Documentation for mean-field model (Matlab)

Erin Angelini

May 29, 2018

1 Overview of functions

The following Matlab functions constitute the mean-field model of mitotic spindle rotation in *C. elegans* embryos.

`Main.m`: This is the function that implements the full model. That is, it calculates the potential energy $W(\alpha)$ at angles α from 0 and π. See §1.1 for more information about the step size of α. This function takes inputs on both the cell geometry (`a` and `b`) and the setup of the spindle (`ctr`, `phi_1`, `phi_2`, `push`, `AP`, `LD`, `elas`; see §1.1). The output of `Main.m` is the energy landscape `W` over the range of angles `A` $= [0, \pi]$ in vector form. It also plots `W` against `A`.

`MainWithFilter.m`: This function is the exact same as `Main.m`, except that it puts the output `W` through a high-pass filter (to clean up the curve) before plotting. Note that this file is **only useful for the symmetric well case**: the high-pass filter loses all information about asymmetry in the wells.

`parameters.m`: This script contains all the fixed parameters in the model (see §1.1). It is called by `Main.m` at the beginning of the file; it is also called by `work.m`, `torque.m`, and `find_prob.m`. It does not take any inputs and its output is to load the fixed parameters onto the workspace.

`alpha_halfplane.m`: This function is called by `Main.m` before calculating `W`. Its inputs are `a`, `r_N`, `ctr`, `phi_1`, and `phi_2` (§1.1). The purpose of this function is to calculate the angles at which the leading and lagging ends of each MT aster are above or below the x-axis (for ease of future calculations in `torque.m`). It saves these angles to two .mat files, `halfplane1.mat` and `halfplane2.mat`, which are later loaded by `torque.m`.

`equal_arcsCircle.m`: This function is called by `Main.m` in the case that the cell is perfectly circular (i.e. aspect ratio = 1 or `a` = `b`). It calculates the locations of the cortical dyneins in terms of the parameter t that parameterizes the boundary of the cell. The output of this function is a vector of these positions.

`equal_arcsHannah.m`: This function is called by `Main.m` in the case that the cell is perfectly circular (i.e. aspect ratio $\neq 1$ or `a` $\neq$ `b`). It calculates the locations of the cortical dyneins in terms of the parameter t that parameterizes the boundary of the cell. The output of this function is a vector of these positions.

`let99.m`: This function calculates the beginning and end positions of the Let-99 push band in the case that the band is present (i.e. `push = 'on'`, see §1.1). Again, these positions are in terms of the parameter t that parameterizes the boundary of the cell. Given the cell resolution (`a` and `b`) and a fixed arc-length `arc`, it outputs a vector `tvec` that contains the beginning and end positions of the push band. In the case that there is no push band present (i.e. `push = 'off'`, see §1.1), `tvec` is an empty vector.

`work.m`: This is the function called by `Main.m` that actually calculates the energy landscape. It has the same inputs as `Main.m` plus an additional input `alpha`, which is the current angular orientation of the spindle. It calculates the potential energy of the spindle at `alpha` by approximating the integral of the torque function from angle 0 to angle `alpha`. This integral is approximated numerically by a Riemann sum of step size `d_alpha` (see §1.1) that calls `torque.m`.

`torque.m`: This function, which has all the same inputs as `work.m`, calculates the total torque on the spindle at a given angular orientation `alpha`; this value is its output. The following seven functions (`find_env_pi.m` to `find_pull.m`) are the helper functions that allow it to do so.

`find_env_pi.m`: This function finds the beginning and end positions of the MT aster along the cell boundary (in terms of the boundary parameter t) given that the aster's angular spread `phi_i` is equal to π. Its inputs are `a`, `b`, `r_N`, and `ctr`, and its output are the beginning and end points of the aster. It finds these points by selecting the correct roots from symbolic solutions for the intersection of the line defined by the aster spread and the cell boundary (Figure 1). This function is called separately for each of the upper and lower (or "mother" and "daughter") asters.

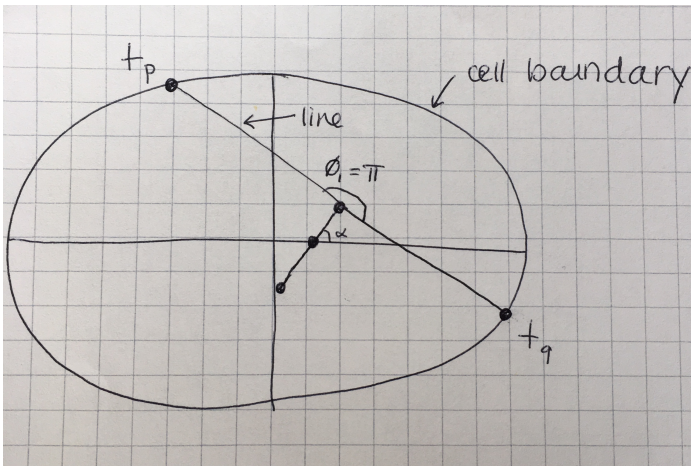

Figure 1: Geometric setup for the function `find_env_pi.m`.

`find_env_notpi.m`: This function finds the beginning and end positions of the MT aster along the cell boundary (in terms of the boundary parameter t) given that the aster's angular spread `phi_i` is less than π. It takes the same inputs as `find_env_pi.m` plus the aster spread `phi_i`. It also takes the angles from `alpha_hlafplane` as inputs. It makes calls to `find_env_pi.m` and `find_end.m`, and uses the following geometric setup to compute the beginning and end of the MT aster with correct spread `phi_i` (Figure 2). This function is also called separately for each of the upper and lower (or "mother" and "daughter") asters.

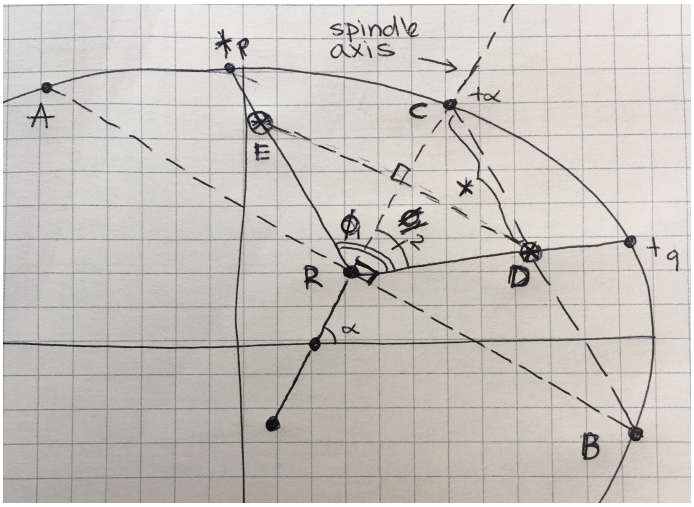

Figure 2: Geometric setup for the function `find_env_notpi.m`.

`find_end.m`: This function finds the point `D` in the above configuration (Figure 2). It does so by finding the line segment with length `x` (found by the law of sines as we know |`CR`|, |`CB`|, ∠`RCD`, and ∠`DCR` = `phi_i/2`) in the direction of the line segment `CB` (Figure 2). Call this segment `CD` (Figure 2). Using geometric properties of vector addition, we obtain the point (i.e. vector) `D` by adding the vector `C` to the vector `CD` (Figure 2). This point `D` is crucial for finding the lagging endpoint of the MT aster because it allows the function `find_env_notpi.m` to define a line from the MTOC (point `R`) to the cell boundary that makes angle `phi_i/2` with the spindle axis (Figure 2). The function `find_env_notpi.m` then reflects `D` across the spindle axis to the point `E`, and similarly finds the leading endpoint of the aster (Figure 2).

`find_dyneins.m`: Given the endpoints `t_p` and `t_q` of the MT aster (along with the vector of all dynein positions `set_psi` and some other inputs), this function finds the dynein positions that are within the MT aster. It stores these dyneins in the output vector `env_d` It also takes the Let-99 band positions `tvec`, and places the dynein positions in the Let-99 band that overlap with the aster in a separate vector `let` (so these positions are NOT included in `env_d`). This output vector now contains the positions along the boundary in the Let-99 band where MTs will grow up against the boundary (i.e. no dyneins) and create pushing forces.

`find_sintheta.m`: This function finds the sine of the angle `theta_i` made by each MT in the aster and the spindle axis. We need the sines of these angles in order to calculate the torque on the spindle center (i.e. the pronucleus). This function is called separately for each MT aster, as well as for the MTs making contact with dyneins (`env_d`) versus those in the push band (`let`). This function also implements MT sliding and elasticity. If this feature is present (i.e. `elas = 'on'`), then this function perturbs each angle `theta_i` with a random angle `omega` (uniformly sampled from the angle between the two adjacent dynein locations) before calculating the sines.

`find_prob.m`: This function is for implementing the anterior-to-posterior difference in MT-dynein binding probabilities (which itself is for scaling down the anterior forces). If this difference is present (i.e. `AP = 'on'`), it loops through the vector of dynein positions `env_d` and builds a vector `myProbs` of corresponding binding probabilities depending on if the dynein is in the anterior region or the posterior region. The anterior-to-posterior split is defined at the 60:40 egg-length mark (`0.2*a`), and should be kept as such unless we get new experimental findings. Similarly, we want to fix the posterior and anterior binding probabilities (`P_p` and `P_a`, respectively) fixed, as these are taken from the original study by Coffman *et al.* Otherwise, if there is no anterior-to-posterior difference in place (i.e. `AP = 'off'`), then the vector of probabilities contains all ones. This function is called separately for each MT aster.

`find_pull.m`: This function is for implementing length dependent pulling forces. If length dependence is in place (i.e. `LD = 'on'`), then it loops through the vector of dynein positions `env_d` and builds a vector `myForces` of corresponding pulling forces. If not (i.e. `LD = 'off'`), then the vector of pulling forces contains all ones (i.e. pulling force is constant and equal to 1).

`ellipse.m`: This function is not used in calculating the energy landscape, but is used for plotting purposes. Specifically, it plots the boundary of the cell given the dimensions `a` and `b`. There is a commented out section of code in `torque.m` that calls this function as part of a plot of the complete spindle setup (dynein locations, MT asters, etc.). This function is called separately for each MT aster.

`PDFplotCTRL.m`: Given the ellipse major axis `a`, this function plots the steady state p.d.f. $p(\alpha)$ corresponding to $W(\alpha)$, for the case that `b = 15` and the asters are symmetric (control case, i.e. no `push` no `AP`, no `LD`, no `elas`). It does so by loading the .mat file of scale factors (`k = k_1*Wmax`, where `Wmax` is the maximum value of `W`, and `k_1` is a pre-fit parameter; see §2) and plotting the function $p(\alpha) = Ne^{-\mathtt{k}W(\alpha)}$ (N is a normalization constant).

1.1 Inputs & parameters

`a`: Ellipse/cell major axis (i.e. half-length of x-axis)

`b`: Ellipse/cell minor axis (i.e. half-length of y-axis)

`ctr`: Spindle-center position x-coordinate

`phi_1`: Upper MT aster spread, between 0 and π

`phi_2`: Lower MT aster spread, between 0 and π

`push`: String (`'on'` or `'off'`), for push band

`AP`: String (`'on'` or `'off'`), for anterior-to-posterior difference

`LD`: String (`'on'` or `'off'`), for length-dependent pulling forces

`elas`: String (`'on'` or `'off'`), for MT slipping and elasticity

`r_N`: Nuclear/spindle radius

`b_basal`: Set equal to 15 (wild type = 15, where `a` = 25)

`a_basal`: Set equal to 15 (for wild type when elongating y-axis)

`a_WTscale`: Wild type dimension (= 40) for scaling up volume

`b_WTscale`: Wild type dimension (= 24) for scaling up volume

`N_d_basal`: Number of cortical dyneins in control case (= 128)

`arc`: Full arc length of Let-99 band

`d_alpha`: Step size for computing energy landscape

`end_alpha`: Maximum value of `alpha` for energy landscape

`A`: Range of `alpha` values from 0 to `end_alpha` with step size `d_alpha`

`phi_pi`: For when MT aster spread is equal to π (set = π)

`rho_asymm`: Angular MT density for asymmetric envelope (from compt. model)

`rho_symm`: Angular MT density for symmetric envelope (from compt. model)

`myCutoff`: Cutoff (x-coordinate) for 60:40 egg-length mark (= `0.2*a`)

`P_p`: Posterior binding probability (= 1)

`P_a`: Anterior binding probability (= 0.65, from compt. model)

1.2 .mat files

`dyneins16to30.mat`: Contains a cell of vectors, where cell index `a` corresponds to the vector of dynein locations for given input `a` to `Main.m` (`b` is fixed at 15).

`dyneins_b16to35.mat`: Contains a cell of vectors, where cell index `b` corresponds to the vector of dynein locations for given input `b` to `Main.m` (`a` is fixed at 15).

`dyneinsAR1p67VolScaleBy1p6.mat`: Contains two different vectors of dynein locations specific to a cell with `a = 40` and `b = 24`. These dimensions give us wild type aspect ratio (AR = 1.67) but with cell volume scaled up (from the usual `a = 25` and `b = 15`) by a factor of 1.6. The first vector of dynein positions, `scaleDynWithVol`, is for when we scale the number of dyneins with total arc length so that the arc length between dyneins remains a constant 1. The second vector of dynein positions, `fixDynAt128`, is for when we fix the number of dyneins to be 128, which is the same number of dyneins as when `a = 25` and `b = 15`.

`kValsCtrlAR1p06to2.mat`: Contains vector `kvec` of scale factors `k` for `PDFplotCTRL.m`.

2 Fitting the parameter `k_1`

Earlier work with this model included fitting the unknown parameter `k_1`, where

$$k_1 = \frac{1}{\eta\, D}, \qquad \eta = \text{drag coefficient, and } D = \text{diffusion coefficient.}$$

This fitting was done using the file `plotPDF.m`. This function takes as input a vector of final angles `psi` from the computational model and the maximum value of the energy landscape `Wmax`. It calls the function `nhist` (written by Johnathan Lansey, open source) to plot a normalized histogram of final angles. Finally, it fits the steady-state p.d.f,

$$p(\alpha) = e^{-\texttt{k\_1}W(\alpha)},$$

to this histogram using a least-squares method (Matlab's `lsqcurvefit` function) with respect to the unknown parameter `k_1`.