ReadMe File of the Vertebrate Segmentation Model

July 18, 2020

By Vani Kanoria

1. Compatibility and system requirements:

   Portability: This package requires an installation of MATLAB software on a 64-bit Windows or a 64-bit MacOS machine.

   Parallelization: The code has been designed both for use on a local computer or a computer cluster (supercomputer), as simulations may be run in parallel. To run on multiple processors (a computer cluster), this code requires installation of the Parallel Processing Toolbox in MATLAB.

2. Running Simulations

   Biological and Computational description of the simulation:

   The biological model: A gene expression oscillator, called the segmentation clock, controls segmentation of precursors of the vertebral column. This simulation illustrates the oscillatory behavior of genes belonging to the Hes/her family, which form dimers and negatively autoregulate their own mRNA transcription and protein concentrations. The two-celled computational model is based on the models described in the paper titled 'Short-lived Her proteins drive robust synchronized oscillations in the zebrafish segmentation clock' by Ahmet Ay, Stephan Knierer, Adriana Sperlea, Jack Holland4, and Ertuğrul M. Özbudak.

   Biological system: Her1, Her7 and Hes6 proteins form homo- and heterodimers at different levels and these dimers repress transcriptions of *her1*, *her7* and *deltaC* forming a negative feedback loop that has the potential to create oscillatory gene expression. It is the oscillatory expression patterns of these transcriptional repressors within the presomitic mesoderm that are proposed to be the mechanism underlying the segmentation clock. Delta-Notch signaling enhances the transcription of *her1* and *her7* and ensures their oscillations are synchronized across neighboring cells which is

why in two cell systems, the *deltaC* protein concentration of the other cell is taken into account to calculate the propensity of production of *her1* and *her7* mRNA.

The genes used in this model are the *her1, her6, her7* and *delta* genes. These genes, their mRNA transcriptions and their respective proteins are referred to in the model using the abbreviations in the following table.

Gene name	Her1	Her7	Her6	Delta
Gene name in the Model	h1	h7	h6	d
Model mRNA name	mh1	mh7	mh6	md
Model protein name	ph1	ph7	ph6	pd

Table 1. Abbreviations used for gene, mRNA and proteins in the model.

Mathematical solution:

Each simulation is controlled by a set of parameters, each representing a biological rate, such as *her1* protein synthesis (psh1) or degradation (pdh1) rates. These rates determine the concentration levels of mRNAs, proteins, and protein complexes in the system.

The code includes three kinds of simulations: stochastic, deterministic and hybrid, all of which are in their own folders.

(1) Deterministic

The deterministic model uses ordinary differential equations to simulate this two-cell biological system and solves them using Euler's method.

Running the simulation for a particular parameter set:

Go to the folder titled `VertSeg\deterministic`

To perform the simulation for a given parameter set, in the command window, type:

```
mh1 = deterministic_model(p);
```

where `p` is the parameter set whose solution is being tested.

(i) *Parameter set description:* Parameter sets should be in the form of a vector of size 44 by 1 or 1 by 44.

(ii) *Duration of simulation*: 60000 time steps. It is taken to be equivalent to 600 minutes.
It can be modified by changing the right hand side of the following line in the script `deterministic_model.m`:

time_steps = 60000;

(iii) *Output:* This should return either a matrix of size 2 x time_steps or the double 0. If it returns the double 0, it means the model failed since one of the species' concentrations became negative. If it returns a matrix of size 2 x time_steps, that is the concentration levels of the two cells at each time step i.e. row 1 contains the mh1 concentrations of cell 1 at each time_step and row 2 contains the concentration levels of the two cells at each time step.

(iii) *Plotting simulation:* if your output is saved as mh1 e.g. by running the model as

```
mh1 = deterministic_model(p);
```

To plot the simulation, type into the command window:

```
plot(mh1(1,:));
```

This will plot the mh1 levels of the first cell, which is the same as that of the second cell, since the two cells will be completely in sync in the deterministic simulation.

Testing a given parameter set:

For each set of codes:

i. *Parameter set description:* Parameter sets should be in the form of a vector of size 44 by 1 or 1 by 44.

ii. Generating score: To test the score of a given parameter set, simply enter the following into the command window:

```
score = findScoreVertSeg(p)
```

where `p` is the parameter set being tested.

This will give the score out of 16 by running vani_deterministic for the parameter set and for mutant conditions.

The script `findScoreVertSeg` uses 16 different conditions to test the parameter set.

Mutants and scores: Parameter sets are tested with experimental data collected from the literature on various genetic backgrounds including wild-type and genetic knockdowns and mutations. Each tested experimental condition reports a score based on whether the given parameter set could allow the individual to pass that condition. A parameter set's total score is the sum of the scores produced by each condition. The conditions involve period and amplitude conditions for a given mRNA transcript.

Finding parameter sets with SRES

a. *How SRES finds parameters:* A full description of SRES, written by the creators of this method, Runarsson and Yao, can be found at https://notendur.hi.is/~tpr/software/sres/Tec311r.pdf. In brief, SRES creates a random "population" of parameter sets based on the provided ranges, and it uses an evolutionary algorithm to test this population over a series of generations. The best parameter set, as determined by conditions specified in a conditions file (`vani_deterministic.m`, in the case of our model) outcompetes others, mimicking the famous evolution principle of the survival of the fittest.

b. *Using a parallelized version of the code:* To run code in parallel, as it was mentioned in section **1-b**, you should use the parallelized version of the code. The code can be run in parallel either in a supercomputer (computer cluster) or in a local computer with multiple processors (or cores). The parallel code package is provided to you along with the non-parallel one (serial version). In **Appendix A**, you can find a step-by-step guide on how to set up a cluster profile and how to run parallel jobs from your computer on the cluster.

   In the script `vani_deterministic`, make sure that the parfor loop in `vani_deterministic` (line 50) is not commented out and that the for loop (line 51) is commented out.

Input and output

i. *SRES conditions:* Population size (`popSize`), number of generations (`generations`), number of parents (`parents`), and cutoff score (`Cutoff`) can all be modified in `maximizeVertSeg.m`. Note that in general, the number of parents should be approximately `15%` of the population size.

ii. Ranges: Parameter ranges can be modified in `maximizeVertSeg.m`. `lb` signifies lower bound, and should be a vector containing the lower bounds for each parameter. `ub` signifies upper bound, and should be a vector containing the upper bounds for each parameter. `lb` and `ub` are vectors in `maximizeVertSeg.m`.

Example: We need to change the lower bound of the transcription of *Her7* mRNA to `20`. First, we find out which index of the vector represents it in `deterministic_model.` In line 19 you can see `psh7=param_set(3);` so the 3rd value in the `lb` vector should be changed to `20.`

```
lb=[30,27,20,...]
```

iii. *Outputs:* To run SRES, type the following in the command window:

```
[output, statistics, Gm, VertGoodSets] = maximizeVertSeg();
```

After the code finishes running, the variable output (which can be accessed through the workspace or by typing `output` in the command window) will contain the best parameter set found. Similarly, the variable `statistics` will contain the maximum, minimum, and mean scores for every generation, the variable `Gm` will contain the generation in which the best score was found, and the variable `goodSets` will contain a matrix of all sets with scores above the cutoff score provided in `maximizeVertSeg.m`.

Modifying the code:

1. *Adding states:* A state represents the level of transcription of a specific gene. To add genes, you will need to modify the file `deterministic_model.m`. To create the state, add a new element in the script `deterministic_model` in the section `%STATES` (lines 85-107):

   ```
   newState = zeros(cells,time_steps)
   ```

   Next, add any additional parameters that are necessary for the creation of this state. Please see section **2** (*Adding parameters*) for more details.
   Then, write the equation that determines the rate of change of that state (the rate of change of the concentration level of the gene). For instance, if `newState` is a gene transcript that is constitutively expressed (`newParam1`) and then degraded (`newParam2`), the equation would appear as below:

   ```
   newState_dot = newParam1 – newParam2*newState;
   ```

2. *Adding parameters:* Adding parameters requires modifications to `deterministic_model.m`, `vani_deterministic.m`, and `maximizeVertSeg.m`. First, add the new parameter to the `param_set` vector in `deterministic_model.m` (lines 17-60) as follows, where `new_p_num` is the number of the new parameter:

   ```
   newParam = param_set(new_p_num);
   ```

   Next, modify `vani_deterministic.m` to add constraints to the new parameter by editing matrix `g` as follows:

   ```
   g(:,2*new_p_num-1)=-x(:,new_p_num)+lb(new_p_num);
   g(:,2*new_p_num) = x(:,new_p_num) - ub(new_p_num);
   ```

Then, modify `maximizeVertSeg.m. VertGoodSet` (line 13) should be modified to include the correct number of parameters as follows (e.g. if one parameter was added to a 90-parameter model):

```
VertGoodSet = ones(1, new_p_num);
```

Next, the parameter ranges should be modified. If parameter `new_p_num` is allowed to range between `0` and `3`, for instance, the number `0` should be added to the `lb` (line 25 in `maximizeVertSeg.m`), while the number `3` should be added to the `ub` vector (line 26 in `maximizeVertSeg.m`) as follows:

```
lb = [ …,200,240,0]
```

```
ub = [ …,920,720,3]
```

3. *Adding a test condition:* To add a test condition, first modify `vani_deterministic.m`. Initially the model contains a `param_set_wt` matrix of `6` rows. To add an additional condition, add an additional row to the `param_set_wt` matrix as shown below.

```
param_set_wt=repmat(x,[1,1,7]);
```

Next, specify the condition in the newly added matrix row. For instance, if the condition requires parameter `12` to be `0`, the newly added condition would be written as follows after `line 47` of `vani_deterministic.m`:

```
param_set_wt(:,7,12)=0;
```

Then, determine which features must be checked in the result, and check those conditions only. Inside the for-loop/parfor loop (line 49 in `vani_deterministic.m`), call `deterministic_model` for the new condition similar to the following:

```
mh1_new_mutant = deterministic_model(param_set_wt(k,7,:));
if length(mh1_new_mutant) ==timesteps
[new_period,
new_amplitude]=findPeriodandAmplitude(mh1_new_mutant);
```

Next, calculate the score of the condition, for example:

```
     new_period_score = new_period/wperiod > 1.5;
Here, wperiod is the period of the wild_type.
```

And then add the results to the matrix f:

```
     f(k) = f(k) + new_period_score
```

Then add an `end` for the if-statement. If deterministic_model returns 0 instead of a matrix of size (2 x timesteps), the score for the tests checked for that condition will automatically be 0.

4. *Modifying the cutoff score*

Finally, in `maximizeVertSeg.m`, increase the cutoff score by 1 if necessary on line 14.

```
     Cutoff = 17;
```

Testing the parameter search on a local personal computer: comment out the parfor loop in vani_deterministic (`line 49`) and uncomment the next line (`line 50`) which has a for loop instead.

Adjust the population size (`popSize`), number of generations (`generations`), number of parents (`parents`), and cutoff score (`Cutoff`) in `maximizeVertSeg.m`. Using a small population size, number of generations and parents is sufficient for testing.

Run `maximizeVertSeg.m` as shown in the section above for running it on a cluster by entering the following into the command window.

```
[output, statistics, Gm, VertGoodSets] = maximizeVertSeg();
```

Deterministic_extended

This is in the folder titled `deterministic_extended`, which is separate from the folder titled `deterministic`.

This is an extension of the deterministic code.

`maximizeVertSeg.m` in the `deterministic_extended` folder calls `vani_deterministic_extended` instead of `vani_deterministic`.

`vani_deterministic_extended` adds an additional condition in lines 73-75. It calls the script `satisfies_all_conc_constraints` which tests that the mRNA, protein and dimer concentrations are below the set upper limits for each of those. If the concentrations are within the constraints, it adds a point to the total score.

Therefore, the maximum possible score for `vani_deterministic_extended` is 17, and not 16.

Further, the last condition tested in `vani_deterministic_extended` calls `deterministic_model_extended` instead of `deterministic_model`.

`deterministic_model_extended` returns the concentrations of these species `[mh1,mh7,md,ph1,ph7,pd,ph11,ph17,ph16,ph77,ph66,ph76]` instead of just mh1 concentrations.

Parameter set description, Duration of simulation, *Output, Plotting simulation:* same as in `deterministic`.

Also, finding parameter sets using SRES and modifying the code: same as in `deterministic`.

To call `maximizeVertSeg.m`, type the following into the command window:

```
[output, statistics, Gm, VertGoodSets] = maximizeVertSeg();
```

(2) Hybrid

There are deterministic-stochastic hybrid algorithms implemented in the hybrid code, each of which have a one-cell version and a two-cell version:

1. hybrid_model2 (one-cell version) and hybrid_model2_2cell (two-cell version) are based on Gillispie's First Reaction Method. We have incorporated a delay into the method.
2. hybrid_model3 and hybridmodel3_2cell implement the Next Reaction Method for systems with delays .

We modeled these codes on the algorithm used in the HIV model in the paper 'HIV Quasispecies Dynamics during Pro-Active Treatment Switching: Impact on Multidrug Resistance and Resistance Archiving in Latent Reservoirs.' The code was obtained from the supplementary materials of the paper. The code is included in the hybrid_code folder as HIV_hybrid_code.m and HIV_other_code.m. The codes are not used directly in any manner to run the hybridmodel2_2cell or hybridmodel3_2cell or their respective one-cell versions. We also referred to the first and next reaction hybrid methods described in the paper titled 'Adaptive Simulation of Hybrid Stochastic and Deterministic Models for Biochemical Systems.'

The algorithms use ordinary differential equations to model the deterministic part and we use ode45, a built-in MATLAB solver to solve them. We use the 'events' function of ode45 to stop the integration when an 'event' occurs. When the event occurs, the code fires a stochastic reaction.

Running the simulation for a particular parameter set:

Go to the folder titled `VertSeg\hybrid_code`

You can simulate the system using any of the following codes:

(i) `hybrid_model2.m` (ii) `hybrid_model2_2cell.m`

(iii) `hybrid_model3.m` (iv) `hybrid_model23_2cell.m`

To perform the simulation for a given parameter set, in the command window, using the respective codes above, type the following :

```
(i) Y = hybrid_model2(p);
(ii) Y = hybrid_model2_cell(p);
(iii) Y = hybrid_model3(p);
(iv) Y = hybrid_model3_cell(p);
```

where `p` is the parameter set whose solution is being tested.

i. *Parameter set description:* Parameter sets should be in the form of a vector of size 44 by 1.

ii. *Duration of simulation*: 100 minutes.

It can be modified by changing the right hand side of the following line in the relevant script (the hybrid code being used):

```
minutes =100;
```

This is on `line 17 of hybrid_model2` and `hybrid_model2_2cell`, and on `line 18` of `hybrid_model3` and `hybrid_model3_2cell.`

iii. *Output:*

This is different for the one-cell versions and the two-cell versions.

1. *One-cell versions* (`hybrid_model2` and `hybrid_model3`)

   The output is of size (minutes x num_states).

   `num of states = 14` as stated at the beginning of the codes.

   These codes return the concentrations of all the species at one minute intervals of the simulation.

   The 14 molecule types in order are:

   [ph1;ph7;ph6,pd;mh1;mh7;mh6;md;ph11;ph76;ph17;ph16;ph77;ph66].

   Each column contains the concentrations of the species of the corresponding index at one minute intervals of the simulation. For example, column 2 contains the concentrations of species of index two in the above vector i.e. it contains the ph7 concentrations.

2. *Two-cell versions* (`hybrid_model2_2cell` and `hybrid_model3_2cell`)

The output is of size (minutes x (num_cells*num_states)).

`num_states = 14` and `num_cells = 2`, as stated at the beginning of the codes, so for these codes, the output is of size (minutes x 28).

Since these codes simulate two-cell systems, they return the concentrations of all the species of the simulation for both the cells at one minute intervals.

The output consists of 28 columns and rows = `minutes`.

The first 14 columns contain the concentrations of the following species of cell 1 (in the given order):

species_array = [ph1;ph7;ph6,pd;mh1;mh7;mh6;md;ph11;ph76;ph17;ph16;ph77;ph66].

Columns 15-28 contain the concentrations of the species in the same order but for cell 2.

i.e. if column number <=14, the species is of cell and species type = species_array(column number),

and if column number >14, the species is of cell 2 and species type = species_array(column number-14),

Example 1: column 2 contains the concentrations in cell 1 of species of index two in the above vector i.e. it contains the ph7 concentrations.

Example 2: column 20 contains the concentrations in cell21 of species of index 20-14=6 in the above vector i.e. it contains the mh7 concentrations.

iv. *Plotting simulation:* if your output is saved as `Y` e.g. by running the model as

```
Y = hybrid_modelX(p);
```

where `hybrid_modelX` is the version of the hybrid model used.
and `minutes = 100,`

1. To plot all concentrations, type into the command window:
   `plot(Y);`

2. To plot mh1 concentrations of the first cell, type into the command window,
   `plot(Y(:,5))`

Modifying the code:

Disclaimer: note that the codes are complex, and any changes made to the structure of the code must be reflected in the functions used and the other structures impacted.

1. *Adding states:* First, change num_states.
   `num_states = N1;`
   where `N` is the new number of states.

   Next, add any additional parameters that are necessary for the creation of this state.
   Please see section **2**(*Adding parameters*) for more details.

   Also, modify `get_R.m` to include any reactions that may involve this state.
   `get_R()` returns a matrix of net changes in species levels caused by firing of the reactions (1st dimension: states, 2nd dimension: reaction)
   e.g. `R(2,20) = -1` represents that the second state is changed by -1 units when the second reaction is fired.
   Further, change the value of the variable `num_reactions` to reflect this.
   `num_reactions = N2;`
   where `N` is the new number of reactions.

Also, if the reaction is delayed, modify the variable `partition` and the cell array `s` accordingly. You will also have to modify the function `start_delayed_reaction` to add the reaction.

Also modify the function get_propensities to include the propensity for the new reaction and include the new state:

```
new_state = y(N1)
```

2. *Adding parameters:* First, add the new parameter to the vector `x` in `the hybrid code` as follows, where `new_p_num` is the number of the new parameter:

```
newParam = x(new_p_num); (lines 39-46)
```

Incorporate any use of the new parameter: in the propensity array `a` and the `get_r()` function etc.

(3) Stochastic

The stochastic simulations in our study are performed using the Next Reaction Method incorporating delays, which discretely computes concentration levels based on probabilistic calculations in the paper by David F. Anderson (2007). Probabilistically determined propensities and reaction times are used to decide which reaction fires at each iteration. Reactions with higher propensities are more likely to fire. A delayed reaction queue is incorporated into the standard NRM algorithm to accommodate time delays. Each iteration in NRM is computed as follows:

1. Update the propensity values related to the most recently fired reaction for each cell.

2. Calculate the time gap (the size of the next time step) using propensities.

3. Increment the time step and the relevant molecular counts.

4. If a delayed reaction is initiated, add it to the appropriate list. Otherwise fire immediate reactions and delayed reactions that are finished.

5. Repeat until simulation time expires.

The code was modeled after the scripts nrm_genepaired.m and nrmgeneunpaired.m which also implement the Next Reaction Method.

Delayed Reactions:

Here, some of the reactions are delayed while others are fired without delay. The reactions which are delayed are the transcription of mh1, mh7, mh6, md, ph1, ph7, ph6 and pd.

There are also several versions of the stochastic code:

Some of them use a dependency structure for updating the propensities which means that when a reaction is carried out, only the propensities altered by that reaction are altered instead of

calculating the propensities of all the reactions again before the next reaction is chosen. This helps increase the speed of simulations.

*Versions of the cod*e:

(i) vani_stochastic_v1.m: a one-cell system with 34 reactions and without dependency structure.

(ii) vani_stochastic_v2.m: a two-cell system with 34 reactions and without dependency structure.

(iii) vani_stochastic_v3: a 2-cell system which uses approximations for dimer formation and degradation, which is why it has 16 reactions. It does not use a dependency structure.

(iv) vani_stochastic_v4: a 2-cell system which uses approximations for dimer formation and degradation, which is why it has 16 reactions. It utilizes a dependency structure, updating propensities wherever it updates concentrations.

Running the simulation for a particular parameter set:

Go to the folder titled `VertSeg\stochastic`

To perform the simulation for a given parameter set, in the command window, type:

```
Y = vani_stochastic_v<X>(p);
```

where `<X>` is replaced by the version that you wish to use: 1, 2, 3 or 4.

and `p` is the parameter set whose solution is being tested.

i. *Parameter set description:* Parameter sets should be in the form of a vector of size 44 by 1 or 1 by 44.

ii. *Duration of simulation*: 60000 time steps. It is taken to be equivalent to 600 minutes.

It can be modified by changing the right hand side of the following line in the script:

`tend = 100;` (in lines 50-55 of the script)

iii. *Outputs:*

(i),(iii) vani_stochastic_v1 and vani_stochastic_v1 return `Data` as output where

`Data = [Time' mh1v']` for vani_stochastic_v1
and
`Data = [Time' mh1v_c1' mh1v_c2']` for vani_stochastic_v13

`Time` is a vector containing doubles that indicate the time in the simulation at which any stochastic reaction is fired.
`mh1v` is the mh1 concentration of the cell to the corresponding time in `Time`.
i.e. for vani_stochastic_v1, `Data` returns a matrix of the number of rows = the timesteps recorded i.e. the times at which the stochastic reaction was fired, and number of columns = 2: the first column being the time steps and the second column being the corresponding mh concentrations in the cell.

For vani_stochastic_v3,
`mh1v_c1` is the mh1 concentration of cell 1 to the corresponding time in `Time` and
`mh1v_c1` is the mh1 concentration of cell 2 to the corresponding time in `Time`.

i.e. `Data` returns a matrix of the number of rows = the timesteps recorded i.e. the times at which the stochastic reaction was fired, and number of columns=3: the first column being the time steps and the columns 2 and 3 being the corresponding mh concentrations in cell 1 and cell 2 respectively.

(ii),(iv) vani_stochastic_v2 and vani_stochastic_v4 return `[sync_score, Data]` as output where

`Data = [Time' mh1v_c1' mh1v_c2'];`

`mh1v_c1` is the mh1 concentration of cell 1 to the corresponding time in `Time` and `mh1v_c1` is the mh1 concentration of cell 2 to the corresponding time in `Time`.

i.e. `Data` returns a matrix of the number of rows = the timesteps recorded i.e. the times at which the stochastic reaction was fired, and number of columns=3: the first column being the time steps and the columns 2 and 3 being the corresponding mh concentrations in cell 1 and cell 2 respectively.

`sync_score` is a double which reflects the synchronization between cells 1 and 2. It is the Pearson coefficient of correlation between columns 2 and 3 of `Data.`

iv. *Plotting simulation:* if your output is saved as Data e.g. by running the model as

`Data = vani_stochastic_vX(p);` (v1)

or `[Data, sync_score] = vani_stochastic_vX(p);` (v2, v3, v4)

where `vani_stochastic_vX` is the version of the stochastic code used.

and `minutes = 100,`

1. To plot the mh1 concentrations of both cells, type into the command window:
   `plot(Data(:,1), Data(:,2), Data(:,1), Data(:,3));`

2. To plot mh1 concentrations of the first cell, type into the command window,
   `plot(Data(:,1), Data(:,2));`

v. *To find the period and amplitude:* The function `findPeriodandAmplitude(mh1)` in the folder stochastic takes an input of mh1 concentrations (either one cell or two cell) and returns the period and amplitude of cell 1 as output. The input should have the following dimensions: (num_cells x num_timesteps) where num_cells is the number of cells (1 or 2) and num_timesteps

is the number of times at which the mh1 concentration is recorded. Number of columns of mh1 should be greater than 15 for the function to work.
The function uses a moving average with k=40 timesteps to smoothen the data before calculating the period or amplitude. A value more than five values to its left and five to its right was considered a local maximum and a value less than five values to its left and five to its right was considered a local minimum. For every peak-trough pair the value of the period and amplitude was calculated. To obtain an overall value of the period and amplitude for a run these values were then averaged.

vi. *To find the synchronization score of the cells in vani_stochastic_v3*: vani_stochastic_3 does not return the sync_score of cells 1 and 2. To find the synchronization score, type into the command window,

```
sync_score = corr(Data(:,2), Data(:,3),'Type','Pearson');
```

Modifying the code:

1. *Adding states:* A state represents the level of transcription of a specific gene. To add genes, you will need to modify the file `vani_stochastic_vX.m` where `X` is the version being used. To create the state, add a new element in the section `%States for each of the cells` (lines 84-86).

   ```
   newState = [0;0];
   ```

   where `newState` is the new state added.

   Next, add any additional parameters that are necessary for the creation of this state. Please see section **2**(*Adding parameters*) for more details.

   Also, modify the script to include any reactions that may involve this state.

   change the value of the variable `num_reactions` to reflect this.

   ```
   num_reactions = N2;
   ```

   where `N` is the new number of reactions.

Also, if the reaction is delayed, modify the variable `partition` and the cell array `s` accordingly. You will also have to modify the function `start_delayed_reaction` to add the reaction.

Also modify the function get_propensities to include the propensity for the new reaction and include the new state:

```
new_state = y(N1)
```

2. *Adding parameters:* First, add the new parameter to the vector `param_set` in the code as follows, where `new_p_num` is the number of the new parameter:

   ```
   newParam = x(new_p_num); (lines 5-48)
   ```

   Incorporate any use of the new parameter: in the propensity array `a` or in any reactions.

Disclaimer: vani_stochastic_v3 and vani_stochastic_v4 do not produce oscillations in mh1 concentrations. This may be due to any bugs in the code or due to approximation of dimer formation and degradation equations.

Appendix A

Using the Matlab Parallel Processing Toolbox on the
Colgate Biomath Cluster from Local Computers

This file contains the procedures to set up a cluster profile and run jobs on the biomath cluster easily from your local computer. Note: The cluster setup instructions are written for the Colgate University Biomath Cluster, and various clusters might have different access modes and different setup procedures.

How to set up the cluster profile in your local computer

- Unzip the **matlab.zip** folder included in the package, copy all of its contents (not the folder itself) and paste it to the **local** subfolder in your **MATLAB** package contents.
  For Mac: Applications → MATLAB_R2016a (right click on the application icon → Show Package Contents) → toolbox → local
  For Windows: C: → MATLAB → SupportPackages → R2016a → toolbox → local.

- Open **MATLAB_R2016a** (the application).

- Add the **local** folder to path on your working directory (for a Mac drag it from the finder).

- On the command window, type **configCluster**

- Type your Colgate Biomath username when you see **"Username on Biomath (e.g. joe):"** on the command window.

- Set the cluster queue name to 'matlab' (type **ClusterInfo.setQueueName**('matlab') on the Command window)

- Shortly after you type your username, in a separate dialog box, you will be asked to type your password for your cluster account. If you are not prompted to type your password, you will be requested to type it during the validation step.

- Edit the number of workers for the new cluster: **Home → Parallel (Drop-down Tab)→ Manage Cluster Profiles → biomath_remote_r2016a** (choose from cluster profiles) → **Edit** (button at bottom right) → **NumWorkers** (The default is 128 workers, which is the maximum number of workers you can use for one job in the Biomath Cluster. You could set it to a number smaller than 128 - 18, 37, 64 - if your jobs do not require a large number of workers. Please note that validating might take longer with a large number of workers).

- Now, validate the new cluster profile as follows: **Home→ Parallel →Manage Cluster Profiles→biomath_remote_r2016a→Validate** (button on top)

- The validation process will prompt you to use a validation file. Click NO, and then provide the password to your cluster account. Wait for it to finish validation so that you can start submitting jobs.

The last(5th) validation test might not pass, but it doesn't matter since we will run jobs using a different pool from the one that the configCluster function uses.

How to queue a job to the cluster

- Open MATLAB

- Add the unzipped **clusterFunctions** folder attached to the email to your path.

- In the MATLAB command window create a cluster object by typing the following:

  c = parcluster;

- Set your cluster preferences by typing the following in the command line (look at the end for more options on cluster preferences)

  ClusterInfo.setProcsPerNode(12) - sets the number of processors per node used for the job (this job, for example, uses 12 processors per node – The larger your parallel loop is, the more processors you will need to run the code faster.)

  ClusterInfo.setQueueName('matlab') - sets the name of the queue in which the job is being queued

- Type the following in the command window:

  job1 = c.batch(@function, output, input, 'Pool', #ofWorkers, 'AttachedFiles', {'your code folder path here'})

@function – your main function that you are running on the cluster

output – number of outputs of your function

input – number of inputs

#ofWorkers – the number of workers you want to use for the job.

Example: If you want to run a job with 12 processors per node and you want to use 3 nodes (12ppn*3nodes = 36 processors) you should request 35 workers since the head node is counted outside from the pool workers.

Job Examples:

```
job_windows = c.batch(@sampleCode, 4, {}, 'Pool', 11,
'AttachedFiles', {'C:\MATLAB'})
```
(windows machine)

```
job_mac = c.batch( @sampleCode, 1, {}, 'Pool', 11,
'AttachedFiles', {'/Users/jsmith/Downloads/CodePackage'}
)
```
(Mac machine)

Here, make sure that the **CodePackage** is in the right folder.

Useful Commands:

job1.fetchOutputs – returns the outputs of the job once it finishes running.
job1.State - returns the state of the job; queued, finished or running.
job1.diary - shows information about a job (eg. printed statements)
job1.delete - deletes a job.

ClusterInfo.clear - clears all the cluster properties and sets them back to their default values.

To monitor the jobs that you have submitted: **Home → Parallel → Monitor Jobs.**

Appendix B

A sample study on how to find parameter sets using SRES

Initial ranges of the parameters are determined based on literature search. Original initial ranges can be found commented out in maximize.m. Using the initial ranges, we simulate a parameter search as follows:

For running in a local personal computer

- Open Matlab and add the Code Package folder to path.
- In `maximizeVertSeg.m` set the number of generations, population size, parents and the minimum score cutoff.
- Run the following line in the command window:

```
[output, statistics, Gm, VertGoodSets] = maximize()
```

Once the simulation finishes running, in the workspace or simply by typing the variable name in the command window, you can access the `output, statistics, Gm`, and `VertGoodSets` variables.
You can use either the parameter set titled 'output' or any of the sets from VertGoodSets. Here, I have used Set1 from VertGoodSets.

Set1 = 37.1000 47.8740 16.8370 43.5350 0.2983 0.2509 0.2992 0.3091 57.2650
44.1160 47.4550 59.3180 0.3381 0.1972 0.3872 0.1880 0.2639 0.2875 0.2728
0.3043 0.2718 0.2893 9.8992 8.7802 8.9150 1.6025 0.9123 1.6195 10.9670
0.0173 0.2478 0.0277 0.1098 0.0011 0.2455 0.0127 0.2803 0.0214 0.0805
0.0054 0.1363 711.4900 280.1800 511.8000

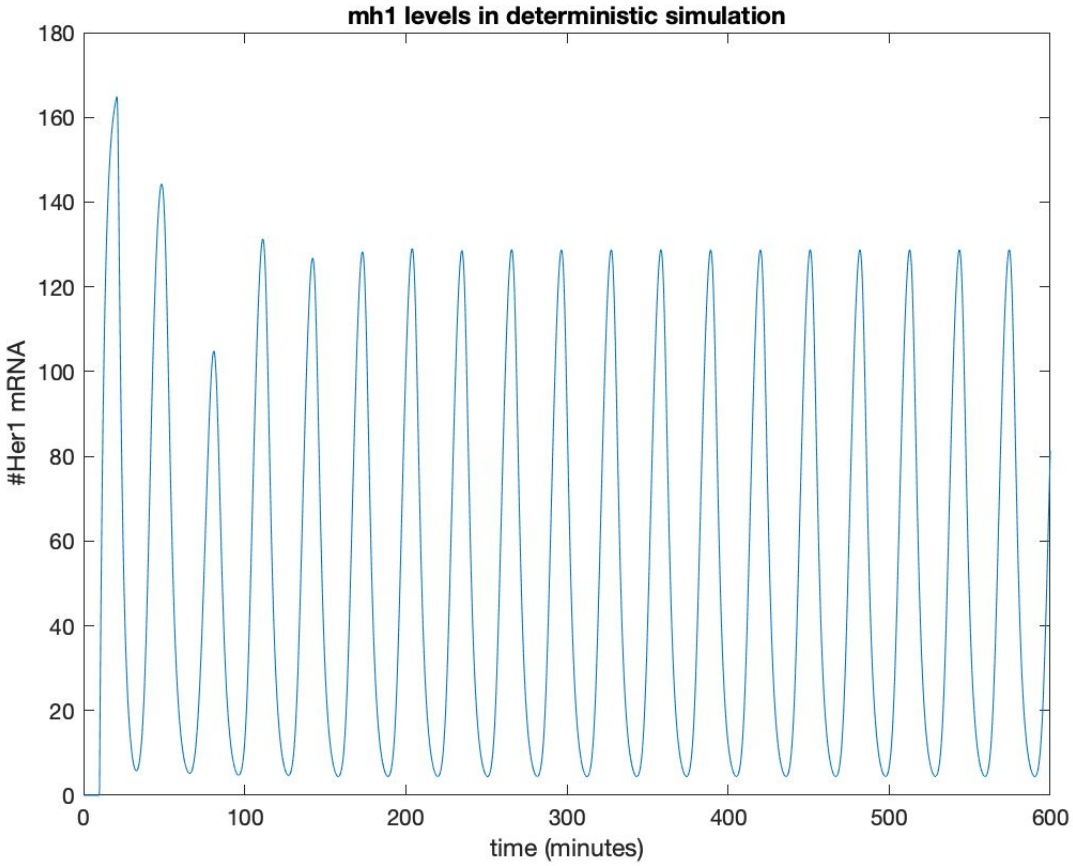
We get the mh1 levels by running deterministic_model.m with that parameter set.

```
Y = deterministic_model(Set1)
```

Next, we plot this parameter set (we plot only the first cell since both cells are synchronized in the deterministic simulation):

```
plot(Y(1,:));
```

Figure 1. The concentrations of all *Her1 mRNA* using the script deterministic_model.m and the parameter set 'Set1' as input



We can clearly see that the mh1 levels oscillate regularly.

For running in the cluster

1. Connect to the cluster using the instructions outlined in Appendix A and validating the cluster for a particular number of workers NumWorkers (e.g. 16 or 32)
2. Update the script titled 'RunScript' so that :

   (i) `ppn = NumWorkers` e.g. `ppn = 16.`

   (ii) `total Procs = NumWorkers-1` (This should be equal to ppn-1)

   (iii) `job5 =`

   `c.batch(@maximizeVertSeg,4,{},'Pool',totalProcs,'AttachedFiles',{'/Users/vani/Documents/MATLAB/VertSeg/S20ProfAy'});`

   a. `job5` is replaced with the job and new job number. When running more than one job, change the job number so that the outputs of the multiple jobs can be fetched later.

   b. `/Users/vani/Documents/MATLAB/VertSeg/S20ProfAy` is replaced with the location in your personal computer where you have stored maximizeVertSeg.m.

   (iv) job181_State = job181.State;

   %job172_Diary = job171.diary;

   job181_Output = job181.fetchOutputs;

   make sure the job variables above have the same job number as in (iii).

   Other information about RunScript:

   `c = parcluster` ensures that the default cluster profile will be used

   (v) `dlmwrite('BestSets.csv',job181_output{4},`

   `'delimiter',',','-append');`

   `BestSets.csv` is the name of the csv file in which the fourth output of `maximizeVertSeg` will be saved. You can change the name if you wish.

   make sure `job181_output{4}` has the same job number as in (iii).

3. RunScript is divided into sections using %%.

   Run the first and second sections of RunScript using the 'Run Section' button in the 'Editor' tab of Matlab. If you run the whole script, you will probably get an error saying that jobs cannot be fetched until they are in the state 'finished.
4. Monitor your job on the Job Monitor, which can be accessed by clicking the 'Parallel' button in the 'Home' environment of Matlab, and then clicking on 'Job Monitor.'
5. When the job is indicated to be 'finished' in the Job Monitor, run the third and fourths sections of Run Script.
6. You can access the output by clicking on `jobXoutput` where X is the job from sections2 and 3 of RunScript or by entering `job181_output{4} in the command window`. The different parts of the output are explained in the Deterministic section.

Appendix C

Two sample studies on how to run the hybrid codes

Here, too we use the parameter set Set1.

(i) Running a 100 minute simulation using `hybrid_model2.` This is the one-cell version. (This can be done only on a personal local computer, not on a cluster since it is not parallelized.)

- Open Matlab and add the Code Package folder to path.
- In `hybrid_model2.m,` set the number of minutes (line 17).
  We set, `minutes = 100;`
- Save the script.
- Run the following line in the command window:

```
[Y] = hybrid_model2(Set1)
```

- When the simulation has finished running (this one-cell takes about 20 minutes for a 100 minute simulation with parameter set Set1), we plot the mh1 concentrations in this simulation. We enter the following into the command window:

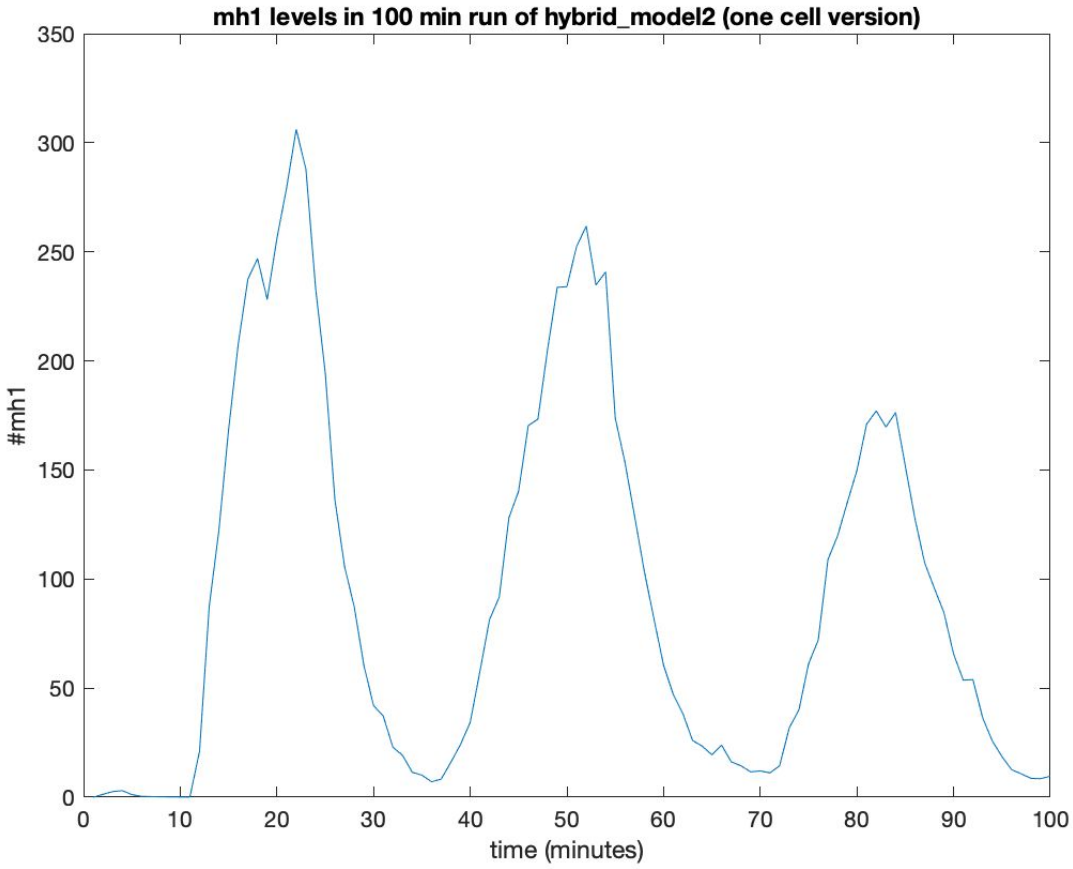
```
plot(Y(5,:));
```

- We can labels to the axes using the following commands:

```
xlabel('time (minutes)');
ylabel('#mh1');
```

- The output:

Figure 2. The concentrations of all *Her1 mRNA* using the script hybrid_model2.m and the parameter set 'Set1' as input



- Here, we see that the system is oscillating because we see 3 distinct peaks.

(ii) Running a 200 simulation using `hybrid_model3_2cell`. This is the two-cell version of hybrid_model3. (This simulation can be done only on a personal local computer, not on a cluster since it is not parallelized.)

- Open Matlab and add the Code Package folder to path.
- In `hybrid_model3_2cell.m`, set the number of minutes (line 18).

  We set,

  ```
  minutes = 200; (line 18)
  ```

- Save the script.
- Run the following line in the command window:

  ```
  [Y] = hybrid_model3_2cell(Set1)
  ```

- When the simulation has finished running (this one-cell takes about 50 minutes for a 200 minute simulation with parameter set Set1), we plot the concentrations of all the species in this simulation. We enter the following into the command window:

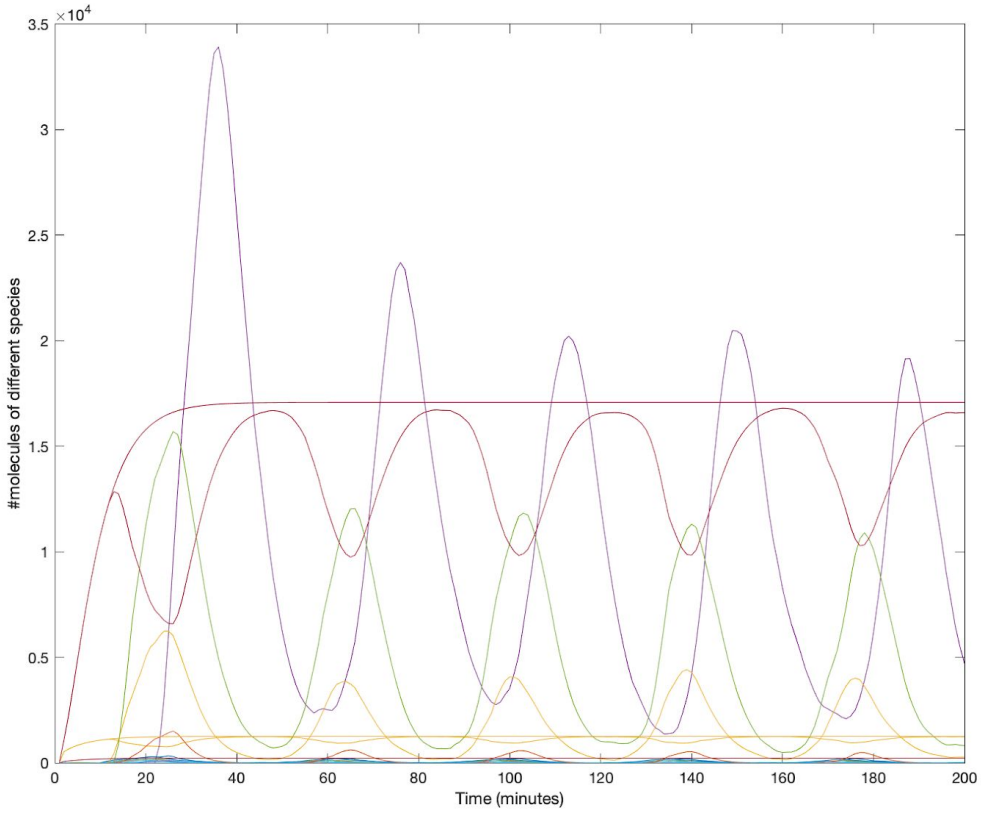
  ```
  plot(Y);
  ```

- We can labels to the axes using the following commands:

  ```
  xlabel('time (minutes)');
  ylabel('#molecules');
  ```

- Result:

Figure 3. The concentrations of all species using the script hybrid_model3_2cell.m and the parameter set ‘Set1’ as input



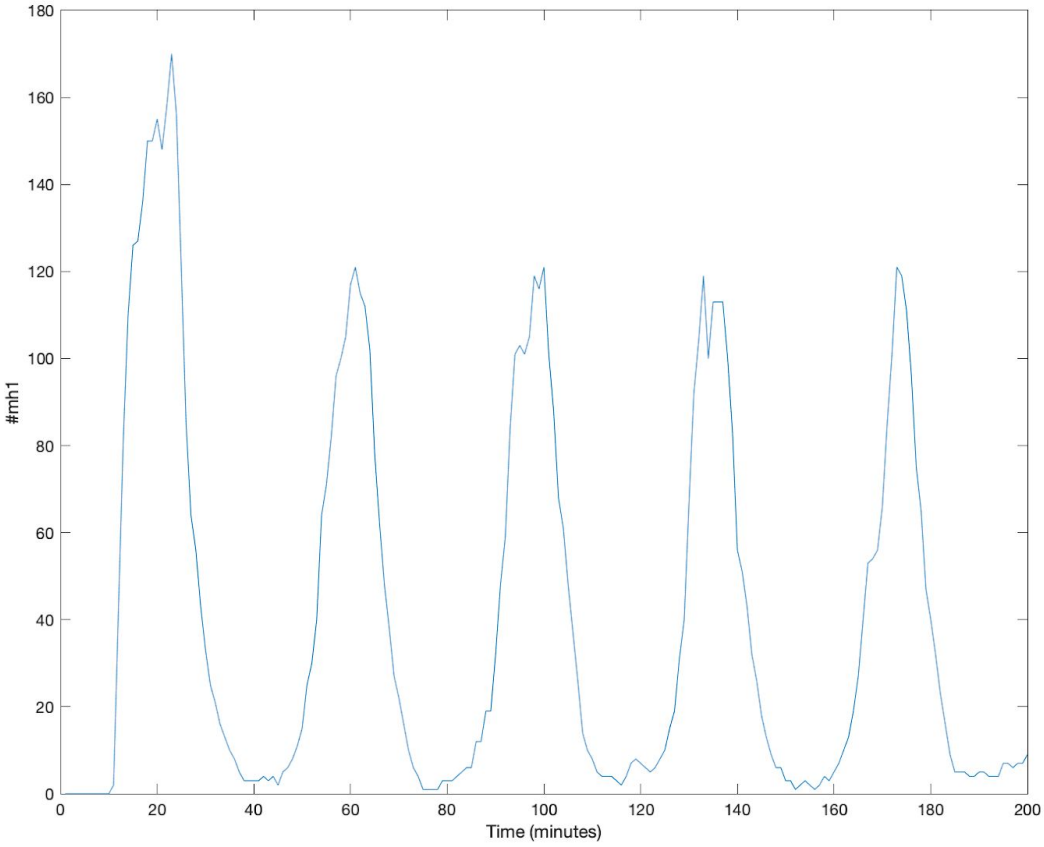
- We see that the concentrations of different species are oscillating in the above figure.

- We can also plot the mh1 concentrations of cell 1 by typing the following into the command window:

```
plot(Y(:,1))
```

- Results:

Figure 4. The concentrations of all species using the script hybrid_model3_2cell.m and the parameter set 'Set1' as input



- We can see that the mh1 concentration oscillates and we observe five peaks in 200 minutes.

Appendix D

Three sample studies on how to run the stochastic codes

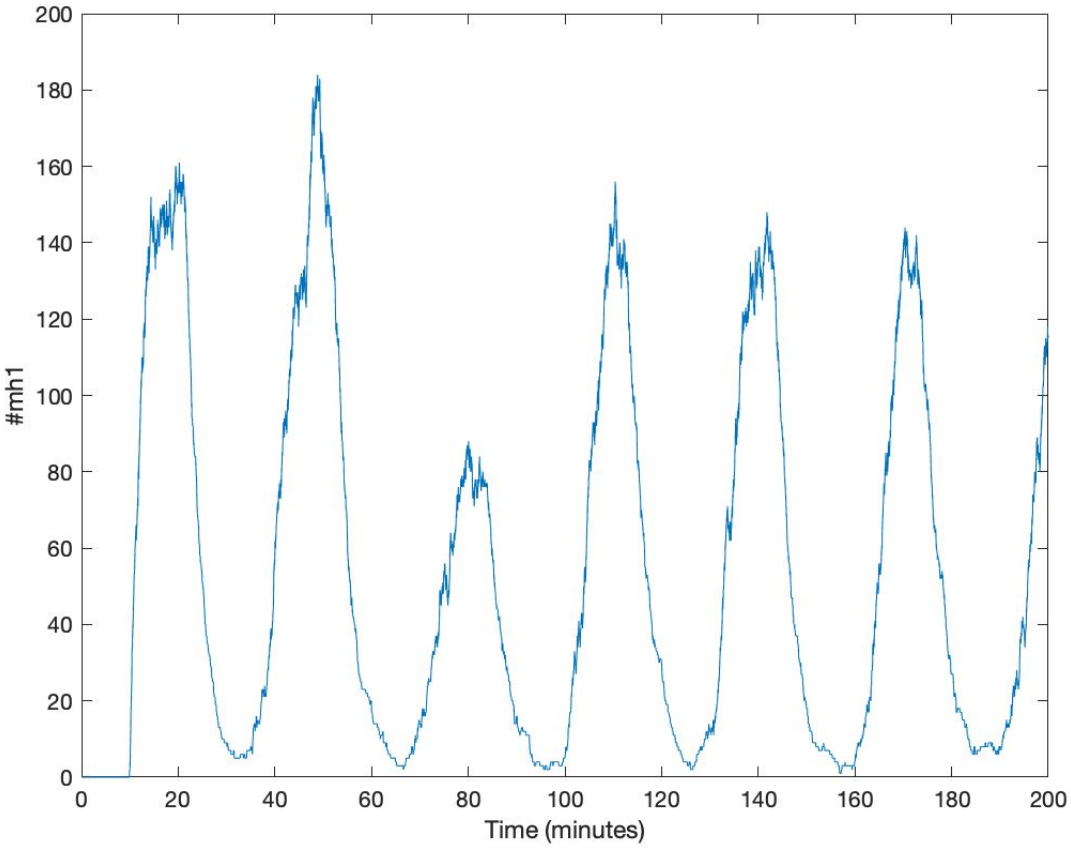
(i) Running a 200 minute simulation using `vani_stochastic_v1` and the parameter set 'Set1'. This is the one-cell version with 34 reactions and neither any approximations for dimer formation or degradation, nor any dependency structure. (This simulation can be done only on a personal local computer, not on a cluster since it is not parallelized.)

- Open Matlab and add the Code Package folder to path. This package is titled `stochastic.`
- In `vani_stochastic_v1.m,` set the number of minutes (line 18).
  We set `tend = 200; (line 53)`
- Save the script.
- Run the following line in the command window:
  `[Y] = vani_stochastic_v1.m`

- When the simulation has finished running (this one-cell takes about 20 minutes for a 200 minute simulation with parameter set Set1), the code automatically plots a figure of the Time vs the mh1 concentration. It also adds the xlabel and ylabel.

- Results:

Figure 5. The concentrations of all *Her1 mRNA* using the script vani_stochastic_v1.m and the parameter set 'Set1' as input



- The mh1 levels are clearly oscillating: we see 6 distinct peaks in the above figure.

- To find the period and amplitude of the oscillations, we type the following into the command window:

```
[period, amplitude]= findPeriodandAmplitude(Y)
```

The result was:

```
period =
   30.8519
amplitude =
  124.2541
```

- Disclaimer: the vani_stochastic_v1 code runs unexpectedly fast for the parameter set Set1: it could be because of the parameter set or because of some bug in the script. A check of the code is recommended.

(ii) Running a 200 minute simulation using `vani_stochastic_v4`. This is a 2-cell system which uses approximations for dimer formation and degradation, which is why it has 16 reactions instead of 34 reactions. It also utilizes a dependency structure, updating propensities wherever it updates concentrations.(This simulation can be done only on a personal local computer, not on a cluster since it is not parallelized.)

- Open Matlab and add the Code Package folder to path. This package is titled `stochastic.`
- In `vani_stochastic_v4.m`, set the number of minutes (line 18).
  We set `tend = 200; (line 55)`

- Save the script.
- Run the following line in the command window:

```
[Data, sync_score] = stochastic_model_v4(Set1)
```

- When the simulation has finished running, we plot the mh1 concentrations of both cell 1 and cell 2 in this simulation. We enter the following into the command window:

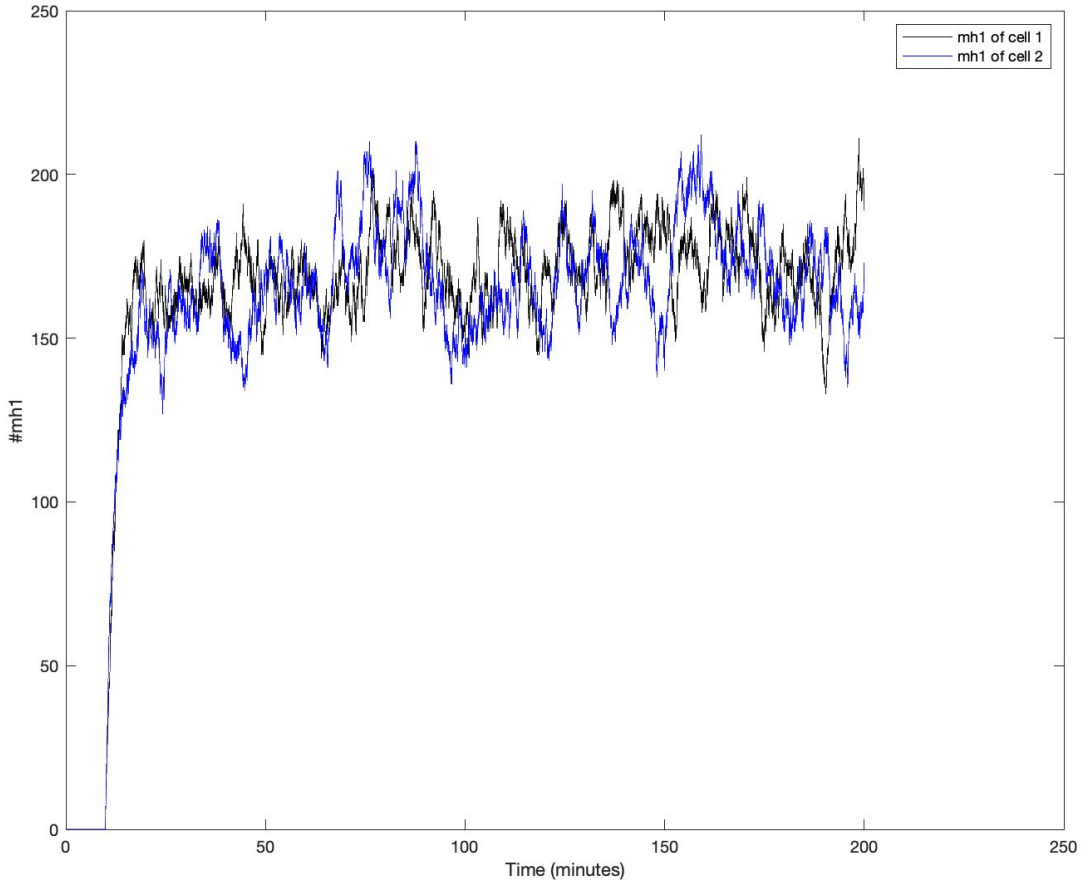
```
plot(Data(1,:), Data(2,:), Data(1,:), Data(3,:));
```

- We can labels to the axes using the following commands:

```
xlabel('time (minutes)');
ylabel('#mh1');
legend('mh1 of cell 1','mh1 of cell 2')
```

- Results:

Figure 6. The concentrations of *Her1 mRNA* of cell 1 and cell 2 using the script vani_stochastic_v4.m and the parameter set 'Set1' as input




```
sync_score = 0.36
```

- It is clear from the figure above that the mh1 concentrations are not oscillating.
- Disclaimer: This may be due to some bug in the code or due to the approximation of dimerization formation and degradation reactions. A check of the code is recommended.

(iii) Running a 100 minute simulation using `vani_stochastic_v3`. This is a 2-cell system which uses approximations for dimer formation and degradation, which is why it has 16 reactions instead of 34 reactions. It does not utilize a dependency structure, .This simulation can be done only on a personal local computer, not on a cluster since it is not parallelized.)

- Open Matlab and add the Code Package folder to path. This package is titled `stochastic`.
- In `vani_stochastic_v3.m`, set the number of minutes (line 18).
  We set `tend = 100; (line 55)`

- Save the script.
- Run the following line in the command window:

```
Data = stochastic_model_v4(Set1)
```

- When the simulation has finished running, we plot the mh1 concentrations of both cell 1 and cell 2 in this simulation. We enter the following into the command window:

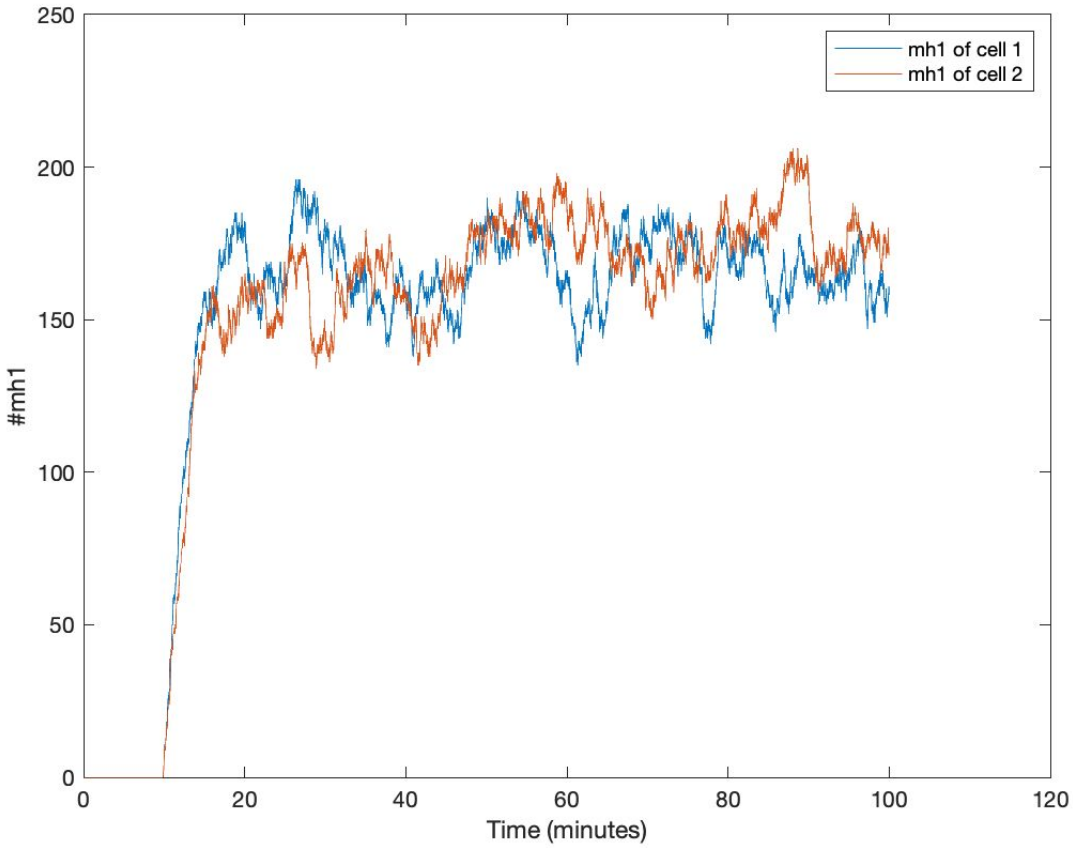
```
plot(Data(1,:), Data(2,:), Data(1,:), Data(3,:));
```

- We can labels to the axes using the following commands:

```
xlabel('time (minutes)');
ylabel('#mh1');
legend('mh1 of cell 1','mh1 of cell 2')
```

- Results:

Figure 7. The concentrations of *Her1 mRNA* of cell 1 and cell 2 using the script vani_stochastic_v3.m and the parameter set 'Set1' as input



- The mh1 levels are clearly not oscillating.

- To find the synchronization score, type into the command window:

```
>> sync_score = corr(Data(:,2),Data(:,3),'Type','Pearson');
```

Result:

```
sync_score = 0.4405
```
