Documentation for networkFVMsims: finite volume method implementation of evolving reaction-diffusion fields on a network

E. F. Koslover

Last updated April 24, 2024

The code in this package can be used to simulated dynamics of buffered calcium ions or of other particles spreading across a tubular network. In its default mode, it treats networks as point-like nodes linked by one-dimensional edges of constant radius. Some extensions are provided for inputting 3D meshes of more complicated structures (eg: sheets), but these are not yet documented. Alternately, rapidly-equilibrating 'reservoirs' can be included in the networks.

The code stores information about network structure (edge connectivity, edge lengths) in a NETWORK object. Note that while network edges can be curved (and thus longer than the node-to-node distance), the spatial embedding of the edges is not stored. The network structure is input by the user in a .net file (see networktools project on github for Matlab code to manipulate, visualize, and output network structures). The code builds its own MESH structure consisting of linked mesh elements. These can be loaded and visualized with the matlab code provided in scripts.

NOTE: this documentation is far from complete and currently only includes the most basic of the features implemented.

1 Compilation Instructions

To compile and run the program, you will need the following:

- a compiler capable of handling Fortran90. The code has been tested with the gfortran compiler.
- Optionally: Matlab to visualize output data

The code has been tested on Ubuntu Linux 20.04.

To compile with gfortran, go into the `source` directory. Type `make`. To compile with any other compiler that can handle Fortran90, type

```
make FC=compiler
```

substituting in the command you usually use to call the compiler.
If the compilation works properly, the executable `treeParticleSim.exe` will appear in the main directory.

2 Usage Instructions

To run the program in the main directory, type:

```
./treeparticleSim.exe suffix > stdout.suffix
```

Here, `suffix` can be any string up to 100 characters in length. The program reads in all input information from a file named `param.suffix` where `suffix` is the command-line argument. If no argument is supplied, it will look for a file named `param`. If the desired parameter file does not exist, the program will exit with an error.

The parameters in the input file are given in the format "*KEYWORD* value" where the possible keywords and values are described in Section 4. Each keyword goes on a separate line. Any line that starts with "#" is treated as a comment and ignored. Any blank line is also ignored. The keywords in the parameter file are not case sensitive. For the most part, the order in which the keywords are given does not matter. All parameters have default values, so you need only specify keywords and values when you want to change something from the default.

3 Example for a Quick Start

A few example parameter files are provided

The main one is `examples/param.MCFO1`): This will run a simulation of particles moving around on a dendritic tree. Particles are produced at the soma, move anterograde until they hit a distal tip, then turn around. Switch to the alternate set of TRANSRATE lines in the parameter file to enable the particles to halt and turn around partway.

To run this simulation:

```
../treeParticleSim.exe MCFO1
```

To parse the snapshots for the example and visualize the particle motion, use
`scripts/visualizeExample.m`

A similar simulation on a simple 2-level tree: `examples/param.prodtree`

Another simulation with a fixed number of particles that reflect off all terminal nodes: `examples/param.reftree`

4 Keyword Index

The code will attempt to read parameters out of a file named `param.suffix` where "suffix" is the command line argument. If no command line arguments are supplied, it will look for a file named `param`. If multiple arguments are supplied, it will read multiple parameter files in sequence.

The parameter file should have one keyword per line and must end with a blank line. All blank lines and all lines beginning with # are ignored. For the most part, the order of the lines and the capitalization of the keywords does not matter. All keywords except *ACTION* are optional. The default values for each parameter are listed below. If a keyword is supplied, then values may or may not be needed as well. Again, the required and optional value types are listed below.

Keywords and multiple values are separated by spaces.

When reading the parameter file, lines longer than 500 characters will be truncated. To continue onto the next line, add "+++" at the end of the line to be continued. No individual keyword or value should be longer than 100 characters.

Floating point numbers can be formated as 1.0, $1.1D0$, $10e-1$, $-1.0E+01$, etc., where the exponential notation specifier must be D or E (case insensitive). Integer numbers can also be specified in exponential notation without decimal points (eg: 1000 or 1E3). Logical values can be specified as T, F, TRUE, FALSE, 1, or 0 (with 1 corresponding to true and 0 to false).

List of Keyword inputs

Some of the below keywords are not actually used in this skeleton code, but are saved for later.

- *ABSORBTERM*
  - value: 1 integer, then several logical values; default: false
  - 1st num = particle type, then flag for whether each state disappears when hitting a terminal node
- *ACTION*
  - value: 1 string of at most 20 characters; no default
  - This keyword sets the overall calculation performed by the program
  - Possible values are: RUNSIM
- *DELT*
  - value: 1 float; default 1D-4
  - Time-step for dynamics.
  - Not used
- *DIRECTEDTREE*
  - 1 logical or nothing; default false
  - If no value provided, flip to true
  - Set up network as a directed tree, placing parent edge first in NODEEDGES and then ordering by edge ID
  - Will raise error if edges are not properly directed
- *INITEDGES*
  - value: 1 or more integers; default = -1
  - List of edges such that particles start uniformly distributed over those edges
  - If first number is negative, start particles uniformly over all edges in the network
- *INITSTATE*

  - value: integer, then 1 or more floats; default: 1 1D0
  - Distribution of starting states
  - first value: particle type
  - float values: probability of starting in each state

- *NETWORKDIM*
  - value: 1 integer; default: 3
  - Dimensionality of space in which network is embedded

- *NETFILE*
  - value: 1 string; default: *.net
  - Input network file (full format with NODE lines, etc) used to start the simulation
  - Any * in the file name will be replaced by the command-line argument (suffix)

- *NPARTINIT*
  - value: 1 integer; default: 1
  - Number of initial particles to place in the system

- *OUTFILE*
  - value: 1 string, up to 100 characters; default: *.out
  - Output file for something (not sure what yet)
  - Any * in the file name will be replaced by the command-line argument (suffix)
  - Not used

- *PRINTEVERY*
  - value: 1 integer; default: 1
  - Print to screen every so many steps

- *PRODUCTION*
  - value: 2 integers, float, 1 or more integers; default: 1 1 0D0 1
  - Control production of new particles at nodes
  - First number: particle type, second number: state that new particles are produced in
  - 3rd number: production rate
  - remaining number: list of nodes where particles are produced

- *READEDGEVALS*
  - values: nothing or 1 logical; default=false

  - if no value provided, switch to true
  - Read in one additional floating point value for each edge from network file
- *READNODEVALS*
  - values: nothing or 1 logical; default=false
  - if no value provided, switch to true
  - Read in one additional floating point value for each node from network file
- *RNGSEED*
  - 1 integer; default: 0
  - seed for random number generator
  - value of 0 will seed with system time in milliseconds
  - value of -1 will use the last 5 characters in the suffix
  - value of -2 will use the last 4 charactes in the suffix and the millisecond time
  - other positive value: the seed is used directly for repeatable simulations (should be positive)
- *RUNSPEED*
  - value: 1 or more floats; default 0D0
  - Not used
- *SNAPSHOTFILE*
  - value: 1 string; default: *.snap.out
  - File for dumping out snapshots. Can also be specified within SNAPSHOTS keyword.
- *SNAPSHOTS*
  - 1 optional integer, 1 optional string, 1 optional logical; defaults: 1, *.snap.out, false
  - Dump snapshots over the course of the calculation
  - integer: how often to dump snapshots; string: snapshot file (* is replaced with suffix); logical: append rather than rewriting the snapshot file
  - Snapshot file contains multiple snapshots of network structure and other info. Should be read with `parseDynNetworkSnapshots.m` script
- *SPLITTYPE*
  - value: 2 integer, string of up to 30 characters
  - Default: `EQUAL_OTHER`
  - numbers are particle type, state. Then a string describing how to split among branches when passing a junction, for each state

  - Allowed values:
    * `EQUAL_OTHER`: equally likely to step on all edges other than the one it starts on
    * `DOWNSTREAM_EDGEVALS`: when moving downstream, split among daughter edges in proportion to edgevals. DIRECTEDTREE must be set
    * `UPSTREAM_EDGEVALS`: when moving upstream, go into trunk. DIRECTEDTREE must be set

- *TRANSRATES*
  - value: 2 integers, then several positive floats; default: 0
  - 1st num = particle type, 2nd num = output state, floats = rates of transitioning from all other states to this output state

- *TRANSTERM*
  - value: 2 integers, then several positive floats; default: 0
  - 1st num = particle type, 2nd num = output state, floats = probability of switching from all other states into this output state when hitting a dead end node

- *VELOCITY*
  - value: integer, then (nstate) floats; default: 0D0
  - For a given type of particle (first number), list the velocity in each state

- *VERBOSE*
  - value: 1 logical; default: false
  - Print extra output. Not currently implemented