Usage Guide for M3SYM v1.1

Papoian Lab, University of Maryland

Contents

1 Introduction

Cell motility plays a key role in human biology and disease, contributing ubiquitously to such important processes as embryonic development, wound repair and cancer metastasis. The Papoian laboratory is interested in gaining deeper understanding of the physical chemistry behind these complex, far-from-equilibrium mechanochemical processes. His approach and model, named the *Mechanochemical Dynamics of Active Networks* (MEDYAN), is based on combining stochastic reaction-diffusion treatment of cellular biochemical processes with polymer physics of cytoskeletal filament network growth, while explicitly coupling chemistry and mechanics. For a more detailed description of MEDYAN, see [1].

The Papoian laboratory has developed M3SYM, a software package based on the MEDYAN model, to simulate growth dynamics of actin based filamentous networks *in vitro* and *in vivo*. Recent papers where M3SYM or its predecessor, StochTools, were used can be found on the publication section of the Papoian group's main web page. The M3SYM package can also be extended to simulate the dynamics of any active matter network.

2 Overview of Features

M3SYM is a package that can simultaneously simulate complex chemical and mechanical dynamics of an active matter network. For more information on the MEDYAN model, which M3SYM implements, see [1].

2.1 Chemical capabilities

The chemical capabilities of M3SYM include:

- Stochastic reaction-diffusion on a three dimensional grid using stochastic simulation algorithms, including the *direct* Gillespie algorithm and the *Next Reaction Method*.
- Complex chemical representation of filaments, allowing for heterogeneous chemical monomers in a single filament segment.
- A wide range of filament reactions, including:
  - Polymerization of either end of filament
  - Depolymerization of either end of filament
  - Filament nucleation, either branching or spontaneous
  - Severing of filament at chosen sections
  - Cross-linker binding and unbinding to filament
  - Motor walking, binding, and unbinding to filament
  - Monomeric reactions within a filament

2.2 Mechanical capabilities

M3SYM allows for a wide range of mechanical interactions, including:

- Force fields for filament interactions, including
  - Filament stretching and bending
  - Branching point stretching, bending, and dihedral
  - Excluded volume
- Stretching force fields for cross-linkers and motors
- Boundary steric repulsion force field
- Bubble repulsion force field

These force fields can be minimized by a choice of conjugate gradient algorithm.

2.3 Mechanochemical coupling

M3SYM couples chemistry and mechanics by altering reaction rates based on mechanical stresses in a given network. This allows for a full treatment of the complex mechanochemical responses in active matter networks. See [1] for a more detailed description.

3 Running M3SYM

To run the M3SYM executable, execute the following command in the terminal shell:

```
> ./M3SYM -s <SystemFile> -i <InputDirectory> -o <OutputDirectory>
```

The `SystemFile` will be described in the later sections.

The `InputDirectory` specifies where all input files are contained, with their names being specified in the `SystemFile`. This must be an absolute directory path. The `OutputDirectory` specifies where the produced output will be placed. This also must be an absolute directory path. See the later sections for details on input and output files.

4 Input

4.1 System file

The system file is a simple text file that defines all parameters of the simulation. The M3SYM executable must take in a system file as a command line argument.

Each parameter must be defined in the following syntax:

<PARAMETER>: <PARAMETERVALUE>

where the parameter name is followed by a semicolon, and the value of the parameter is placed after the semicolon. Outlined below are the parameters that can be included.

Unless otherwise noted, all distance parameters are in units of nanometers.

All filament properties must be listed in the order of filament definition, and must be consistent across all definitions. To list various parameter values, provide a space between those values after the parameter qualifier. In cases where a filament type is needed, filament types range from 0 to NUMFILAMENTTYPES, as defined accordingly.

Parameter options will be listed as <PARAMETER> – value type – description.

4.1.1 Geometry

The following geometric parameters can be set. All geometry parameters must be set in the system file, or a startup error will result.

NDIM – 1, 2, 3 – Number of dimensions in system.

NX – int – Number of compartments in X direction.

NY – int – Number of compartments in Y direction.

NZ – int – Number of compartments in Z direction.

COMPARTMENTSIZEX – double – Size of compartment in X direction.

COMPARTMENTSIZEY – double – Size of compartment in Y direction.

COMPARTMENTSIZEZ – double – Size of compartment in Z direction.

MONOMERSIZE – double – Size of monomer for filament growth .

CYLINDERSIZE – double – Size of cylinder in filament.

BOUNDARYSHAPE – SPHERICAL, CUBIC, CAPSULE – Boundary shape.

BOUNDARYDIAMETER – double – Diameter for applicable shapes, including SPHERICAL and CAPSULE geometries.

If movement of boundaries is desired, the following parameters can also be set. If these parameters are not set, the system will assume non-moving boundaries. Currently, moving boundaries are only implemented for the CUBIC boundary shape.

`BOUNDARYMOVE` – NONE, ALL, TOP – Movement of a boundary. ALL specifies that all boundaries will move in the given direction, and top specifies that the top of the boundary in the z direction will move.

`BMOVESPEED` – double – Speed of boundary movement in nm/s. If a negative value is given, the boundary will move towards the center of the grid. If positive, the boundary will move away from the center of the grid.

`BMOVESTARTTIME` – double – Time at which the boundary will begin to move. If not specified, the boundary will start moving at the beginning of the simulation.

`BMOVEENDTIME` – double –Time at which the boundary will stop movement.

4.1.2 Mechanics

The following mechanical parameters can be set. It is noted that the number of parameters for each force field must match the number of species of that type, specified in the `SystemFile`. This must be consistent for all simulation elements, including filaments, cross-linkers, motors, branchers, and bubbles. To set multiple parameters corresponding to multiple species, list the parameter values with space in between after the parameter qualifier.

Force field constant units are dependent on the potential used, but in general will be in pN and nm scaling. For more information on force fields used in the MEDYAN model, see [1]. If a force field type is left blank, that force field will not be included in the simulation.

`CONJUGATEGRADIENT` – POLAKRIBIERE, FLETCHERRIEVES, STEEPESTDESCENT – Type of conjugate gradient minimization.

`GRADIENTTOLERANCE` – double – Gradient tolerance in conjugate gradient (in pN).

`MAXDISTANCE` – double – Maximum distance beads can be moved in minimization.

`LAMBDAMAX` – double – Maximum lambda that can be returned in line search.

`FSTRETCHINGTYPE` – HARMONIC – Filament stretching force field.

FSTRETCHINGK – double – Filament stretching force constant.

FBENDINGTYPE – HARMONIC, COSINE – Filament bending force field.

FBENDINGK – double – Filament bending force constant.

FBENDINGTHETA – double – Filament bending angle (radians).

LSTRETCHINGTYPE – HARMONIC – Cross-linker stretching force field.

LSTRETCHINGK – double – Cross-linker stretching force constant.

MSTRETCHINGTYPE – HARMONIC – Motor stretching force field.

MSTRETCHINGK – double – Motor stretching force constant.

BRSTRETCHINGTYPE – HARMONIC – Branching point stretching force field.

BRSTRETCHINGK – double – Branching point stretching force constant.

BRBENDINGTYPE – COSINE – Branching point bending force field.

BRBENDINGK – double – Branching point bending force constant.

BRBENDINGTHETA – double – Branching point bending angle (radians).

BRDIHEDRALTYPE – COSINE – Branching point dihedral force field.

BRDIHEDRALK – double – Branching point stretching force constant.

BRPOSITIONTYPE – HARMONIC – Branching point position force field.

BRPOSITIONK – double – Branching point position force constant.

VOLUMEFFTYPE – REPULSION – Volume force type.

VOLUMECUTOFF – double – Volume interaction cutoff distance.

VOLUMEK – double – Volume force constant.

BOUNDARYFFTYPE – REPULSIONEXP – Boundary force type.

BOUNDARYCUTOFF – double – Boundary interaction cutoff distance.

BOUNDARYINTERACTIONK – double – Boundary force constant.

`BOUNDARYSCREENLENGTH` – double – Boundary screening length constant.

`BUBBLEFFTYPE` – REPULSIONEXP – Bubble force type.

`BUBBLECUTOFF` – double – Boundary interaction cutoff distance.

`BUBBLEINTERACTIONK` – double – Bubble force constant.

`BUBBLESCREENLENGTH` – double – Bubble screening length constant.

`BUBBLERADIUS` – double – Bubble radius.

`NUMBUBBLETYPES` – int – Number of different bubble types.

`MTOCFFTYPE` – ATTACHMENTHARMONIC – MTOC force type.

4.1.3 Chemistry

The following chemical parameters can be set. It should be noted that the number of parameters listed for each chemical species type that resides on a filament must match the number of filament types, specified in the `SystemFile`. This must be consistent for all filament types. To set multiple parameters corresponding to multiple filaments, list the parameters with space in between after the parameter qualifier.

All chemical parameters must be set unless otherwise noted in the description. For the motor parameters, the number of parameters must match the number of motor species in the system. For more information on chemical algorithms, see [1].

`CHEMISTRYFILE` – string – Input chemistry file. Should be in the `InputDirectory`.

`CALGORITHM` – GILLESPIE, NRM – Chemistry algorithm used.

`NUMTOTALSTEPS` – int – Number of total chemical steps. If RUNTIME is set, will not be used.

`RUNTIME` – double – Total runtime of simulation (s).

`NUMSTEPSPERS` – int – Number of steps per snapshot. If SNAPSHOTTIME is set, will not be used.

`SNAPSHOTTIME` – double – Time of each snapshot (s).

`NUMCHEMSTEPS` – int – Number of chemical steps per mechanical equilibration.

`NUMSTEPSPERN` – int – Number of chemical steps per neighbor list update. This includes updating chemical reactions as well as force fields which rely on neighbor lists.

`NUMDIFFUSINGSPECIES` – int – Diffusing species in system.

`NUMBULKSPECIES` – int – Bulk species in system.

`NUMFILAMENTTYPES` – int – Number of different filament types.

`NUMFILAMENTSPECIES` – int – Filament species in system for each filament type defined.

`NUMPLUSENDSPECIES` – int – Plus end species in system for each filament type defined.

`NUMMINUSENDSPECIES` – int – Minus end species in system for each filament type defined.

`NUMBOUNDSPECIES` – int – Bound species in system for each filament type defined.

`NUMLINKERSPECIES` – int – Cross-linker species in system for each filament type defined.

`NUMMOTORSPECIES` – int – Motor species in system for each filament type defined.

`NUMBRANCHERSPECIES` – int – Brancher species in system for each filament type defined.

`NUMBINDINGSITES` – int – Number of binding sites per cylinder for each filament type defined. This will set binding sites for cross-linkers, motors, and other binding molecules.

`NUMMOTORHEADSMIN` – int – Minimum number of motor heads per motor species defined.

`NUMMOTORHEADSMAX` – int – Maximum number of motor heads per motor species defined.

`MOTORSTEPSIZE` – double – Single motor head step size.

4.1.4 Dynamic rates

The following dynamic rate forms and parameters can be set. These parameters are characteristic lengths and amplitudes of the rate changing equations outlined in [1]. These can be tuned to mimic the stall and unbinding mechanochemical coupling of cross-linkers and myosin II motors. Note that if dynamic rates are enabled, the number of dynamic rate forms for each type of reaction must match the number of species of that type specified in the `SystemFile`, i.e. the number of forms for cross-linker unbinding must match the number of cross-linker species, etc.

The number of parameters specified for each type of dynamic rate form must match the number of parameters required for those forms. See below for details, and see [1] for more

information on the explicit forms. Parameters must be listed in order of the form that they correspond to, also corresponding to the species that they represent.

`DFPOLYMERIZATIONTYPE` – BROWRATCHET – Filament polymerization dynamic rate form.

`DFPOLYMERIZATIONLEN` – double – Characteristic length for filament polymerization dynamic rate form.

`DLUNBINDINGTYPE` – BASICCATCHSLIP, BASICSLIP – Cross-linker unbinding dynamic rate form. If BASICCATCHSLIP, two parameters for DLUNBINDINGLEN and DLUNBINDINGAMP are needed to define the functional form. If BASICSLIP, one DLUNBIDINGLEN is needed to define the functional form.

`DLUNBINDINGLEN` – double – Characteristic length of cross-linker unbinding dynamic rate form.

`DLUNBINDINGAMP` – double – Amplitude of cross-linker unbinding dynamic rate form.

`DMUNBINDINGTYPE` – LOWDUTYPCMCATCH – Myosin II unbinding dynamic rate form.

`DMUNBINDINGFORCE` – double – Characteristic force of myosin II unbinding dynamic rate form.

`DMWALKINGTYPE` – LOWDUTYHILLSTALL – Myosin II walking dynamic rate form.

`DMWALKINGLEN` – double – Characteristic force of myosin II walking dynamic rate form.

4.1.5 Starting filament configuration

The following filament initialization parameters can be set. These parameters define the initial configuration and length of filaments in the system. It is noted that at least one filament, plus end, and minus end chemical species must be initialized in the chemistry input file, or a startup error will result.

`FILAMENTFILE` – string – Name of filament initialization file. This is not required.

`NUMFILAMENTS` – int – Number of random filaments to initialize. These filaments will be randomly distributed in the system volume.

`FILAMENTLENGTH` – int – Number of cylinders per filament to initialize, defining the initial length of the filaments.

`FILAMENTTYTPE` – int – Filament type to initialize.

4.1.6 Starting bubble configuration

The following bubble initialization parameters can be set. These parameters define the initial configuration of bubbles in the system, similar to the filament configuration parameters. It is noted that at least one type of bubble must be set, or a startup error will result.

`BUBBLEFILE` – string – Name of bubble initialization file. This is not required.

`NUMBUBBLES` – int – Number of random filaments to initialize. These filaments will be randomly distributed in the system volume.

`BUBBLETYTPE` – int – Bubble type to initialize.

4.1.7 Special setup

The following special setups can be initialized with corresponding parameters. These setups must be set on different lines, so users should specify a new parameter, on separate lines, for each setup desired:

`SPECIALSETUP` – MTOC – Type of special setup.

For a microtubule organizing center (MTOC) setup, the following parameters can be defined:

`MTOCFILAMENTTYPE` – int – Type of filament to attach to the MTOC.

`MTOCNUMFILAMENTS` – int – Number of filaments to create and attach to the MTOC.

`MTOCFILAMENTLENGTH` – int – Length of filaments to attach to the MTOC, in number of cylinders.

`MTOCBUBBLETYPE` – int – Type of bubble to be the MTOC.

4.1.8 Output formats

The output of M3SYM will be directed to the `OutputFile` specified. The following output can be set. These outputs must be set on different lines, so users should specify a new parameter, on separate lines, for each output value desired. Output files will be explained in more detail in a later section.

`OUTPUTTYPE` – SNAPSHOT, FORCES, STRESSES, BIRTHTIMES, CHEMISTRY – Output type.

4.2 Chemistry input file

The chemistry input file, whose name is specified in the `SystemFile`, contains the chemical configuration of the system, including species and reactions. It is noted that the order in which cross-linker, motor, and branches species are defined in the chemistry input file should match the relevant mechanical parameters, which are defined in the `SystemFile`. The number of species of each type should also match the `SystemFile`'s species type numbers, or a startup error will result.

All species names given must be unique strings, or a startup error will result.

In all species and reaction definitions, `FILAMENTTYPE` is an integer that specifies the type of filament that this filament species belongs to. For example, if there are two filament types defined, the `FILAMENTTYPE` parameter could be 0 or 1. An invalid value of this parameter will result in an error.

4.2.1 Species

Different types of species can be defined as follows:

- A **diffusing species** is defined in the following form:

  `SPECIESDIFFUSING: <NAME> <COPYNUMBER> <DIFFRATE> <RELEASETIME> <QUALIFIER> (<NUMEVENTS>)`

  where `NAME` is any string defining the name of the species, `COPYNUMBER` is the number of molecules of that species in the system, and `DIFFUSIONRATE` is a float value that determines the diffusion rate of this molecule between compartments. `RELEASETIME` specifies when this molecule populates the system in simulation (in seconds).

  The `QUALIFIER` field is used to define the type of reacting species. The options are the following:

  - `REG` : A regular reacting species. Copy numbers are updated typically.
  - `AVG` : An averaging reacting species. The species will use a copy number averaged over a set number of copy number changes (`NUMEVENTS`) for efficiency.

  The `NUMEVENTS` field, denoted in parentheses as optional, only used in the case of defining an averaging reacting species. If using a regular, this should not be included in the file or an error will result.

- A **bulk species**, which is assumed to be spatially homogeneous, is defined in the following form:

```
SPECIESBULK: <NAME> <COPYNUMBER> <RELEASETIME> <QUALIFIER>
```

where `NAME` is any string defining the name of the species, `COPYNUMBER` is the number of molecules of that species in the system, and `RELEASETIME` specifies when this molecule populates the system in simulation (in seconds).

The `QUALIFIER` field is used to define the type of reacting species. The options are the following:

- `REG` : A regular reacting species. Copy numbers are updated typically.
- `CONST` : An constant reacting species. The species will never change copy number upon reacting.

- Any **filament-related species** can be defined in the following form:

```
SPECIES<SPECIESTYPE>:  <NAME> <FILAMENTTYPE>
```

where `SPECIESTYPE` can be:

- `FILAMENT` : A filamentous species. At least one filament species must be defined if using filaments in simulation.
- `PLUSEND` : A plus end species on a filament, which is defined as the front of the filament. There must be at least one plus end species for every filament species defined in the system.
- `MINUSEND` : A minus end species on a filament, which is defined as the back of the filament. There must be at least one minus end species for every filament species defined in the system.
- `BOUND` : A bound species on a filament. There must be at least one bound species defined for each filament type.
- `LINKER` : A cross-linker species. The ordering of cross-linker initializations should match their mechanical parameters, as stated above.
- `MOTOR` : A myosin II motor species. The ordering of motor initializations should match their mechanical parameters, as stated above.
- `BRANCHER` : A branching species. The ordering of branches initializations should match their mechanical parameters, as stated above.

4.2.2 Binding sites

For every species that binds to filaments (linkers, motors, branchers), a binding site species must be set. This binding site must be a bound species on any filament type. It is declared

in the following form:

```
LINKERBINDINGSITE: <NAME> <FILAMENTTYPE>
MOTORBINDINGSITE: <NAME> <FILAMENTTYPE>
BRANCHERBINDINGSITE: <NAME> <FILAMENTTYPE>
```

where `NAME` is the name of a pre-defined bound species on a filament of type `FILAMENTTYPE`.

4.2.3 Reactions

Reaction definitions must follow these common rules:

- Species that are defined in reactions must be previously defined in the chemistry file.
- For filament-related reactions, most species type and ordering parameters are fixed; if they are fixed, they will be pre-defined in the reaction definition below. If the ordering is not properly followed, a startup error will result.
- All species declarations in a reaction must be separated by white space, with `+` markers between reactants and products. A `->` must be placed between reactants and products, separated by whitespace. If this syntax is not followed, a startup error will result.

Different types of reactions can be defined as follows:

- A **general reaction** between any bulk or diffusing species can be defined in the following form:

  ```
  GENREACTION:
  <NAME>:BULK/DIFFUSING + <NAME>:BULK/DIFFUSING + ...  ->
  <NAME>:BULK/DIFFUSING + <NAME>:BULK/DIFFUSING + ...  <RATE>
  ```

  where any bulk or diffusing species can be included, and `<RATE>` is a float value that determines the rate constant of the reaction.

- A **bulk reaction** between bulk species only can be defined in the following form:

  ```
  BULKREACTION:
  <NAME>:BULK + <NAME>:BULK + ...  ->
  <NAME>:BULK + <NAME>:BULK + ...  <RATE>
  ```

  where any bulk species can be included. If the reaction only contains bulk species, it must be specified as a bulk reaction. `<RATE>` is a float value that determines the rate constant of the reaction.

- A **polymerization reaction** can be defined in the following form:

  ```
  POLYMERIZATIONREACTION: <FILAMENTTYPE>
  <NAME>:BULK/DIFFUSING + <NAME>:PLUSEND/MINUSEND ->
  <NAME>:FILAMENT + <NAME>:PLUSEND/MINUSEND <RATE>
  ```

  where `<NAME>` is the string name of the species, and `<RATE>` is a float value that determines the rate constant of the reaction. It is noted that the first species listed can be either `DIFFUSING` or `BULK`, and the reaction can contain a `PLUSEND` or `MINUSEND`.

  This reaction will polymerize the filament, producing a new chemical species on the end of the filament and increasing the length of the filament by a single monomer.

- A **depolymerization reaction** can be defined in the following form:

  ```
  DEPOLYMERIZATIONREACTION: <FILAMENTTYPE>
  <NAME>:FILAMENT + <NAME>:PLUSEND/MINUSEND ->
  <NAME>:BULK/DIFFUSING + <NAME>:PLUSEND/MINUSEND <RATE>
  ```

  where `<NAME>` is the string name of the species, and `<RATE>` is a float value that determines the rate constant of the reaction. It is noted that the third species listed can be either `DIFFUSING` or `BULK`, and the reaction can contain a `PLUSEND` or `MINUSEND`.

  This reaction will depolymerize the filament, removing a chemical species from the end of the filament and decreasing the length of the filament by a single monomer.

- A **cross-linker reaction** between two filaments can be defined in the following form:

  ```
  LINKERREACTION: <FILAMENTTYPE>
  <NAME>:BOUND:1 + <NAME>:BOUND:2 + <NAME>:BULK/DIFFUSING <->
  <NAME>:LINKER:1 + <NAME>:LINKER:2 <ONRATE> <OFFRATE> <RMIN> <RMAX>
  ```

  where `<NAME>` is the string name of the species, and `<ONRATE>` and `<OFFRATE>` are float values that determines the rate constant of the binding and unbinding reactions. `<RMIN>` and `<RMAX>` are the range of the chemical reaction, and this can be set depending on the structure of the simulated cross-linker. It is noted that the third species listed can be either `DIFFUSING` or `BULK`. The bound species listed must be the corresponding cross-linker binding site.

  This reaction produces cross-linker species at two separate positions on each respective filament which are chemically and mechanically connected. If mechanical force fields are defined for the cross-linkers, a potential will be created between the filaments. The unbinding reaction will remove these species from the filaments, as well as remove any linker potentials that have been created between the filaments.

- A **motor reaction** between two filaments can be defined in the following form:

  ```
  MOTORREACTION: <FILAMENTTYPE>
  <NAME>:BOUND:1 + <NAME>:BOUND:2 + <NAME>:BULK/DIFFUSING <->
  <NAME>:MOTOR:1 + <NAME>:MOTOR:2 <ONRATE> <OFFRATE> <RMIN> <RMAX>
  ```

  where `<NAME>` is the string name of the species, and `<ONRATE>` and `<OFFRATE>` are float values that determines the rate constant of the binding and unbinding reactions. `<RMIN>` and `<RMAX>` are the range of the chemical reaction, and this can be set depending on the structure of the simulated motor. It is noted that the third species listed can be either `DIFFUSING` or `BULK`. The bound species listed must be the corresponding motor binding site.

  This binding reaction produces motor species at two separate positions on each respective filament which are chemically and mechanically connected. If mechanical force fields are defined for the motor, a potential will be created between the filaments. The unbinding reaction will remove these species from the filaments, as well as remove any motor potentials that have been created between the filaments.

- A **motor walking reaction** can be defined in the following form:

  ```
  MOTORWALKINGREACTION: <FILAMENTTYPE>
  <NAME>:MOTOR:N/N+1 + <NAME>:BOUND:N/N+1 ->
  <NAME>:MOTOR:N/N+1 + <NAME>:BOUND:N/N+1 <RATE>
  ```

  where `<NAME>` is the string name of the species, and `<RATE>` is a float value that determines the rate constant of the reaction. The choice of `N/N+1` will determine whether the motor is stepping forward or backward. A motor movement from `N` to `N+1` is defined as forward movement (towards the plus end of the filament), and the opposite is backward (towards the minus end). These choices for the reactants and products must be self-consistent as well as consistent with the bound species positions chosen in the reaction, or a startup error will result. The bound species listed must be the corresponding motor binding site.

  This reaction will move a motor head in the given direction.

- A **branching reaction** can be defined in the following form:

  ```
  BRANCHINGREACTION: <FILAMENTTYPE>
  <NAME>:BULK/DIFFUSING + <NAME>:BULK/DIFFUSING + <NAME>:BOUND <->
  <NAME>:BRANCHER + <NAME>:PLUSEND <RATE> <NUCLEATIONZONE> <NUCLEATIONDIST>
  ```

  where `<NAME>` is the string name of the species, and `<RATE>` is a float value that

determines the rate constant of the reaction. It is noted that the first and second species listed can be either `DIFFUSING` or `BULK`. The bound species listed must be the corresponding branching binding site.

The `NUCLEATIONZONE` and `NUCLEATIONDIST` specify the volume in which a branching reaction can occur. The choices for the zone parameter are the following

- `ALL` : A new filament can nucleate anywhere in the simulation volume due to branching.
- `BOUNDARY` : A new filament can nucleate a given distance away from a boundary due to branching, specified by the `NUCLEATIONDIST` from the system boundary.
- `TOPBOUNDARY` : Similar to `BOUNDARY` except only in the top half of the volume (in the z direction).

It is noted that `NUCLEATIONDIST` needs to be specified for all nucleation zones, but it is unused for `ALL`.

This reaction will create a new branching point, as well as a filament with the desired chemical plus end. If mechanical force fields are defined for the branching point, a potential will be created between the parent and child filament. The unbinding reaction will remove the branching point from the filaments, thus freeing the child filament from the parent. It will also remove any branching point potentials that have been created between the filaments.

- A **nucleation reaction** can be defined in the following form:

```
NUCLEATIONREACTION: <FILAMENTTYPE>
<NAME>:BULK/DIFFUSING + <NAME>:BULK/DIFFUSING ->
<NAME>:PLUSEND + <NAME>:FILAMENT + <NAME>:MINUSEND <RATE>
```

where `<NAME>` is the string name of the species, and `<RATE>` is a float value that determines the rate constant of the reaction. It is noted that the first and second species listed can be either `DIFFUSING` or `BULK`.

This reaction will create a new filament with the given chemical plus end, minus end, and filament species.

- A **destruction reaction** can be defined in the following form:

```
DESTRUCTIONREACTION: <FILAMENTTYPE>
<NAME>:PLUSEND + <NAME>:MINUSEND ->
<NAME>:BULK/DIFFUSING + <NAME>:BULK/DIFFUSING <RATE>
```

where <NAME> is the string name of the species, and <RATE> is a float value that determines the rate constant of the reaction. It is noted that the third and fourth species listed can be either DIFFUSING or BULK.

This reaction will destroy a filament, removing it from the system.

- An **filament aging reaction** can be defined in the following form:

```
AGINGREACTION: <FILAMENTTYPE>
<NAME>:FILAMENT/PLUSEND/MINUSEND ->
<NAME>:FILAMENT/PLUSEND/MINUSEND <RATE>
```

where <NAME> is the string name of the species, and <RATE> is a float value that determines the rate constant of the reaction. Either of the reactant or product species can be FILAMENT, PLUSEND, or MINUEND, but the product and reactant species must be the same type, or a startup error will result.

This reaction will change the chemical species that resides in a filament.

- A **filament severing reaction** can be defined in the following form:

```
SEVERINGREACTION: <FILAMENTTYPE>
AT <NAME>:FILAMENT <RATE>
```

where <NAME> is the string name of the species, and <RATE> is a float value that determines the rate constant of the reaction.

This reaction will sever the filament at the closest cylinder connection to a given chemical position, producing two child filaments.

4.3 Filament input file

The filament input file, whose name is specified in the SystemFile, contains the type and coordinates of filaments to initialize in the system at startup. The format of an initial filament declaration is as follows:

```
FILAMENT: <FILAMENTTYPE> coord1x coord1y coord1z coord2x coord2y coord2z
```

where {coord1x, coord1y, coord1z} and {coord2x, coord2y, coord2z} specify the starting and ending coordinates of the filament.

4.4 Bubble input file

The bubble input file, whose name is specified in the SystemFile, contains a coordinate of bubbles to initialize in the system at startup. The format of an initial filament declaration

is as follows:

```
BUBBLE: <BUBBLETYPE> coordx coordy coordz
```

where {`coordx, coordy, coordz`} is the coordinate of the bubble.

5 Output

M3SYM can produce a number of output types, set in the `SystemFile`, produced at a snapshot frequency also defined in the `SystemFile`. These output files will be placed in the `OutputDirectory` specified at runtime. The output types and visualization of this output are described below.

5.1 Types of output files

5.1.1 snapshot.traj

The snapshot file gives the basic trajectory information of the system. It includes a brief description for all filaments, cross-linkers, motors, and branching points in the system, as well as information on the current chemical step. It is produced with the following form:

```
chemstepnumber time numfilaments numlinkers nummotors numbranchers
F filamentid filamenttype filamentcyllength deltal deltar
beadcoord1x beadcoord1y beadcoord1z beadcoord2x beadcoord2y beadcoord2z ...
...
L linkerid linkertype
startcoordx startcoordy startcoordz endcoordx endcoordy endcoordz
...
M motorid motortype
startcoordx startcoordy startcoordz endcoordx endcoordy endcoordz
...
B brancherid branchertype
coordx coordy coordz
```

5.1.2 forces.traj, stresses.traj, and birthtimes.traj

The forces file gives the forces on each element in the system, in similar form to the snapshot file. It is produced with the following format:

```
chemstepnumber time numfilaments numlinkers nummotors numbranchers
F filamentid filamenttype filamentcyllength deltal deltar
```

```
bead1property bead2property ...
...
L linkerid linkertype
linkerproperty
...
M motorid motortype
motorproperty
...
B brancherid branchertype
*no property printed for branching points*
...
```

where the properties are as follows:

- forces.traj: the magnitude forces on each cylinder, as well as the magnitude of stretching force on each cross-linker and motor are printed.
- stresses.traj: the stretching stress on cylinders, cross-linkers, and motors are printed.
- birthtimes.traj: the birth time of on cylinders, cross-linkers, and motors are printed.

5.1.3 chemistry.traj

The chemistry trajectory file gives the copy numbers of all species in the system, along with the current chemical step and time. It is produced with the following form:

```
chemstepnumber time
SPECIESNAME COPYNUMBER
```

where `SPECIESNAME` represents the name of the system species and `COPYNUMBER` is the current copy number of that species at the given timestep.

5.2 Visualization of output

The output described in the previous section can be visualized using a python script found in `InstallDirectory/visual`, named `AnimateTrajectory.py`. This script uses MayaVi (http://mayavi.sourceforge.net) to produce a visualization of trajectory frames, as well as an animation of an entire simulation.

5.2.1 Installation of MayaVi

The following python-related dependencies for MayaVi should be installed:

- vtk5 5.10.1

- qt4 4.8.6
- ipython 2.2.0
- matplotlib 1.4.0
- pyside 1.2.2

Most of these packages are available through MacPorts or Homebrew, if using an Apple computer. The following environment variables may need to be declared, depending on your system configuration:

```
export QT_API = pyside (or qt4)
```

A helpful alias to run ipython is:

```
alias ip = "ipython --gui = qt --pylab = qt"
```

We will assume for the next section that this alias is configured.

5.2.2 Running the visualization script, AnimateTrajectory.py

The visualization script must be edited by the user on lines 6-8 to include the desired snapshot file. A color file can also be specified, which will color the network based on either a force, stress, or birth time file. If no color file is included, the elements of the network will be colored to a default value. The script can also be edited on lines 231-267 to include titles, boundaries, scales, default colors, and choice of color map.

Run the script using the following commands:

```
> ip
> run -i AnimateTrajectory.py
```

This will load all snapshots of the trajectory files specified. To show a snapshot, execute the following commands:

```
> show_snapshot(snapshot number)
```

To show the entire simulation frame by frame, execute the following:

```
> anim()
```

6 Developer guide

6.1 Pull requests

Pull requests for the M3SYM repository should be directed to James Komianos (jkomianos@gmail.com). From there, detailed instructions on cloning the repository, and general guidelines on adding features will be given.

6.2 Coding style

M3SYM follows standard style guides and object-oriented design principles. For details on general C++ coding style, see the Google C++ style guide at http://google-styleguide.googlecode.com/svn/trunk/cppguide.html. In general, the following best practices should be followed:

- Use camelCase for all declarations, with classes beginning with a capital letter
- Use proper tabs for all declarations, loops, control flow, etc.
- Keep code under 80 characters per line

For documentation, please use Doxygen-style commenting. See the Doxygen user guide at http://www.stack.nl/ dimitri/doxygen/ for more information.

6.3 Other information

Please direct all code inquiries to James Komianos (jkomianos@gmail.com).

7 References

[1] Popov K, Komianos J, and GA Papoian. "MEDYAN: Mechanochemical Simulations of Actomyosin Networks and Other Active Matter." PLoS Computational Biology (in review), 2015.