LHON Matlab Package Description

Author: Pooya Merat

Dec 4, 2015

Contents

1 Preliminary Information

All the inputs and outputs of the functions explained below are stored in a single variable which is passed as the first *argument* (or input) except for the `generate_model`. Aside from the parameters it also includes some function, such as plot, create_csg and save, which will be explained below.

"models" folder within the working directory contains data of previously generated, meshed and simulated results, that are saved by user. Each function and its performance is described below.

2 Create Model (`generate_model`)

2.1 Overview

The first function that must be called in order to create the model is `generate_model`. This function first creates randomly placed circles as bundles in a parent circle (eye nerve), and then within each bundle randomly distributes other circles which represent the neurons. The radius minimum and maximum for both bundles and neurons should be set as well as radius of the eye nerve.

The radius of bundles are random numbers between the minimum and maximum range. Radius of neurons are still random numbers within the minimum and maximum limits but the average radii of neurons within each bundles is biased with respect to the location of each bundle, to comply with the results from Pan et. al [2012], Figure 1.

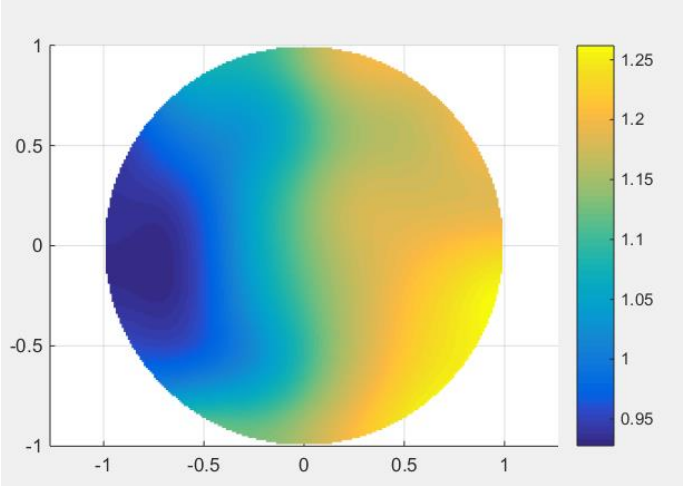

Figure 1: Neuron average radius based on location of bundle, adapted from Pan et. al [2012] generated with: M.plot.avg_r()

Also the distribution of neuron radii is made skewed symmetric in order to resemble the results from Pan et. al [2012]. The skewed distribution parameters can be changed in the `skewed_distr` function in `generate_model` file.

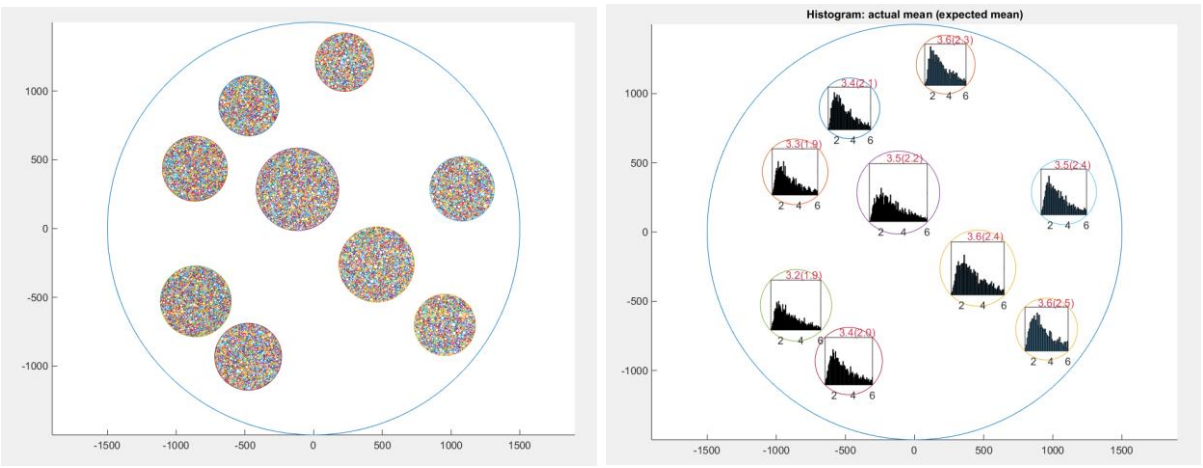

*Figure 2: **Left**: a sample model with 9 bundles (`M.plot.model()`)*
***Right**: Skewed Distribution of neuron radii within each bundle (`M.plot.histogram()`)*
(with a neuron scale of 2)

To decrease the complexity of calculations for the rest of procedure `neuron_scale` variable should be set to scale up the actual size of neurons, otherwise the program will slow down due to large number of points.

2.2 Syntax and User Interface

- `M = generate_model(...)`
  Creates a model using the default settings and outputs to `M`.
- `M = generate_model(..., 'nerve_r', nr,...)`
  Sets the nerve radius where `nr` is the desired nerve radius.
- `M = generate_model(..., 'bundle_r_range', [rm rM], ...)`
  Sets the bundle radius range, where `rm` is the minimum value and `rM` the maximum limit.
- `M = generate_model(..., 'min_bundles_dis', m_dis, ...)`
  Sets the minimum distance/clearance between the bundles and between the bundles and the nerve perimeter, where `m_dis` is the minimum distance.
- `M = generate_model(...,'neuron_r_range', [rm rM], ...)`
  Sets the minimum and maximum values for neuron radius, where `rm` is the minimum value and `rM` the maximum value.
- `M = generate_model(...,'neuron_dens', nds, ...)`
  Sets the neuron density within each bundle, radius, where `nds` is the density. Higher values will increase the density of neurons. Default value is 1. Large numbers may slow down the model generation.
- `M = generate_model(...,'bundle_dens', bds, ...)`
  Sets the density of bundles within the nerve, radius, where `bds` is the density. Higher values will increase the density of neurons. Default value is 1.

Note: If none of the settings above is set, the default values are used which are modifiable at the first section of `generate_model`.

2.2.1 Plot functions

- `M = generate_model(...,'plot_model', ...)`
  Will also plot the model after its generation.
- `M.plot.model()`
  Will plot the model with neurons.
- `M.plot.model('no_neuron')`
  Will only plot the bundles and the nerve without the neurons.
- `M.plot.histogram()`
  Plots the histograms of neuron radii within each bundle, Figure 2 (*Right*).
- `M.plot.avg_r()`
  Plots the expected average for neuron radii inside the bundles, based on location of the bundle, as described in Pan et. al [2012], Figure 1.

2.2.2 Graphical User Interface

- `M = generate_model(...,'GUI', ...)`
  Opens a window for easier setting and viewing the model of bundles before creating the model of neurons within them. The ***Update*** button updates the model after changing the values in the text boxes. Hitting ***Done/Continue*** will proceed with the rest of modeling process.

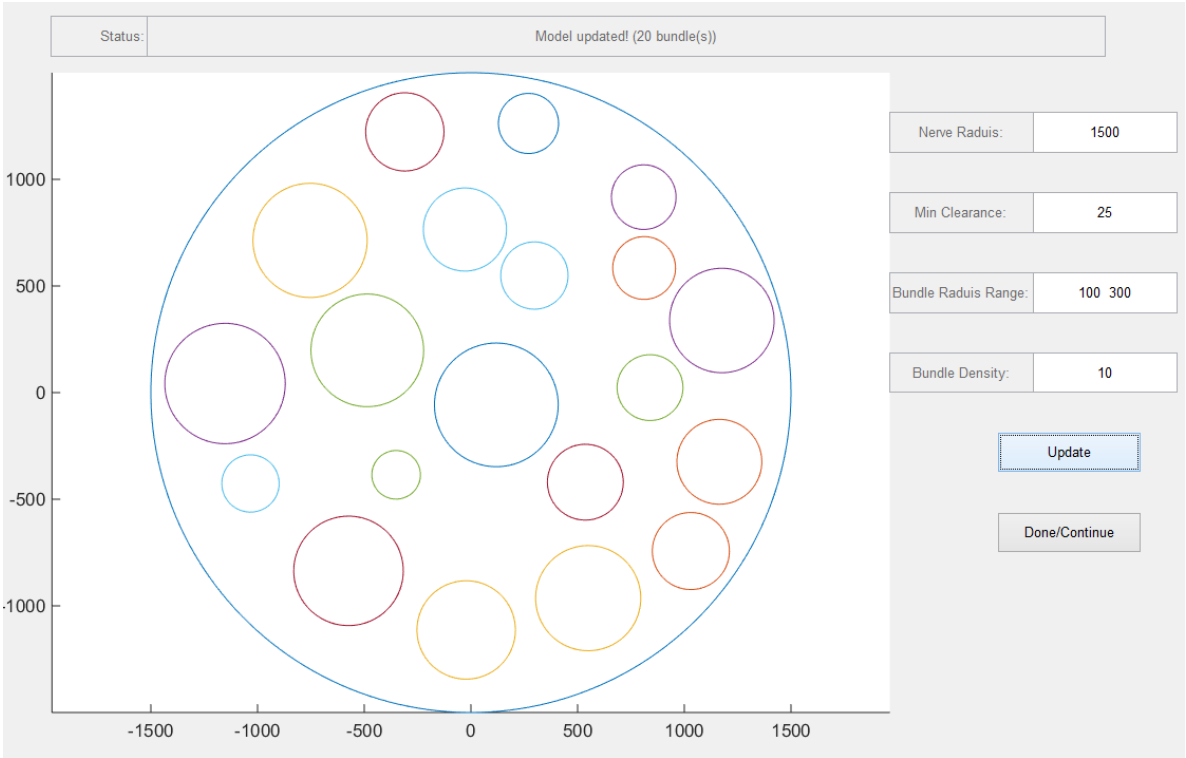

Figure 3: `generate_model` GUI

2.2.3 Saving/Loading

- `M = generate_model(..., ' file', file_name,...)`
  If `file_name` exist in the **models** folder (which is in the working directory) the saved model will be loaded into `M`. If the model does not exist in the models directory, upon `M.save(M)` the model will be saved in *models* directory with the name `file_name`.

- `M = generate_model(..., ' file', file_name, 'rewrite', ...)`
  Same as above with the difference that if the model already exists in the **models** directory, the function will ignore it and generates a new model with the same name or in other words replaces the model.

3 Process Model

3.1 Overview

The first step to process the model is decomposing the geometry into *minimal regions* which then is used to create a mesh. Creating mesh is a vital step for simulation as the simulation runs based on meshed model. The meshed model is a simplified and finite representation of the model.

3.2 Syntax (Minimal Region Decomposition)

- `M = M.create_csg(M)`
  Will create minimal regions and updates the main variable `M`. But will not replace existing CSG data.

- `M = M.create_csg(M, 'rewrite')`
  Will create minimal regions and updates the main variable `M`. Replaces any previous CSG data in the model if existing.

3.3 Syntax (Mesh Generation)

- `M = generate_mesh(M, ...)`
  Will create minimal regions and updates the main variable `M`. Replaces any CSG data.
- `M = generate_mesh(M, ..., 'refine', n, ...)`
  Will refine the mesh `n` times, larger values result in higher number of points, more accuracy but slower performance. Default value is 0 (no refine).
- `M = generate_mesh(M, ..., 'plot_mesh', ...)`
  Plots the mesh after generating mesh. Alternatively the mesh could be plotted later using the following command: `M.plot.mesh()`.
- `M = generate_mesh(M, ..., 'rewrite', ...)`
  Replaces any meshing data in the model if existing, otherwise the existing meshing information will be used.

4 Propagation Simulation (`propogation_alg`)

4.1 Overview

The propagation algorithm is based on meshed data. Each point in the mesh is connected to its closest surrounding points. Based on the length of each connection and the medium speed the surrounding points will be infected in case of infection of each point. The bundle and neuron boundaries can have a delay, this is to resemble their surrounding tissue(?). Different speed of propagation in neurons, bundles and nerve can be set. The default formula for speed of propagation inside a neuron is set to:

Base_Neuron_Speed * 2 / R

Where R is the radius of the neuron. The formula can be changed in the `propogation_alg` file.

4.2 Syntax

- `M = propogation_alg(M, ...);`
  Runs the propagation algorithm
- `M = propogation_alg(M, ..., 'init_insult', ii, ...);`
  Sets the initial insult location, where `ii` is the initial insult coordinates.
- `M = propogation_alg(M, ..., 'meduim_speed', ms, ...);`
  Sets the propagation speed in the nerve, where `ms` is the speed.
- `M = propogation_alg(M, ..., 'bundle_speed', bs, ...);`
  Sets the propagation speed in the bundles, where `bs` is the speed.
- `M = propogation_alg(M, ..., 'neuron_speed', ns, ...);`
  Sets the **base** propagation speed in the neurons, where `ns` is the speed, the speed of neurons will be further scaled but the 2/R formula.
- `M = propogation_alg(M, ..., 'edge_delay', ed, ...);`
  Sets the delay of propagation on the bundle and neuron perimeters/edges.
- `M = propogation_alg(M, ..., 'rewrite', ...);`
  Rewrites any previous simulation data in `M`.
- `M = propogation_alg(M, ..., 'GUI', ...);`
  Opens a Graphical User Interface to choose the initial insult position. Bu clicking on any point the model that point will be selected as the initial insult. By clicking the ***Done/Continue*** button the simulation will be run.

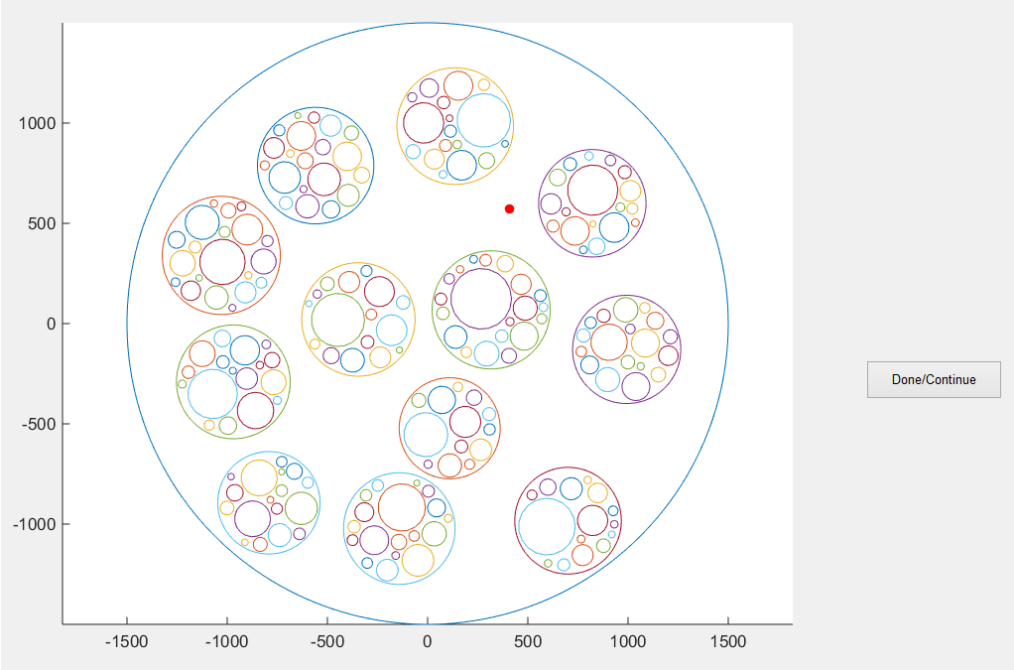

Figure 4: `propogation_alg` GUI

5 Payback and Record Simulation

After simulation all the data will be stored in the main variable (here `M`) and it can be played back or recorded using the following commands:

- `M.P.anim()`
  Opens a window as shown in Figure 5. The slider at the top of the window (also controllable through mouse wheel) controls the speed of playback (which can reversed the play too).
- `P.anim('movie_file', 'nerve1.avi')`
  Will write an avi file to the working directory, where `'nerve1.avi'` is the name of the file to be generated.

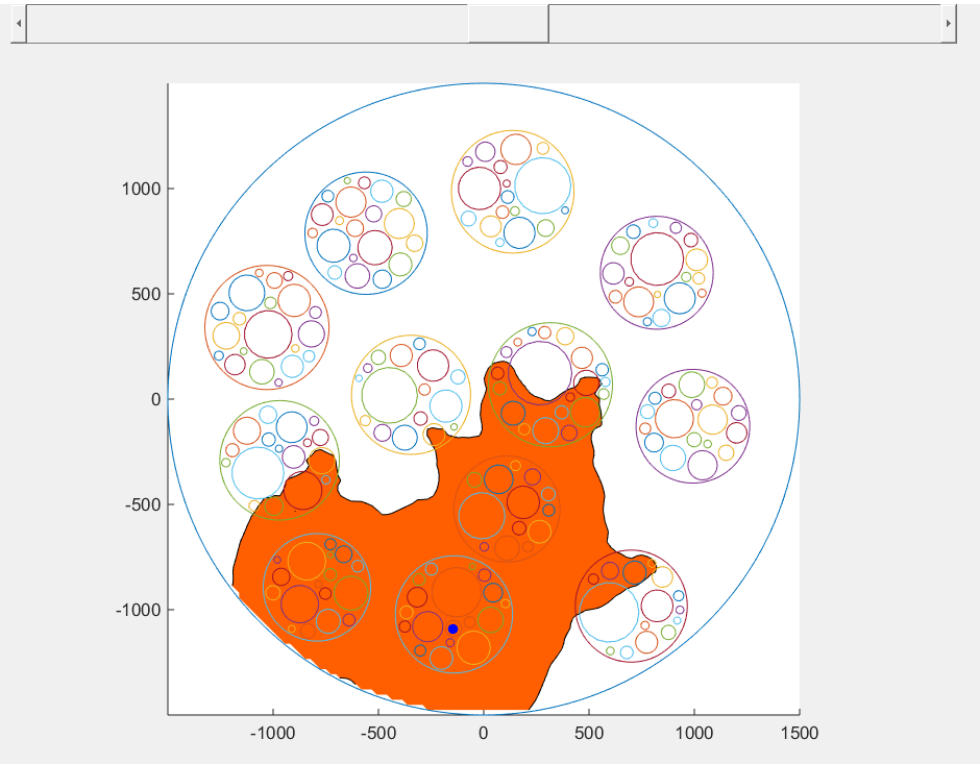

Figure 5: Animation GUI