIET=`0,1` dont print out stuff
- LOGFILE=`filename` store prontouts in logfile
- MAXSTOR=`integer` sets the total number of time steps that will be kept in memory. The default is 5000. If you want to perform very long integrations change this to some large number.

- FORECOLOR=`rrggbb` sets the hexidecimal frame color, and text menu color.
- BACKCOLOR=`rrggbb` sets the background color on the menus, sliders, dialogs, and buttons
- MWCOLOR=`rrggbb` sets the background color for the main frame in the popups.
- DWCOLOR=`rrggbb` sets the color of the drawing window and the browser numerical displays
- BACKIMAGE=`file.xbm` sets the back image. The format is the not generally readily found X11-bitmap format. On the mac, you can use some unix tools like gimp, xv, or ImageMagick.
- SMALLFONT=`fontname` where `fontname` is some font available to your X-server. This sets the "small" font which is used in the Data Browser and in some other windows.
- BIGFONT=`fontname` sets the font for all the menus and popups.
- SMC={0,...,10} sets the stable manifold color. These colors correspond to the colors available when plotting in XPP and are roughly, `''black'',red,redorange,orange,yelloworange,yello,yellowgreen,green,bluegr` Here "black" means the FORECOLOR.
- UMC={0,...,10} sets the unstable manifold color
- XNC={0,...,10} sets the X-nullcline color
- YNC={0,...,10} sets the Y-nullcline color
- SEC,UEC,SPC,UPC=color set the colors for the screen on AUTO for stable eq, unstable eq, stable per, unstable per.
- OUTPUT=filename sets the filename to which you want to write for "silent" integration. The default is "output.dat".
- GRADS={0,1} turn off or on gradients in the XPP buttons ¡/LI¿
- HEIGHT=pixels,WIDTH=pixels, sets the height and width of the main window intially
- RUNNOW=1 run the simulation as soon as the windows are up (Don't wait for Init conds Go, e.g.
- BUT=name:kbs defines button on the top of the main window. You can define up to 20 such buttons. They will appear across the top when you press them, they will execute an XPP keyboard shortcut. For example, **BUT=Fit:wf** will create a button labeled **Fit** and when you press it, it will be as if you had clicked **Window/zoom Fit**. Most of the menu items are available.

The remaining options can be set from within the program. They are

- LT=`int` sets the linetype. It should be less than 2 and greater than -6.
- SEED=`int` sets the random number generator seed.
- XP=name sets the name of the variable to plot on the x-axis. The default is `T`, the time-variable.
- YP=name sets the name of the variable on the y-axis.
- ZP=name sets the name of the variable on the z-axis (if the plot is 3D.)
- COLORMAP=`0,1,2,3,4,5` sets the colormap
- NPLOT=`int` tells XPP how many plots will be in the opening screen.
- XP2=name,YP2=name,ZP2=name tells XPP the variables on the axes of the second curve; XP8 etc are for the 8th plot. Up to 8 total plots can be specified on opening. They will be given different colors.
- MULTIWIN=`0,1`, puts multiple windows up and one each of the NPLOT curves in them.
- SIMPLOT=`0,1` turns on the simultaneous plot flag for multiple windows
- XHI2=value,YHI2=value,XLO2=value,YLO2=values dimensions the extra windows up to 8.
- AXES={`2,3`} determine whether a 2D or 3D plot will be displayed.
- COLORIZE=1,COLORVIA=`name`, COLORLO=`value`, COLORHI=`value` all set the colorization of trajectories and in the `Numerics colorcode` command. If `name` is `speed`, then colorcoding will be via velocity.
- TOTAL=value sets the total amount of time to integrate the equations (default is 20).
- DT=value sets the time step for the integrator (default is 0.05).
- NJMP=`integer` tells XPP how frequently to output the solution to the ODE. The default is 1, which means at each integration step.
- T0=value sets the starting time (default is 0).
- TRANS=value tells XPP to integrate until `T=TRANS` and then start plotting solutions (default is 0.)
- NMESH=`integer` sets the mesh size for computing nullclines (default is 40).
- DFGRID=`integer` sets the grid size for direction fields, flows etc (default is 10);

- METH={ `discrete,euler,modeuler,rungekutta,adams,gear,volterra, backeul, qualrk,stiff,cvode,5` sets the integration method (default is Runge-Kutta.)
- DTMIN=value sets the minimum allowable timestep for the Gear integrator.
- DTMAX=value sets the maximum allowable timestep for the Gear integrator
- VMAXPTS=value sets the number of points maintained in for the Volterra integral solver. The default is 4000.
- { JAC_EPS=value, NEWT_TOL=value, NEWT_ITER=value} set parameters for the root finders.
- ATOLER=value sets the absolute tolerance for CVODE.
- TOLER=value sets the error tolerance for the Gear, adaptive RK, and stiff integrators. It is the relative tolerance for CVODE.
- BOUND=value sets the maximum bound any plotted variable can reach in magnitude. If any plottable quantity exceeds this, the integrator will halt with a warning. The program will not stop however (default is 100.)
- DELAY=value sets the maximum delay allowed in the integration (default is 0.)
- BANDUP=`int`, BANDLO=`int` bandwidths for the Jacobian computation.
- PHI=value,THETA=value set the angles for the three-dimensional plots.
- XLO=value,YLO=value,XHI=value,YHI=value set the limits for two-dimensional plots (defaults are 0,-2,20,2 respectively.) Note that for three-dimensional plots, the plot is scaled to a cube with vertices that are ±1 and this cube is rotated and projected onto the plane so setting these to ±2 works well for 3D plots.
- XMAX=value, XMIN=value, YMAX=value, YMIN=value, ZMAX=value, ZMIN=value set the scaling for three-d plots.
- POIMAP={ `section,maxmin, period`} sets up a Poincare map for either sections of a variable, the extrema, or period.
- POIVAR=name sets the variable name whose section you are interested in finding.
- POIPLN=value is the value of the section; it is a floating point.
- POISGN={ `1, -1, 0` } determines the direction of the section.
- POISTOP=1 means to stop the integration when the section is reached.
- RANGE=1 means that you want to run a range integration (in batch mode).
- RANGEOVER=name, RANGESTEP=number, RANGELOW=number, RANGEHIGH=number, RANGERESET=`Yes,No`, RANGEOLDIC=`Yes,No` all correspond to the entries in the range integration option.
- TOR_PER=value, defined the period for a toroidal phasespace and tellx XPP that there will be some variables on the circle.
- FOLD=name, tells XPP that the variable ¡name¿ is to be considered modulo the period. You can repeat this for many variables.
- PS_Font=`fontname`,PS_LW=`linewidth`,PS_FSIZE=`fontsize`,PS_COLOR=`0,1` sets up postscript options.
- S1=`name`, SLO1=`number`,SHI1=`number` sets the variables, parameters associated with a slider and their low and high values. Use S2,S3 for the other sliders (This is different for the iPad/iPhone.)
- STOCH=`1,2` set the stochastic flag. Set up a range and use this to compute the mean etc of an ensemble of trajectories for batch mode integration. 1 returns the mean and 2 the variance.
- POSTPROCESS=`1,2,3,4,5,6` is for batch integration and allows you to process your data as follows: 1-histogram, 2-Fourier,3-Power,4-Power spectral density, 5-cross spectrum, 6-coherence.
- HISTHI=`number`,HISTLO=`number`,HISTBINS=`number`, HISTCOL=`variable name`, sets up the relevant quantitues for a batch histogram; used in conjunction with POSTPROCESS.
- SPECCOL=`name`, SPECCOL2=`name`, SPECWIDTH=`number`, SPECWIN=`0,1,2,3,4` (corresponding to square,parabolic,hamming,bartlett, or hanning windows), sets up relevant spectral stuff for use in conjunction with postprocessing.
- AUTO-stuff. The following AUTO-specific variables can also be set: `NTST, NMAX, NPR, DSMIN, DSMAX, DS, PARMIN, PARMAX, NORMMIN, NORMMAX, AUTOXMIN, AUTOXMAX, AUTOYMIN, AUTOYMAX, AUTOVAR`. The last is the variable to plot on the y-axis. The x-axis variable is always the first parameter in the ODE file unless you change it within AUTO.
- DLL_LIB=`file` Dynamically linked library
- DLL_FUN=`name` Dynamically linked function.

- DFDRAW =1, 2, or 3 will force the drawing of direction fields on batch plots; 1 is unscaled, 2 is scaled, and 3 is colorized.
- NCDRAW=1 forces the drawing of nullclines in batch plots

COLOR MEANING 0-Foreground color; 1-Red; 2-Red Orange; 3-Orange; 4-Yellow Orange; 5-Yellow; 6-Yellow Green; 7-Green; 8-Blue Green; 9-Blue; 10-Purple.

KEYWORDS You should be aware of the following keywords that should not be used in your ODE files for anything other than their meaning here.

```
sin cos tan atan atan2 sinh cosh tanh
exp delay ln log log10 t pi if then else
asin acos heav sign mod flr ran abs del\_shft
max min normal besselj bessely besseli erf erfc poisson
lgamma arg1 ... arg9  @ $ + - / * ^ ** shift
| > < == >= <= != not \# int sum of i'
```

These are mainly self-explanatory. The nonobvious ones are:

- heav(arg1) the step function, zero if arg1<0 and 1 otherwise.
- sign(arg) which is the sign of the argument (zero has sign 0)
- ran(arg) produces a uniformly distributed random number between 0 and arg.
- besselj, bessely, besseli take two arguments, n, x and return $J_n(x)$ $Y_n(x)$, $I_n(x)$ the Bessel functions.
- erf(x), erfc(x) are the error function and the complementary function.
- lgamma(x) is the log of the gamma function.
- normal(arg1,arg2) produces a normally distributed random number with mean arg1 and variance arg2.
- poisson(arg) produces the number of events from a Poisson process with parameter arg. It is a random number.
- max(arg1,arg2) produces the maximum of the two arguments and min is the minimum of them.
- if(<exp1>)then(<exp2>)else(<exp3>) evaluates <exp1> If it is nonzero it evaluates to <exp2> otherwise it is <exp3>. E.g. if(x>1)then(ln(x))else(x-1) will lead to ln(2) if x=2 and -1 if x=0.
- delay(<var>,<exp>) returns variable <var> delayed by the result of evaluating <exp>. In order to use the delay you must inform the program of the maximal possible delay so it can allocate storage.
- del_shft(<var>,<shft>,<delay>). This operator combines the delay and the shift operators and returns the value of the variable <var> shifted by <shft> at the delayed time given by <delay>. (See sine-circle.ode for an example.)
- mod(arg1,arg2) is arg1 modulo arg2
- flr(arg) is the integer part of<arg> returning the largest integer less than <arg>.
- t is the current time in the integration of the differential equation.
- pi is π.
- arg1, ..., arg9 are the formal arguments for functions. You never will actually see them, but they are used internally in the parser.
- int, # concern Volterra equations.
- shift(<var>,<exp>) This operator evaluates the expression <exp> converts it to an integer and then uses this to indirectly address a variable whose address is that of <var> plus the integer value of the expression. This is a way to imitate arrays in XPP. For example if you defined the sequence of 5 variables, u0,u1,u2,u3,u4 one right after another, then shift(u0,2) would return the value of u2.
- set(<var>,int,value) is a weird little function that will assign the variable <var> shifted by int the value value. It would be used for example to reset some random variable in a global statement. Since *all* XPP functions return values, this needs an assignment and will retun value. For example:

  ```
  global 1 t-t0 {u=set(x0,50*ran(1),3.14159}
  ```

will pick a random index `int` and assign `shift(x0,int)` the value 3.14159. Weird, but I needed it for a problem, so there it is.

- `sum(<ex1>,<ex2>)of(<ex3>)` is a way of summing up things. The expressions `<ex1>,<ex1>` are evaluated and their integer parts are used as the lower and upper limits of the sum. The index of the sum is `i'` so that you cannot have double sums since there is only one index. `<ex3>` is the expression to be summed and will generally involve `i'`. For example `sum(1,10)of(i')` will be evaluated to 55. Another example combines the sum with the shift operator. `sum(0,4)of(shift(u0,i'))` will sum up `u0` and the next four variables that were defined after it.

Add a `.xpprc` file to set your favorite options, e.g

```
@ bell=0,grads=0,dwcolor=eeddff
@ bigfont=lucidasanstypewriter-bold-14
```

Command line arguments

There are many command line arguments for Xpp. While some options affect the appearance of the GUI, other options provide an API for Xpp. Using the API, other programs or scripts can interact with Xpp in a batch mode. This can be useful for processing many files or runs.

The command line arguments are listed below in the order in which they were written. There are several that are pretty much useless but I have kept them for posterity.

-xorfix This changes the way rubber-band drawing for zooms and other things is done. If you do not see a box when you zoom in, you should run XPP with this argument.

-convert This allows you to convert old style parser format to the new style which is much more readable. The program creates a file with the same name as the input file but with the extension `.new` appended. It works on on the examples I have tried but it is still in beta testing.

-silent This allows you to run XPP's integrators without using the X-windows stuff. The result of the integration is saved to a file called "output.dat" but this can be changed. The length of integration, methods, Poincare sections, etc, are all specified in either the options file (see section **??**) or in the internal options. Note that when you run a range integration in silent mode, if the parameter `RANGERESET` is `yes` (the default) then a new output file will be opened for each integration. Thus, if you range over 50 values, you will get 50 output files named e.g. `output.dat.0`, `output.dat.1`, etc. If you have set `RANGERESET=no`, then only one file is produced.

-allwin tells XPP to make the parameter window, browser, etc immediately visible.

-setfile **filename** loads the setfile, **setfile** after loading up the ODE file.

-newseed uses the machine time to re-seed the random number generator.

-ee makes the TAB and RETURN in dialogs act like those in Windoze. By default, they are reversed.

-white This swaps foreground and background colors.

-runnow This runs ode file immediately upon startup (implied by -silent)

-bigfont *font* This uses the big font whose name is given. Note: On typical X Window installations the command xlsfonts lists available fonts. For example, the following command lists only the available fixed width fonts:

```
xlsfonts |  grep -i -e "typewriter" \
  -e "mono" -e "[0-9]x[0-9]" \
  -e "fixed" -e "-c-" -e "-m-" |  sort
```

Since X fonts often have long unwieldy names wildcards may be used. For example, the font with the name

`-b&h-lucidatypewriter-medium-r-normal-sans-24-240-75-75-m-140-iso10646-1`

may be specified more simply by using wildcards

`*lucidatypewriter*sans*24*`

-smallfont *font* This uses the small font whose name is given.

-parfile *filename* This loads parameters from the named file. The format for a `.par` file is illustrated below for `lecar.ode` (found in the examples `/ode` folder). The first line must give the number of parameters specified in the file followed by `Number params`. Each parameter value is given on a separate line and followed by its name which must be the same as a parameter name in the `.ode` file.

An Example `lecar.par`:

```
12 Number params
0.0     iapp
.333    phi
-.01    v1
0.15    v2
0.1     v3
0.145   v4
1.33    gca
-.7     vk
-.5     vl
2.0     gk
.5      gl
1       om
```

-outfile *filename* This sends output to this file (default is output.dat). The format for an `.out` file is illustrated below for `lecar.ode` (found in the examples `/ode` folder). Note that the first column is reserved for time values while the remaining columns correspond to the ordered variables defined in the `.ode` file.

An Example `lecar.out`:

```
0             -0.36059999   0.0911
0.050000001   -0.36620989   0.087350026
0.1           -0.3715646    0.083690271
0.15000001    -0.37667379   0.080124266
0.2           -0.38154718   0.076654971
0.25          -0.38619456   0.073284775
⋮             ⋮             ⋮
```

The first corresponds to time, the second column to the first variable in the `lecar.ode` file, which is V, and the thrid column corresponds to the second variable in the `.ode` file, which is W.

-icfile *filename* This loads initial conditions from the named file. Initial conditions are expected for any variables which include differential equation, Wiener, and Markov variables. The format for an `.ic` file is illustrated below for `lecar.ode` (found in the examples `/ode` folder).

An Example `lecar.ic`:

```
-0.3606
0.0911
```

The first initial condition will be mapped to the first variable in the `lecar.ode` file, which is V, and the second value will be mapped to the second variable in the `.ode` file, which is W.

-forecolor *color* This sets the RGB hexadecimal color (e.g. 000000) for foreground in the GUI. The foregrouns is all the characters on the menus and the default drawing color. There are lots of web sites that give listings of hexadecimal color codes. Just look them up or experiment. The first two digits are red, then green, then blue. So, for example, 660066 is a rather deep purple.

-backcolor *color* This sets the hexadecimal color (e.g. EDE9E3) for background in the GUI. The backcolor is the color of the data browser, sliders, and menu backgrounds.

-mwcolor *color* This sets the hexadecimal color (e.g. 808080) for the main window in the GUI. The main window color is the area surrounding the windows, sliders, and menus.

-dwcolor *color* This sets the hexadecimal color (e.g. FFFFFF) for the drawing window in the GUI. This is the colow of all the drawing windows in AUTO and XPP.

-backimage *filename* This sets the name of bitmap file (.xbm) to tile in background. Several .xbm files are included in the Xpp installation folders, but you may also create your own.

For example, the following text saved to a file named `stipple2.xbm` can be loaded to impart a stippled background.

```
#define stipple2_width 2
#define stipple2_height 2
static char stipple2_bits[] = {
  0x02,0x01};
```

xbm is an odd format but there are some conversion programs out there. convert(.exe) will work and runs on all platforms.

-grads B This specifies if color gradients will (B=1) or will not (B=0) be used in the GUI buttons. I don't like them so turn them off as they slow the redrawing of the menus down.

-width N This specifies this minimum width in pixels of main window of the GUI. The value N should be an integer no larger than your screen's physical width.

-height N This specifies this minimum height in pixels of main window of the GUI. The value N should be an integer no larger than your screen's physical height.

-bell B This determines if the system bell on events will (B=1) or will not (B=0) be used. This can be especially useful for users requiring increased visibility or to assist people with disabilities. In addition, most OS can be configured to use a "visual beep" so that the screen will *flash* on certain Xpp events.

-internset B This specifies that internal sets will (B=1) or will not (B=0) be run during batch run

-uset *setname* This names an internal set to be run during batch run. Multiple -uset can be given on the same command line.

-rset *setname* This names an internal set that should not be run during batch run. Multiple -rset can be given on the same command line.

-include *filename* This names a file which will be included along with the selected `.ode` file (see `include` directive in `.ode` file format).

-qsets This simply queries the names of internal sets and the results are saved to OUTFILE. This feature can allow external programs or scipts to access information about an .ode model.

-qpars This simply queries the parameters and the results are saved to OUTFILE. This feature can allow external programs or scipts to access information about an .ode model.

-qics This simply queries the initial conditions and the results are saved to OUTFILE. This feature can allow external programs or scipts to access information about an .ode model.

-quiet *B* This specifies that verbose log messages will (B=0) or will not be (B=1) written

-logfile *filename* This names the file to which verbose log messages are written.

-anifile *filename* This loads an Xpp animation (.ani) from the named file at start-up. This can be useful for teaching demonstrations.

-mkplot This is used in conjunction with the -silent option and will produce a plot that has an appropriate name. (It will be named after the ODE file if only one plot is to be produced; otherwise, a number will be appended.) The plot types are currently SVG and PS (Postscript is default).

-noout will not write an output data file in batch mode. Use this along with -mkplot to supress the generation of data.

-plotfmt *ps—svg* sets the plot type format to be postscript or SVG.

-version This outputs the version of Xpp.

-ncdraw *k* If you are creating a figure and want the nullclines, then set *k=1*. Obviously, the ode file must be set up so that there are variables on the two axes and the dimensions of the plot are what you want. If you want the nullclines to be dumped to a file, use *k=2* and the `-silent` option. The file is always called `nullclines.dat`. The file format is a bit strange but designed to work with gnuplot. The X-nullcline is drawn first followed by the Y-nullcline. Look at an example to see the format. For example the following will compute the nullclines, dump them to a file and will not save the output of a run of the ode:

```
xppaut lecar.ode -noout -silent -ncdraw 2
```

On the other hand to create a plot try this:

```
xppaut lecar.ode -noout -silent -ncdraw 1 -dfdraw 1 -mkplot
```

-dfdraw *k* will draw the direction fields in the same manner as the nullclines above. *k=1,2,3* will draw unscales, scaled, or colorized versions on your plot while *k=4,5* will output the coordinates of the scaled or unscaled to a file called `dirfields.dat`.

-readset *filename* allows you to load anything that can be written in an internal set file; that is, OPTIONS, PARAMETERS, INITIAL CONDITIONS. The file consists of one line (up to 1024 characters). For example; the file (call it `tst.opt`) consists of the line:

```
iapp=0.1;phi=.05;total=500
```

Then the command line to run XPP would be:

```
xppaut lecar.ode -readset test.opt
```

This could also be run in silent mode as well as combining with other options like -silent or -mkplot.

-with *string* is exactly the same as the previous command but now all the options etc are contained in the string which should be delimited by quotes. DO NOT USE ANY SPACES!!!. So, for example:

```
xppaut lecar.ode -with "iapp=0.1;phi=0.05;total=1000" -runnow
```

will run XPP setting some parameters and the total time.

The only other thing on the command line should be the file name. Thus,

```
xppaut test.ode -xorfix -convert
```

will convert `test.ode` to the new format and run it with the `xorfix`.