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

E. F. Koslover

Last updated May 21, 2024

The code in this package can be used to simulate dynamics of buffered calcium ions or of other particles spreading across a tubular network. In its default mode, it treats networks as one-dimensional edges of constant radius, connected at point-like nodes. 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 current 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 Dependencies

- This project code: `https://github.com/lenafabr/networkSpreadFVM`
- Network manipulation code: `https://github.com/lenafabr/networktools`

2 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 `netmeshdynamicsFVM.exe` will appear in the main directory.

3 Usage Instructions

To run the program in the main directory, type:

```
./netmeshdynamicsFVM.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 5. 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. Most of the parameters have default values, so you need only specify keywords and values when you want to change something from the default.

4 Example for a Quick Start

A few example parameter files are provided

4.1 Ca^{2+} release from a honeycomb network

The parameters for this example run are stored in `examples/param.example1_ca`). Within the examples directory, run the code as:

```
../netmeshdynamicsFVM.exe example1_ca
```

This will run a simulation of calcium ions and buffer proteins diffusing on a tubular network, with a local region that is releasing free calcium.

Output files:

- `example1_ca.out` contains the total flux of calcium out of the network at each time point
- `example1_ca.snap.txt` contains the concentration of free calcium and total buffer protein at all mesh cells, at each snapshot time

To parse this data and generate plots, an example matlab script is provided in `scripts/example1_ca.m`
[TODO: insert example output figure here]

4.2 Spreading of photoactivated particles

The parameters for this example run are stored in `examples/param.example_PA`). Within the examples directory, run the code as:

```
../netmeshdynamicsFVM.exe example_PA
```

This will run a simulation of diffusive particles on a network spreading from an initial bolus of photoactivated particles.
Output files:

- `example_PA.out` contains the total flux of calcium out of the network at each time point
- `example_PA.snap.txt` contains the concentration of free calcium and total buffer protein at all mesh cells, at each snapshot time

To parse this data and generate plots, an example matlab script is provided in `scripts/example_PA.m`

[TODO: insert example output figure here]

4.3 Continuous photoactivation

[TODO: make an example simulation with a continuously photoactivated region]

4.4 Ca^{2+} refill

[TODO: make an example simulation where a reservoir is refilled with calcium]

5 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. 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 avoid this and 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).

6 Network file format

The network files (ending in .net for the provided examples) store information on the network structure. As with the keywords control file, blank lines and lines starting with # are ignored or treated as comments.

Note that the network file does not store the spatial embedding of edges (paths). It store only the node positions, network topology (which nodes connected by which edges) and the length of the edges. Since edges are treated as effectively one-dimensional tubes, their spatial embedding is never used. This means that the output positions of mesh cells along the edges do not accurately represent the paths of curved edges. This is a problem for visualizing snapshots with curved edges afterwards (have to load separate file into matlab containing edge paths) but does not affect the calculations themselves.

The file contains lines that start with the following keywords and values:

- *NODE* (integer, 2-3 floats, optional string)
  - Index of the node. The code will fail if lines for some node indices are missing.
  - Node positions in 2D or 3D
    - If NETWORKDIM is not supplied, the code will determine the dimension of the network spatial embedding from how many node positions are supplied. If you want to fix the dimension explicitly use *NETWORKDIM* keyword.
  - Node label. Can start with 'F' or 'P'. No spaces in the label
    - Pn. Assume this node this permeable for field n. (*eg:* P1 to be permeable for field 1)
    - Rn. This node is connected to well-mixed reservoir n. All nodes connected to reservoir n will always have identical field values.
    - FO. This is an obligate fixed node. It is fixed to its initial value (if FIXNODEFROMNETFILE is set). This node will always be fixed if RANDFIXNODES is used.
    - F or F[anything else]. This is a node that is fixed to some initial value if FIXNODEFROMNETFILE is set. If RANDFIXNODES is turned on, the node is one of the ones that can be randomly selected to be fixed (but will not necessarily be picked).
- *EDGE* (3 integers, 1 float, 1 optional float)
  - Index of the edge. The code will fail if lines for some edge indices are missing.
  - Indices of the 2 nodes which the edge connects
  - Length of the edge. This is **not** necessarily equal to the distance between the nodes as the edge might be curved.
    - Note that this Fortran code does not store or use the spatial embedding (path) of the edge itself.
  - Optionally, store the radius of the edge. This is only used if USEVARRAD is turned on and EDGERADRANDTYPE is NONE (reading radii directly from net file rather than generating them randomly). If this is the case and no radius is supplied in the network file, the edge radius is set to EDGERADBASE.

List of Keyword inputs

[This is currently a very incomplete list. Some lucky student is going to get the task of filling in all the implemented keywords eventually.] In the meantime, look in `source/readkey.f90` for comments and default values of most of the keyword parameters. The variables set by keywords are declared in `source/keys.f90`

By default, the simulations work with concentrations in units of $mM \times \pi a^2$ where a is the tubule radius, length units of μm, and time units of seconds.

- *ACTION*
  - value: 1 string of at most 20 characters; default NONE
  - This keyword sets the overall calculation performed by the program
  - Possible values are: RUNDYNAMICS (the only one currently implemented)
- *BACKGROUNDCONC*
  - value: 1 or more floats (one for each field); default: 0
  - Set initial concentration for each field in all the mesh elements that do *not* have a specific starting concentration set through STARTCONC
- *CEXT*
  - value: 1 or more floats (one for each field); default: 0
  - Set 'external' concentration for each field. This is used only when calculating flux out of permeable mesh cells (see PERMNODE keyword for details)
- *DELT*
  - value: 1 float; default 1D-4
  - Time-step for forward Euler stepping in the dynamics.
  - Not used
- *DOFLOW*
  - value: 1 optional logical; default: true
  - Include advective flow along edges
- *EDGERADBASE*
  - value: 1 float; default $\sqrt{1/\pi}$
  - Default radius of edge tubules. This is used only if `USEVARRAD` is true, `EDGERADRANDTYPE` is 'None', and the radius of a particular edge is not provided in the network file.
- *EDGERADRAND*
  - value: 1 string; up to 10 optional floats. Defaults: 'None'

- Use this to generate a random value for the radius of each network edge (assumed to be constant along the edge).
- String is the type of distribution for the random edge radii. Floats are the parameters governing the distribution.
  - 'None': do not use random radii. Instead read in radii from the network file, or use `EDGERADBASE` by default if no value provided in network file.
  - 'Uniform': uniformly select radius for each edge between some minimum and maximum value (2 parameters provided as floats).

- *GLOBALRESERVOIR*
  - values: 5 floats, 1 logical; defaults: -, -, -, -, -, F
  - Include a special 'global reservoir' that interacts with all (or most) of the mesh cells. By default, there is no such reservoir.
  - This reservoir gets its own concentration fields (stored as part of the mesh structure)
  - Interchange between the global reservoir and the mesh cells is in terms of Michaelis-Menten kinetics and is not based on transport coefficients (ie: unrelated to diffusion or flow across boundaries). Assume only the first field interacts with the global reservoir
  - The values supplied (in order) are:
    - Volume V_g. If working with 1D concentrations, instead supply $V_g^{(1D)}/(pia^2)$ where a is the tubule radius.
    - Recovery rate constant k_r in units of per area per time. If working with 1D concentrations, instead supply $k_r^{(1D)}(2\pi a)$
    - K_{Mr} = saturation concentration for recovery.
    - Rate constant k_{out} for pumping out of the global reservoir. Units of time^{-1}.
    - $K_{M,\text{out}}$ = saturation concentration for pumping out.
    - PERMTOGLOBALRES. If set to True, permeable nodes release their particles into the global reservoir rather than the extracellular environment.

- *GLOBALRESVSTART*
  - value: 1 float; default 0D0
  - Initial concentration in global reservoir (1st field only. The others are assumed to be 0)

- *MESHFILE*
  - value: 1 string, up to 100 characters; default: *.mesh.txt
  - File name in which to output mesh structure.
  - * is replaced by the command-line suffix

- *MESHSIZE*

  - values: 1 float, 1 optional integer; defaults: 0.1, 2
  - Float: limit for maximum length of an edge mesh cell
  - Integer: minimum number of internal mesh cells along an edge (does not include nodes). Gives smaller mesh cells when edges are short.

- *NETWORKDIM*
  - value: 1 integer; default: 0
  - Dimensionality of space in which network is embedded
  - If positive value: explicitly defines the spatial dimension in which the network is embedded
  - If 0, use the .net file (# items in a NODE row − 2) to set the dimension

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

- *OUTFILE*
  - value: 1 string (up to 100 characters) and 1 optional integer
  - default: *.out 1000
  - String: output file for the flux of released calcium over time
  - Any * in the file name will be replaced by the command-line argument (suffix)
  - The integer N indicates the flux will be written to the file every N timesteps of the simulation. This can also be set separately with keyword OUTPUTEVERY

- *OUTPUTEVERYSTART*
  - value: 2 integers, `OUTPUTEVERYSWITCH` and `OUTPUTEVERYSTART`; default: 0, 10
  - default: 0 10
  - Output flux more often at the start of the run (when there might be very steep release profiles). The two integers are: `OUTPUTEVERYSWITCH` (output more frequently until you hit this number of steps) and `OUTPUTEVERYSTART` (how often to output for the initial period)

- *OUTPUTTOTFLUXONLY*
  - value: 1 optional logical; default: True
  - Output total flux out of the network into the OUTFILE. If set to false, then will output flux from each of the permeable or fixed nodes.

- *PRINTEVERY*

- value: 1 integer; default: 1
- Print simulation progress to screen every so many steps

- *PERMNEARNODEDIST*
  - value: 1 float; default: -1.0
  - Any mesh cell within the given spatial distance of a permeable node will also be made permeable.
  - Warning: unintuitive behavior. because the Fortran code does not know the spatial embedding of the edges (edgepaths), it will stretch the edges out as if they were the same length but straight. This means that up until a neighboring node is hit, the distance set by this keyword is a graph distance rather than a spatial distance. The two are not the same for curved edges.
  - If the value given is negative then only the mesh cell corresponding to specified nodes (set via PERMNODE keyword) is made permeable.

- *PERMNODE*
  - value: 1 integer n, 1 or more floats p; defaults: – 0
  - Make a specific node permeable to one or more of the concentration fields. By default, there are no permeable nodes.
  - n is the node index. p is the permeability or permeability prefactor, scaled as described below, for each of the fields.
  - If P is the membrane permeability in units of length/time, the current out of each mesh cell is $I = AP(c_i - c_{\text{ext}}) = \frac{AP}{\pi a^2}(\rho_i - \rho_{\text{ext}})$ where c_i are 3D concentrations and ρ_i are 1D concentrations. A_i is the mesh cell surface area.
  - If working with 1D concentrations (the default) then:
    - If PERMPREFACTOR is true, then $p = 2P/(aD)$ where D is the particle diffusivity, a is the tubule radius. The current is $I = [p(\frac{A}{2\pi a})D](\rho_i - \rho_{\text{ext}})$
    - If PERMPREFACTOR is false, then $p = AP/(\pi a^2)$ where A is the area of a mesh cell (in units of length^2, assumed the same for all mesh cells). The current is $p(\rho_i - \rho_{\text{ext}})$
  - If working with 3D concentrations then:
    - If PERMPREFACTOR is true, then $p = P/D$. The current is $I = [pAD](c_i - c_{\text{ext}})$
    - If PERMPREFACTOR is false, then $p = AP$. The current is $I = [p](c_i - c_{\text{ext}})$.

- *PERMPREFACTOR*
  - value: 1 optional logical; default: False if keyword not supplied. True if keyword is supplied by itself
  - If this is true, then the permeability value set by PERMNODE is treated as a prefactor, to be scaled by $A \cdot D$ (where A is surface area of mesh cell and D is particle diffusivity) when incorporated into simulations

  - Strongly recommended: set this to true (larger mesh elements should have higher permeability)
  - If this is false, then the permeability is set directly by the value given via PERMNODE keyword, regardless of mesh cell size

- *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. This is useful when launching many iterations nearly simultaneously on a cluster.
  - Other values: the seed is used directly for repeatable simulations.

- *SNAPSHOTFILE*
  - value: 1 string; default: *.snap.out
  - File for dumping out snapshots (concentration fields on all mesh elements). 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 concentration profiles on all the mesh elements. Should be read with `loadSnapshotFVM.m` script

- *UNIFORMBUFFER*
  - value: 1 optional logical; default false
  - Assume the concentration of total buffer sites is uniform in space. This is true if there are no flows and no leakage of buffer

- *USEVARRAD*
  - value: 1 optional logical; default true if keyword is present, false otherwise
  - If this is set to true, then each mesh cell is assigned a radius that may vary along or between edges and affects flux from one mesh cell to another.
  - Control how edge radii are assigned with EDGERADBASE, EDGERADRAND

- *VERBOSE*
  - value: 1 optional logical; default if not present: false; default if value unspecified: true
  - Print extra output. Not really implemented in a useful way right now.