Documentation for `C++` model

Matt McDermott

July 5, 2014

1 Parameter Values:

`Tau`$=\tau$**:** τ measures the timestep for the model and is currently $\frac{1}{4000}$ minutes.

`Contact Length`**:** This measures the length of time that an MT attached to the Cortex remains attached. It is currently `400*Tau`$=\frac{1}{10}$ minutes.

`Vg`$=V_g$**:** This measures the growth velocity of an MT and is currently $40\mu m/\text{min}$.

`Vs`$=V_s$**:** This measures the shrinking velocity of an MT and is currently $120\mu m/\text{min}$.

`Vs_c`$=V_{sc}$**:** This measures the shrinking velocity of an MT post contact and is currently $120\mu m/\text{min}$.

`kc`$=k_c$**:** This measures the catastrophe frequency of an MT and is currently $.1 \cdot 601/\text{min}$.

`kr`$=k_r$**:** This measures the catastrophe frequency of an MT and is currently $.4 \cdot 601/\text{min}$.

`R1_max`$=R_1$**:** This measures the maximum length of the ellipsoidal cell body along the x-axis and is currently $50\mu m$.

`R2_max`$=R_2$**:** This measures the maximum length of the ellipsoidal cell body along the y-axis and is currently $30\mu m$.

`Prad`$=P_{\mathbf{rad}}$**:** This measures the radius of the pronucleus and is currently $5\mu m$.

`Eta`$=\eta$**:** This measures the translational drag coefficient of the pronucleus and is currently $1\frac{\text{pN}}{\mu m \text{min}}$.

`Mu`$=\mu$**:** This measures the translational drag coefficient of the pronucleus and is currently $5\eta = 5\frac{\text{pN}}{\mu m \text{min}}$.

2 Usage Instructions:

The three basic steps to to use this code are:
To Compile: Run `make` on the command line.
To Run: After compilation, run

```
./mtKineticModel <Number of Runs> <Output File> [temp/all]
```

on the command line. Here, `<Number of Runs>` denotes the number of runs of the given system the model should produce and `<Output File>` specifies the file to which the results should be written. Note that this file will be placed in the directory `nuclearKineticsModelling/data`. The final parameter, `[temp/all]`, is an optional parameter. If omitted, the system will only store the final configuration of the system in the result file. If given as `temp`, it will store 30 second time slices in the result system for each model run, in sequence (i.e. it will list various parameters for 30 second slices for the first run, then those for the second, and so on and so forth). If given as `all`, it will write data for every timestep in the result file, in sequence, for each model run.
To View: There are a variety of matlab scripts in the `plottingCode` folder to visualize the results. Of particular note are the histogram scripts, `positionRotationHist.m` and `positionRotationHistCompare.m`, the single-run visualization tool, `readData.m`, and the movie-frame making scripts `movieMaker*.m`. All of these scripts are run with the following basic premise: Open `MATLAB`, set values `dataDir` and `dataFile` *(and/or `dataFile2`)* in the `MATLAB` console, then run the script. `dataDir` stands for the directory (as a sub-directory of `'nuclearKineticsModelling/data/`) and `dataFile` stands for the dataFile in question.

2.1 Basic Usage:

For basic usage, the only additional step beyond simple compilation and running is parameter modification. Here is a short description of some of the parameters we change most frequently *(All found in the `parameters.cpp` file)*.

`springsOn` This boolean value controls whether the centrosomes are connected to the pronucleus rigidly (and thus not allowed to translate relative to the pronucleus) or via springs (of spring constant given later in the file).

`translation` This boolean value controls whether the pronucleus is allowed to translate. If it is `true`, the system proceeds in full generality. If it is `false`, the pronucleus center is fixed in the cortex (though it is allowed to rotate). If the springs are on, the centrosomes are also allowed to translate (and thus provide force via the springs to the pronucleus).

`kM/kD` These parameters control the relative spring constants for the springs connecting the mother (`M`) centrosome and daughter (`D`) centrosome to the pronucleus, respectively. If springs are off, then these parameters do nothing. We do not yet have good values for these constants.

`startX` This parameter controls the starting x coordinate of the pronucleus, with 0 being the center of the cell. We typically test starting at 0 and starting at 7.5μm.

`startPsi` This parameter controls the starting angular position of the pronucleus, in radians, with 0 being *horizontal*. For on-angle runs, we use $\frac{\pi}{2}$. To test the effects of angular variations, we typically use $\frac{\pi}{2} + k\frac{\pi}{8}$, with $1 \leq k \leq 4$, though it might be sensible to test larger k and finer gradations.

`numRegions/regionAngles/regionProbabilities` These are the band parameters, and control the number of uniform probability regions, the angular endpoints of those regions, and the associated probabilities with those regions.

2.2 Advanced Usage:

There are a variety of other parameters one may wish to modify, all of which found in the `parameters.cpp` file. For these modifications, the same procedure is used: modify the parameter, re-compile, and run.

3 Current Algorithm

3.1 High Level Description

The current algorithm is very straightforward. In particular, for each step in time, stepping by a set parameter `tau` through `Duration` minutes, the algorithm computes the net force and torque on the pronucleus and then uses an eulerian approximation of the solution to the DE inspired by friction dominated domain conditions to update the position and angle of rotation of the body. The algorithm also maintains a list of endpoints of a set number of MTs, updating these through growth, shortening, and respawning when necessary. It also uses the same risk formulas for catastrophe and rescue as did the previous matlab model. The only real differences are that several bugs are corrected, force and torque are calculated upon every iteration directly, and the implementation outputs final data to a file for another program to read in. It is worth noting that the current implementation has a separate, abstracted Vector class to handle vector addition, scalar multiplication, taking projections and finding the norm. This makes the code more readable, but also enables a much easier switch from 2D to 3D. In fact, the Vector class itself is dimension independent (as in, it is set `Vector.hpp` but can be changed without consequence) so moving to a 3D model really only amounts to modifying the cortex positioning code.

3.2 Implementation Details

3.2.1 MT Envelope Restriction:

In this code, there is a parameter which specifies the allowed envelope of growth for MTs from either the mother MTOC or the daughter MTOC (separately). This is accessed via the variable `envWidthM(D)` and `envelopeM(D)`. For standard envelopes, simply setting the width should be sufficient (as this actually is used in the `envelope` parameter) but for stranger cases, it is also possible to just modify the `envelope` parameter directly and set the envelope that way. We have tried and discussed a number of ways to use this parameter, but have finally settled upon the idea that the envelope merely limits growth. In particular, a new MT can only be spawned within the allowed envelope (meaning spawned at an angle contained in the range specified by the `envelope` parameter, with 0 degrees being the angle of rotation of the MTOC anchor point relative to the pronucleus) but an MT that falls outside of this envelope as the pronucleus rotates is allowed to continue to grow, connect, and catastrophe as normal. If this needs to be changed, see line 604 of `main.cpp`, near the comment beginning with `MT-ENV`, which is just before the length-based respawning code.

3.2.2 Spring Implementations:

Currently there is built in spring capabilities that can be activated with or without pushing forces on either MTOC separately that do account for potential contact and resulting force transfer with the pronucleus. The springs are turned on or off in the two boolean parameters `motherSpringOn` and `daughterSpringOn` in the file `parameters.cpp`. If a spring is on, then the motion of that MTOC is governed by three things: the forces from the associated MTs, the spring force between the MTOC and its anchor point (using current displacement), and a random displacement term. Note that this is altered slightly if the MTOC is close enough to the pronucleus. In that case, the spring force directed towards the pronucleus is reflected by the pronucleus, inducing a force on the MTOC away from the pronucleus and on the pronucleus away from the MTOC. Note, however, that this **does not** imply that the MTOC experiences a *net* force away from the pronucleus. This is not a reflection collision event; rather, the force from the MTOC towards the pronucleus is effectively just *transferred* to the pronucleus. The force induced by the pronucleus on the MTOC is completely balanced out by the initial force towards the pronucleus on the MTOC in the first place, resulting in only net tangential forces on the MTOC after contact. Similarly, in this case, the random displacement is only allowed to occur tangentially. Note that when the spring is enabled, there is no *direct* action on the pronucleus by the MTs. Rather, the MTs induce a force on the MTOC, which moves, thus inducing a spring force on the pronucleus (and itself, in opposite magnitude) the next timestep. Thus, the pronucleus is only ever directly moved by the spring with this model.

3.2.3 Boundary Checking:

The current system checks for a cortex pronucleus collision and a MTOC pronucleus collision using slightly different methods. For the MTOC pronucleus collision, the system simply determines if the MTOC is within `Prad + 0.1` of the pronucleus position, using simple vector arithmetic. For the pronucleus cortex collision, the system must do something slightly stranger. Here, the system determines if the center of the pronucleus has impacted a slightly smaller ellipse than the cortex. This doesn't give exactly the desired behavior, as the curvature makes this slightly inaccurate, but in practice it is quite effective.

3.2.4 MT Connection Density Limitation:

The system maintains a collection of windows, within each is only allowed one contact. The contact windows are computed in mathematica, and stored in the `parameters.cpp` file as plain values. There is another, finer set of windows which is commented out in this file as well.

3.2.5 Band Specification:

The band specification is very straightforward. A list of fixed cortex angles are stored, beginning with 0 and ending with 2π. These specify the endpoints of all "cortex pieces," which are angular, uniform sectors of the cortex. This is stored in `regionAngles`. The probability of contact in these areas and force multipliers scaling contacts in these areas are stored in `regionProbabilities` and `regionForceMultipliers`, respectively. These are each one component shorter than `regionAngles`, as index `i` of the probability and multiplier vectors describe the effect on the region between angles `regionAngles[i]` and `regionAngles[i+1]`.

3.2.6 Other Miscellaneous Notes:

- It is important to the current implementation that `mt_numb` is a constant value, as otherwise fixed-size arrays could not be used to store all the codes data.
- The `parameters.cpp` file defines not only fixed constants for the program, but also initializes global variables, a few import `typedef`s, and an `enum` that defines the centrosome options. The first typedef sets up a new `float_T` type, which is used so that precision of the model can be altered painlessly, and the second wraps the separate `Vector` class type with a `vec_T` type, which is actually not needed anymore and could be removed, so long as all the code was updated to use `Vector` instead of `vec_T`.
- The two `operator` functions at the top of `main.cpp` are merely to aid in file output.

- The net force calc function is simple; It merely sums up the all the force directions for the MTs, then multiplies that sum by the force for any given MT. This will give the net force on that centrosome.
- The net force function is just a wrapper around net force calc.
- The `updatePNPos()` function is an extraction of the code to update the position of the pronucleus from the main loop. The actual mechanics of the function are very straightforward currently, and simply translate the components of the force into torque and motion inspiring forces, respectively.
- `advanceMT` is simply an encapsulation of some messy code from the loop. It grows or shrinks MTs.
- `respawnMT` creates a new MT.
- The main loop appropriately orders these functions, updates contact statistics, growth statistics, and growth velocities. It also checks for rescue and catastrophe and writes the data to the file.
- Note here that a major implementation detail between this algorithm and the matlab one is that as this algorithm computes the force on every iteration, not just on a new contact, the `MT_touch` array must be active every second, which in particular means that it cannot be used to distinguish between shortening velocities as it is in the matlab model. So, instead, an additional array is maintained, `MT_GrowthVel_*`, where * is either M or D. This stores the current growth velocity of the given MT.

4 Future Work

1. Add generic, extendable cortex positioning code. This would ensure a smooth change from 2D to 3D. Test and optimize the code. No performance testing or benchmarking has been done to date.
2. Test Stability