Network© Version 1.40

User Manual

December 26, 2001

Please send all comments and bug reports to Farzan Nadim (farzan@njit.edu).

0. Introduction

Network is a neuronal network simulation software implemented in C. Network can be used to model single neurons with Hodgkin-Huxley type dynamics, or networks of such neurons connected with synapses. Network supports both a 4th order Runge-Kutta and a variable-time-step integration method.

The current version of Network runs in UNIX and the parser is written in bash shell script and gawk. Network is run from the command line and outputs to standard output. There is no graphical interface. At the command line, network reads the name of the project (e.g. *project*) and looks in the current directory for a configuration file (*project.cfg*) and a file containing the equations and parameters (*project.par*).

In this text, the command line commands (`$` denotes the command prompt)

```
$ command
```

and the contents of files are displayed using the `Courier` font.

1. Running network

The network command line is of the form

```
bash$ network [OPTIONS] project
```

The first argument on the command line is the (user-assigned) name of the project. Assignments of the cell, synapse or network configurations are done in the file `project.cfg`. Assignments of the equations and parameters are done in the file `project.par`. The program searches for these 2 files in the current directory and gives an error message if these files are not found. An optional file `project.ics` can be provided in the current directory for assignment of initial conditions. If this file exists, network automatically reads it. At the end of each simulation run, network produces a file named `last` that contains the last point of integration. This file can be renamed to `project.ics` to be used as the initial condition in the next run.

If no option is passed to network, by default it will produce the *.c and *.h files and a makefile in the current directory, compiles them and runs the executable. The output is directed to the standard output. The output is currently formatted as a matrix where each row represents one time step. The first element of the row is time and is followed by all the state variables in the order of definition.

Additional options must appear *before* the project name. The order of the options is irrelevant. The options are passed to network in the standard UNIX format with a hyphen. Whenever the option requires an argument (such as `-b begin_time`), the space between the 2 parts is *required.* The current options are:

`-b begin_time`

`-c` generate the *.c and *.h files, compile and produce an executable (called `runnetwork`) in the current directory and exit without running the executable.

`-e end_time`

`-h delta_time`

`-k filename` keep only variables written in filename.

`-keep` same as `–k`.

`-notabulate` run without tables (default).

`-nt` same as `-notabulate`.

`-par filename` use this option to pass additional parameters to network. This option is useful when changing parameters of the model without compiling. Since this file `filename` is always read after `project.par`, it is a good way to modify parameters without touching the original `project.par` file.

`-r` run the executable (`runnetwork`) in the current directory without compiling the functions. Note that using the options –c and –r together is superfluous since the latter option overrides the former. To compile and run use neither of the two options.

`-rm` remove the *.c and *.h files from the current directory after producing the executable file.

`-s skip` only the first point of this many steps is printed to standard output. Use this option when you have a small delta_time to keep the output small.

`-t` run with tables. This option makes look-up tables for all the steady-state activation and inactivation and time constant functions. The tables are from –150 to 150 mV in increments of 1 mV. Values in between are obtained by linear interpolation. Values outside the range are set to the boundary values.

`-tabulate` same as `-t`.

`-v` version.

`-version` same as `-v`.

All options may be included in the file `$HOME/.network_options` and/or `project.opt` (in the project directory). These files are read in order and latter assignments override the former ones. Command line assignments are read last and override those in the option files. The options that may be included in the option files are the following in Table 1. These options should be included in the file as

`option=value`

with no space in between. The # sign at the beginning of the line comments out that option.

The Configuration File

The contents of the file `project.cfg` are necessary for determining the structure of the model. The configuration file is a list of statements, one per line, that describe the structures of the cells or compartments, their connections and the names and types of intrinsic and synaptic currents, without specifying any parameters. Empty lines and anything following a # symbol on each line is ignored. Additional spaces and tabs are also ignored. The program is case sensitive.

Table 1. Options in the option files

Option	**Command Line Equivalent**	**Default Value**	**Other Values**	**Descritpion**
REMOVEFILES	-rm	0	1	remove non-executables
ERRORFILE	-	/dev/null	filename	less serious errors
errorfile	-	/dev/stderr	filename	more serious errors
outputfile	-	/dev/stdout	filename	output
parse	-parse	1	0	parse or not
run	-r	1	0	run or not
compile	-c	1	0	compile or not
tabulate	-t	0	1	tabulate inf and taus (speeds up integration)
t0	-b	0	any number	begin time
tmax	-e	100	any number > t0	end time
dt	-h	0.1	any number	integration dt
skip	-s	10	any positive integer	print one point every “skip”
dd	-	0	1	pipe output through “dd bs=10000” to speed up write to disk
par2	-par	“”	filename	additional parameter file
tolerance	-	-1	any number > 0	tolerance > 0 forces the program to use a variable time step integrator for stiff equations. Otherwise RK4 is used.
varfileoption	-keep	“”	filename	filename contains variables to be kept.

The statement

```
compartment name
```

declares a compartment called `name`. The terms `compartment` and `cell` are equivalent. Such a line also indicates that the lines immediately following will define the contents of this `compartment`. The statements immediately following are usually definitions of intrinsic ionic currents (see Table 2 for a list):

```
Passive   L
```

```
mhTauInf  Na
mTauInf   Kd
```

These 3 lines define 3 ionic currents (in the last defined compartment). The first entry on each line defines the type of current and the second entry is the user-assigned name.

Table 2. Available types of intrinsic ionic current	
Type of intrinsic current	**Properties**
`Passive`	leak current
`mTauInf`	single-gated (activation) current
`mInst`	single-gated (activation) instantaneous current
`mhTauInf`	double-gated (activation/inactivation) current
`mInst_hTauInf`	double-gated (activation/inactivation) current with instantaneous activation

The statement

```
synapse comp1 comp2
```

declares a synaptic connection from the compartment (cell) `comp1` to the compartment (cell) `comp2`. This statement is only meaningful when it is immediately followed by a new line containing the definition of the synapse. For example:

```
GradedmTauInf  S
```

defines a graded synapse named S. See Table 3 for available types of synapses.

The statement

```
connect comp1 comp2
```

connects the compartment `comp1` to the compartment `comp2` symmetrically.

Table 3. Available types of synaptic current	
Type of synaptic current	**Properties**
`GradedmTauInf`	graded (single-gated) synapse
`GradedmInst`	graded (single-gated) instantaneous synapse
`GapJunction`	gap-junctional (electrical) synapse (NOT SYMMETRIC)
`GradedmTauInf_mpostTauInf`	graded (single-gated) synapse with single-gated (activation) dependence on the postsynaptic potential
`GradedmInst_mpostInst`	graded (single-gated) instantaneous synapse with single-gated (activation) instantaneous dependence on the postsynaptic potential
`AlphaFunction`	spike-mediated alpha function $\alpha t\ e^{1-\alpha t}$
`PulseStim I`	does not define a true synapse, but allows a set of square pulse currents of a fixed amplitude, duration and frequency to be injected into the postsynaptic cell

`ControlledPulse`	does not define a true synapse, but allows a set of square pulse currents of a fixed amplitude, duration and frequency to be injected into the postsynaptic cell, depending on whether the presynaptic cell is above or below some fixed threshold

The Parameter File

This section describes the contents of the file `project.par`. These contents depend on the cells and currents defined in the configuration file. The values in the tables are just given as examples. Empty lines and anything following a # symbol on each line is ignored. Additional spaces and tabs are also ignored. The program is case sensitive. The units are arbitrary and the user should be careful to balance the units.

Parameter definitions for intrinsic currents are given in the following Table 4. (`x`, `X`, `v` and `V` are equivalent.)

Table 4. Intrinsic parameters that should be defined

If this is in `project.cfg` file	These should be in `project.par` file

`cell Name`
or
`compartment Name`
	`Name_Iext=0.0 # pA`

	`Name_Cm=100.0 # pF`
	`Name_Ga=100.0 # nS (If more than 1 compartment)`
`Passive L`	`Name_L_Gmax=10.0 # nS`
	`Name_L_Erev=-60.0 # mV`
`mInst Na1`	`Name_Na1_Gmax=100.0 # nS`
	`Name_Na1_Erev=50.0 # mV`
	`Name_Na1_mpower=3 # must be an integer`
	`Name_Na1_minf=1/(1+exp(-(x+45)/2))`
`mTauInf Na2`	`Name_Na2_Gmax=120.0 # nS`
	`Name_Na2_Erev=50.0 # mV`
	`Name_Na2_mpower=3 # must be an integer`
	`Name_Na2_minf=1/(1+exp(-(x+55)/3))`
	`Name_Na2_mtau=1+10/(1+exp(-(x+55)/3))`
`mInst_hTauInf K1`	`Name_K1_Gmax=120.0 # nS`
	`Name_K1_Erev=-80.0 # mV`
	`Name_K1_mpower=4 # must be an integer`
	`Name_K1_minf=1/(1+exp(-(x+51)/12))`
	`Name_K1_hpower=1 # must be an integer`
	`Name_K1_hinf=1/(1+exp((x+50)/3))`
	`Name_K1_htau=10+100/(1+exp((x+52)/3))`
`mhTauInf Ca`	`Name_Ca_Gmax=11.0 # nS`
	`Name_Ca_Erev=-80.0 # mV`
	`Name_Ca_mpower=2 # must be an integer`
	`Name_Ca_minf=1/(1+exp(-(x+47)/2))`
	`Name_Ca_mtau=10+12/(1+exp((x+52)/3))`

	Name_Ca_hpower=1 # must be an integer
	Name_Ca_hinf=1/(1+exp((x+50)/3))
	Name_Ca_htau=100+200/(1+exp((x+52)/3))
mAlphaBeta K	Name_K_Gmax=120.0 # nS
	Name_K_Erev=-80.0 # mV
	Name_K_mpower=4 # must be an integer
	Name_K_malpha=-0.01*(v+55)/(exp(-(v+55)/10)-1)
	Name_K_mbeta=0.125*exp(-(v+65)/80)
mhAlphaBeta Na	Name_Na_Gmax=36.0 # nS
	Name_Na_Erev=50.0 # mV
	Name_Na_mpower=4 # must be an integer
	Name_Na_malpha=-0.01*(v+55)/(exp((v+55)/10)-1)
	Name_Na_mbeta=0.125*exp(-(v+65)/80)
	Name_Na_hpower=1 # must be an integer
	Name_Na_halpha=0.07*exp(-(v+67)/100)
	Name_Na_hbeta=1/(exp(-(v+37)/10)+1)

Parameter definitions for synaptic currents are given in the Table 5. (`x`, `X`, `v` and `V` are equivalent.)

Table 5. Synaptic parameters that should be defined

If this is in `project.cfg` file	These should be in `project.par` file
synapse Pre Post	# Nothing is necessary yet.
GradedmInst S1	Pre_Post_S1_Gmax=10.0 # nS
	Pre_Post_S1_Erev=-80.0 # mV
	Pre_Post_S1_minf=1/(1+exp(-(x+55)/3))
GradedmTauInf S2	Pre_Post_S2_Gmax=10.0 # nS
	Pre_Post_S2_Erev=-80.0 # mV
	Pre_Post_S2_minf=1/(1+exp(-(x+55)/3))
	Pre_Post_S2_mtau=10+20/(1+exp(-(x+55)/3))
GapJunction Elec	Pre_Post_Elec_Gmax=10.0 # nS
GradedmInst_mpostInst S3	Pre_Post_S3_Gmax=10.0 # nS
	Pre_Post_S3_Erev=-80.0 # mV
	Pre_Post_S3_minf=1/(1+exp(-(x+55)/3))
	Pre_Post_S3_mPosttinf=1/(1+exp(-(x+25)/12))
GradedmTauInf_mpostTauInf S4	Pre_Post_S4_Gmax=10.0 # nS
	Pre_Post_S4_Erev=-80.0 # mV
	Pre_Post_S4_minf=1/(1+exp(-(x+55)/3))
	Pre_Post_S4_mtau=10+20/(1+exp(-(x+55)/3))
	Pre_Post_S4_mPosttinf=1/(1+exp(-(x+25)/12))
	Pre_Post_S4_mPosttau=200/(1+exp((x+12)/2))
AlphaFunction A	Pre_Post_A_Gmax=10.0 # nS
	Pre_Post_A_Erev=-80.0 # mV
	Pre_Post_A_AlphaTau=25 # ms
	Pre_Post_A_Threshold=-20 # mV: presyn # threshold
	Pre_Post_A_Slope=1 # 1 or –1: going through # threshold up or down

	Pre_Post_A_Tolerance=1e-3 # defined by # default to be 1e-3. This decides the time # tolerance to within which 2 events # coincide.
	NULL_Post_A_Period=1000 # ms defined only # if the presynaptic cell is NULL. # Determines the period of the synapse.
PulseStim I # defined only with # presyn cell NULL	NULL_Post_A_Period=50 # ms
	NULL_Post_A_Duration=2 # ms
	NULL_Post_A_Amplitude=1000 # pA
ControlledPulse I	Pre_Post_I_Period=100.0 # ms
	Pre_Post_I_Duration=-80.0 # mV
	Pre_Post_I_Amplitude=1000 # pA
	Pre_Post_I_Threshold=-20 # mV
	Pre_Post_I_Delay=10.0 # ms: delay after # threshold crossing & before injection
	Pre_Post_I_Above=1 # 1 or 0: inject above # threshold or below

The Initial Condition File

The initial conditions are read from the file project.ics if this file exists in the project directory. The project.ics file should contain each variable on a separate line and each line should be of the form

variable=<value>

or

variable <value>

Each time network is run, it creates a file called last that contains the last point of integration. This file could be renamed to project.ics and used as initial conditions for the next run.

The UNIX Interface

Network is run from the UNIX command line by typing

```
$ network project > outputfile
```

All files associated with network are included in $NETWORKHOME. This directory should be included in the user path or, alternatively, there should be links to the executable files in /usr/local/bin or /usr/bin.

Networks involves 3 executable commands: network is the main script, parse_network is the parser and tabulate_network builds the lookup tables for the –t option.

2. Advanced Features

The variables file option

If keeping all the variables

Keeping the current and conductance

To keep the values of current (cell_ion_I) and conductance (cell_ion_G), the word `keep` should appear immediately after the name of the ion in the configuration file. For example if `project.cfg` contains

```
cell Pyramidal
  mhTauInf Na keep
```

the output of `network project` will contain two columns for `Pyramidal_Na_G` and `Pyramidal_Na_I`.

The calcium-dependent current

3. Tutorials

Tutorial #1: A passive cell

Make a file called `passive.cfg` containing the following:

```
cell P
  Passive L
```

Make a file called `passive.par` containing the following:

```
P_Iext=0       # pA
P_Cm=200       # pF
P_L_Gmax=10    # nS
P_L_Erev=-60   # mV
```

Make a file called `passive.ics` containing the following:

```
P_Vm=-60
```

At the command line type:

```
   $ network passive > out
```

The program will inform you that it is parsing, compiling and running. The file `out` should contain 1000 lines. On each line there should be 2 entries, the first is time and the second is –60.

Now create a new file called `par2` (this name is arbitrary) that contains

```
P_Iext=1000    # pA
```

At the command line type:

```
   $ cp last passive.ics
   $ network –r –b 100 –e 300 –par par2 passive >> out
   $ cp last passive.ics
   $ network –r –b 300 –e 500 passive >> out
```

You just injected a 1 nA pulse of duration 200 ms into the cell. The result is in the file `out` and can be plotted with `gnuplot`.

Tutorial #2: The Morris-Lecar cell

Tutorial #3: The Hodgkin-Huxley cell

Tutorial #4: A Passive Cable

Tutorial #5: An Active Cable

Tutorial #6: Reciprocal Inhibition