Understanding the dynamics of a complex biological network without parameter information: Case study on the cell-cycle network of fission-yeast (*S.pombe*)

Souvadra Hati (Sr: 15551)

June 18, 2021

Introduction: Understanding and predicting dynamics of complex biochemical networks is one of the fundamental goals of Systems Biology. Although, even a single cell is way too complex for us to model all its biochemical behaviours using our current understanding and compute capacity, scientists have been successful in modelling some of the biochemical pathways essentials in every living organism. One of the computationally feasible ways to predict the dynamical behaviours of these networks is to numerically solve a bunch of coupled Ordinary Differential Equations (ODEs) with appropriate kinetic parameters that can be experimentally measured. But that process requires extensive wet-lab experiments to find out the vast array of biochemical parameters necessary to model even modests of the biological pathways, which slows down the process of building their models. This has motivated scientists to come up with innovative ideas in order to gain insight of a biochemical pathway without extensive knowledge of the parameters involved in it. In this article I am going to discuss two such ways of modelling a network and show how they can be useful by applying those in the cell-cycle pathway of fission-yeast (*S.pombe*).

Discrete Method: One of the most simple yet elegant ways to model the essentials of a network is to model it as a graph model with each protein / regulator as a node and the interactions between those regulators as the edges of the graph. The edges will only represent the the sign of the interaction between those two nodes. Basically, the node will only take into account of the fact that the interaction is activating (+1) or inhibiting (-1). After that, in each discrete ieration the state of the the daughter nodes of the initialized nodes (can be themselves as well if self-activation or self-inhibition is present) will be updated

as per the update rule mentioned below.

$$S_i(t+1) = \begin{cases} 0, & \text{if } \sum_j a_{ij} S_j(t) < \theta_i \\ 1, & \text{if } \sum_j a_{ij} S_j(t) > \theta_i \\ S_i(t), & \text{if } \sum_j a_{ij} S_j(t) = \theta_i \end{cases}$$

Here, $S_j(t) \in \{0, 1\}$ is the binary value assigned to node j at iteration t, which discretely denotes if the protein is present in the system at that iteration or not. $a_{ij} = 1$ denotes an activating interaction from node j to node i, and similarly $a_{ij} = -1$ denotes an inhibiting edge and $a_{ij} = 0$ denotes no interaction (no edge between those two nodes). θ_{ij} is a threshold of activation of node i which is generally 0, unless otherwise mentioned [1].

One interesting aspect of this Boolean modelling is that, we can actually start the model iteration using all the possible initial conditions and that will be in most cases very much computationally feasible. For example, if our network of interest has 7 nodes, then there will be in total $2^7 = 128$ inital conditions, which means we can effectively sample the total solution space of the network which is absolutely impossible in a continuous scenario.

So, in this manner, without any knowledge of the kinetic parameters in the model or ever solving any ODE at all, we can gain insight into the dynamics of the model as I shall discuss using a case study in the later half of the report.

Continuous Method: Although the discrete method in theory can provide us a lot of information regarding the network of interest, the harsh reality is that no biological system is actually discrete and introducing even moderate amount of realism requires us to write the ODE of the reactions and solve them numerically to observe the dynamics of the network. But that requires access to the set of kinetic parameters that we are trying to avoid in our modelling paradigm.

So, to tackle that exact challenge Huang et al. 2018 [2] published an article on a software that they named "RACIPE: Random Circuit Perturbation". It takes, just like the Boolean method, only the topology of the core regulatory circuit and unbiasly generates an ensemble of mathematical models, each of which is characterized by a unique set of kinetic parameters from the ensemble of models, we can analyze the robust dynamical properteis of the core circuit via downstream statistical analysis. In RACIPE, the effects of the "peripheral factors" are modeled as random perturbations to the kinetic parameters. RACIPE samples its parameters across a wide range (via some random distribution) keeping the half functional rule (which states that each regulatory link has about 50% chance of being activated) valid. The RACIPE generated gene-expression data can later be analyzed using differnt statistical tools (Hierarchical clustering analysis (HCA), and Principal Component Analysis (PCA) to name a few) to get a holistic view of the dynamical feature of the network. All these are based of the previous studies that says that robust features in any gene regulatory network remains conserved against large parameter purturbations due to the the restraints from the circuit topology itself.

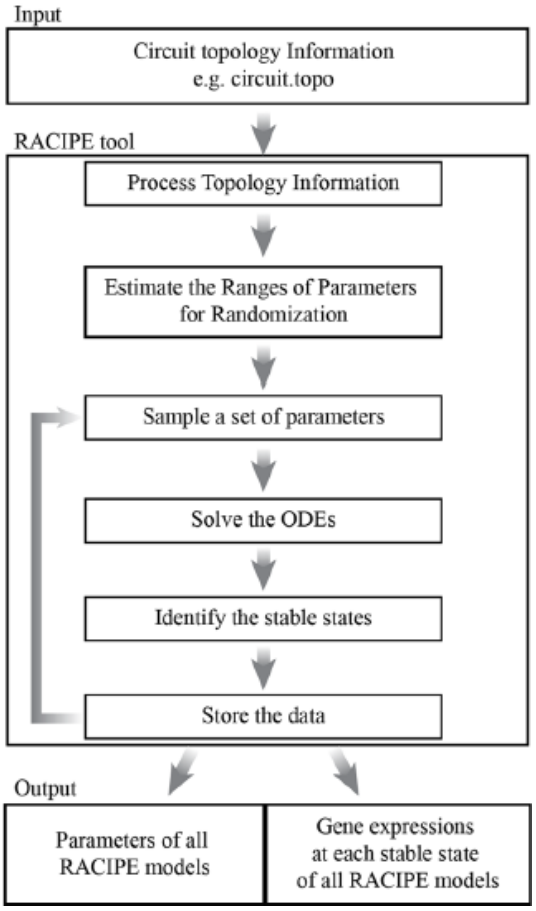


Figure 1: Workflow of RACIPE [2]

Input: The primary input of this toolbox is the circuit topology that is written in a '.topo' file (e.g. "circuit.topo"). Each line of this file specifies a regulatory link, which contains the source node, target node and mode of interaction (1 for activation, 2 for inhibition). Example of a '.topo' file is shown below.

Source	Target	Type
A	B	2
B	A	2

The above .topo file represents a 'Toggle Switch' network which is essentially two master regulators mutually inhibiting each other.

Process Topology Information: This process builds the ODEs based on the circuit topology (input). As an example the above circuit will be represented as

$$\frac{dA}{dt} = G_A H^S(B, B_A^0, n_{BA}, \lambda_{BA}^-) - k_A A$$
$$\frac{dB}{dt} = G_B H^S(A, A_B^0, n_{AB}, \lambda_{AB}^-) - k_B B$$

Where, A and B represents the concentration of the protein A and B as a function of time, G_A and G_B are the maximum production rate of A and B. k_A and k_B are the innate degradation rates of the the corresponding proteins, and H^S is non-linear shifted Hill function defined as:

$$H^S(B, B_A^0, n_{BA}, \lambda_{BA}^-) \coloneqq \lambda_{BA}^- + \frac{1 - \lambda_{BA}^-}{1 + \left(\frac{B}{B_A^0}\right)^{n_{BA}}}$$

λ_{BA}^- is the maximum fold change of A caused by inhibitor B.

When multiple regulators target a gene, the functional form of the rate equations assumes that these regulators are independent and hence, the overall production rate becomes the product of the innate production rate of the target genes and the shifted Hill functions for all the regulatory links.

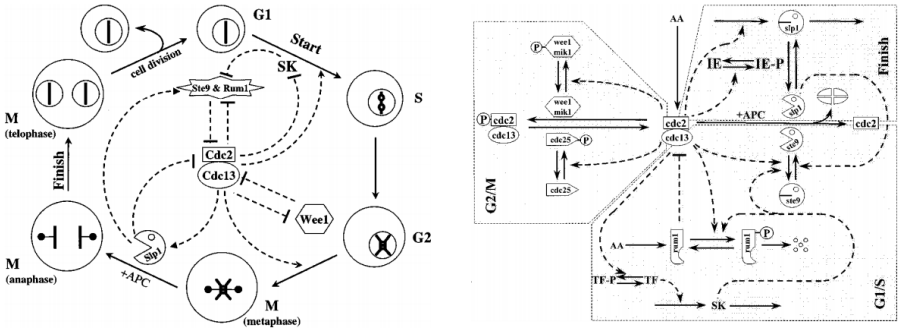
Estimatation of parameters: Ranges of the threshold values in the shifted Hill functions are estimated numerically to satisfy the "half-functoinal" rule. Most of the other parametes are preset and sampled via a random distribution (which is 'uniform distribution' by default). All these parameters are stored in a '.prs' (parameter) file (e.g. "circuit.prs").

Solving the ODE, Identifying the stable steady states: RACIPE repeats the numerical solutions of coupled ODEs numerically for each sampled parameter set for a large number of random initial conditions (optional input to RACIPE) and the steady state solutions for each parameter set are stored in the output solution files.

Case Study: Modelling cell-cycle of fission-yeast To show the effiency of Boolean and RACIPE based models, I have chosen an already well studied model of fission-yeast cell-cycle as a case-study. The goal here is to see how far we can go without any konwledge about the actual parameter values of the network.

Introduction to the model: The full process of the cell-cycle consists of four stages: $G1 \rightarrow S \rightarrow G2 \rightarrow M$. Where

- G1: the cell grows in this phase
- S: DNA synthesis and chromosome replication happens in this phase
- G2: Gap phase
- M: Chromosomes are separated and the cell is divided in this phase



.

Figure 2: (Left) Outer cycle demonstrate the general phases of fission-yeast cell cycle and the inner network shows the raltionships among the principal molecular components. (Right) The wiring diagram of the cell-cycle engine emphasizing on the proteolysis of Cdc13 and phosphorylation of Cdc2 subunit and their stoichiometric inhibition of the complex. [3]

the major role in fission-yeast cell-cycle is played by a cyclin dependent protine kinase complex Cdc2/Cdc13 with Tyr-15(also called MPF (M-phase promoting factor). When Tyr-15 is unphosphorylated, the MPF complex reaches high activity (represented as $Cdc2/Cdc13^*$ in the simplified model). This MPF complex is inactive during G2 phase, when it is phosphorylated (represented as $Cdc2/Cdc13$). The other members in the network are **positive regulators** of kinase Cdc2/Cdc13: "Start" (indicator of the mass/size of the cell), "SK" (Start Kinase), group of Cdk/ cyclin complexed and phosphatase Cdc25, **and** the **negative regulators**: "Slp1", "Rum1", "Ste9", and phosphatase "PP". The simplified network is described in figure 3.

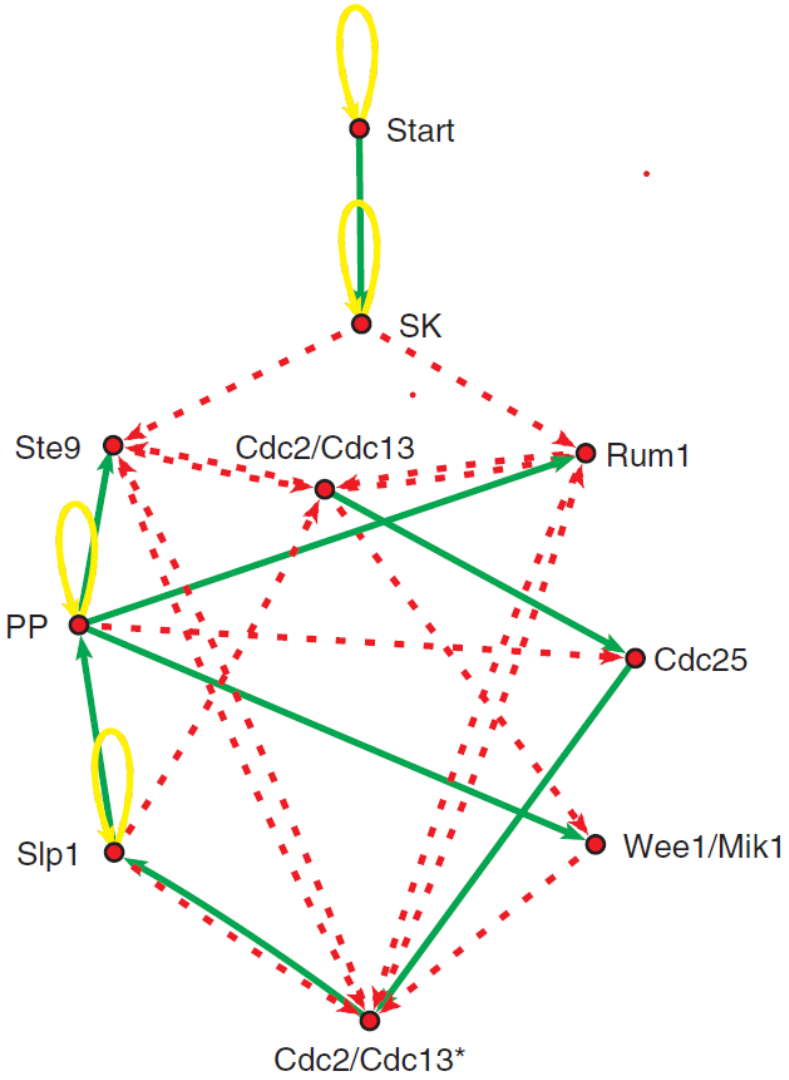


Figure 3: Simplied version of fission-yeast cell-cycle network [1]

In the above network, the green edges denotes activation, the red dotted edges denote inhibition, and the yellow loops denote self-degradation.

Discrete Method: The Biologically relevant starting condition for the network is All nodes OFF except Start, Ste9, Rum1, Wee1/Mik1. So, if we start the Boolean simulation of the network starting from the biolgical starting condition we get the time evolution as shown in table 1.

Time Step	Start	SK	Cdc2/Cdc13	Ste9	Rum1	Slp1	Cdc2/Cdc13*	Wee1 Mik1	Cdc25	PP	Phase	comments
1	1	0	0	1	1	0	0	1	0	0	START	Cdc2/Cdc13 dimers are inhibited, antagonists are active.
2	0	1	0	1	1	0	0	1	0	0	G1	SK are becoming active
3	0	0	0	0	0	0	0	1	0	0	G1/S	When Cdc2/Cdc13 and SK dimers switch off Rum1 and Ste9/APC, the cell passes 'Start' and DNA replication takes place, Cdc2/Cdc13 starts to accumulate
4	0	0	1	0	0	0	0	1	0	0	G2	Activity of Cdc2/Cdc13 achieves moderate level, which is enough for entering G2 phase but not mitosis, since Wee1/Mik1 inhibits the activity of residue Tyr-15 of Cdc2 (Cdc2/Cdc13* is not active)
5	0	0	1	0	0	0	0	0	1	0	G2	Moderate activity Cdc2/Cdc13 activates Cdc25
6	0	0	1	0	0	0	1	0	1	0	G2/M	Cdc25 reverses phosphorylation, removing the inhibiting phosphate group and activating Cdc2/Cdc13*
7	0	0	1	0	0	1	1	0	1	0	G2/M	Cdc2/Cdc13* reaches high activity level sufficient to activate Slp1/APC mitosis
8	0	0	0	0	0	1	0	0	1	1	M	Slp1 degradates Cdc13, that is inhibits complex Cdc2/Cdc13 and Cdc2/Cdc13*.
9	0	0	0	1	1	0	0	1	0	1	M	Antagonists of Cdc2/Cdc13 are reset.
10	0	0	0	1	1	0	0	1	0	0	G1	Cell reaches G1 stationary state (PP is inactive)

Table 1: Biologically relevant time evolution of the cell-cycle [1]

.

The temporal evolution of the network surprisingly demonstrate some of the fundamental steps of cell-cycle, and the fun part is that the coputational requirement of this particular simulation is so small that we can even reproduce this result using just a pen and paper as well.

Then, if the Boolean model is simulated with all the $2^{10} = 1024$ possible initial conditions, we can observe a bunch of stationary points emerging out of the simulation (preseneted in table 2). Surprisingly the largest attractor among those belongs to a fixed point attracting 73% of all network states and that state belongs to biological **G1 phase**.

Attractor	Type	Basin size	Start	SK	Cdc2/Cdc13	Ste9	Rum1	Slp1	Cdc2/Cdc13*	Wee1/Mik1	Cdc25	PP
1	FP	762	0	0	0	1	1	0	0	1	0	0
2	LC	208	0	0	0	0	0	0	0	0	1	1
	LC	0	0	0	0	0	0	1	0	0	1	0
	LC	0	0	0	1	1	1	0	1	1	0	0
3	FP	18	0	0	0	0	1	0	0	1	0	0
4	FP	18	0	0	0	1	0	0	0	1	0	0
5	FP	2	0	0	0	1	0	0	0	0	0	0
6	FP	2	0	0	0	1	0	0	0	0	1	0
7	FP	2	0	0	0	1	0	0	0	1	1	0
8	FP	2	0	0	0	0	1	0	0	0	0	0
9	FP	2	0	0	0	0	1	0	0	0	1	0
10	FP	2	0	0	0	0	1	0	0	1	1	0
11	FP	2	0	0	0	1	1	0	0	0	0	0
12	FP	2	0	0	0	1	1	0	0	0	1	0
13	FP	2	0	0	0	1	1	0	0	1	1	0

Table 2: All attractors found in the Boolean simulation [1]

Continuous Method: Although the Boolean simulation gave us some quite useful information about the network, it cannot possibly encompass all about the the network because of its hard partioning of states into HIGH and LOW and discrete time evolution, which *does not* happen in reality. Besides, there are only 1024 inital conditions possible in the discrete model, which does not represents all the solution space.

To deal with the above issues, I have tried to use RACIPE simulations to model this network. But, there is a fundamental design clash of using RACIPE

to simulate this model, because RACIPE was primarily built to show the robust steady states shown by gene regulatory networks where we are expecting to see oscillatory cycles in our fission-yeast cell-cycle network due to its biological function. So, to deal with these issues, I have tried to merge some of my own codes along with RACIPE in order to gain some insight of the cycles seen in this network.

Our first task is to get hints about oscillations from the RACIPE ouputs. For, that we can simply plot the different stability states shown by the RACIPE solution files. From figure 4, we can see that around 8800 out of 10000 parameter sets have given rise to mono-stable and bi-stable steady state solutions, where as rest of the 1000 parameters have given rise to apprantly 'decastable solutions', which might sound absurd. But, what we are just seeing is a design choice present in RACIPE. First of all, the 10^{th} columns shows solutions which are decastable or higher. But, as we can observe that multistable solutions beyond tristability is statistically insignificant, possibility of decastable solutions are almost none. So, what is happening here is that, when the parameter sets give rise to an oscillating pattern and not a steady state, the RACIPE simulation keeps getting last value of its numerical simulation in different conditions for all the randomly sampled inital conditions, and ends up deciding that the solution state is a higher-order multistable state and puts in the 10th column. So, basically presence of 1000 parameter sets giving rise of decastable solutions is actually saying that 1000 out 10000 sampled parameters showed oscillations, which was expected from a cell-cycle network.

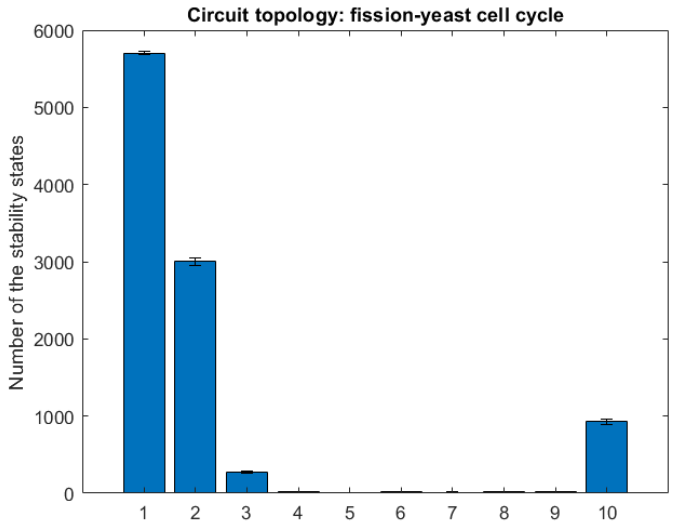
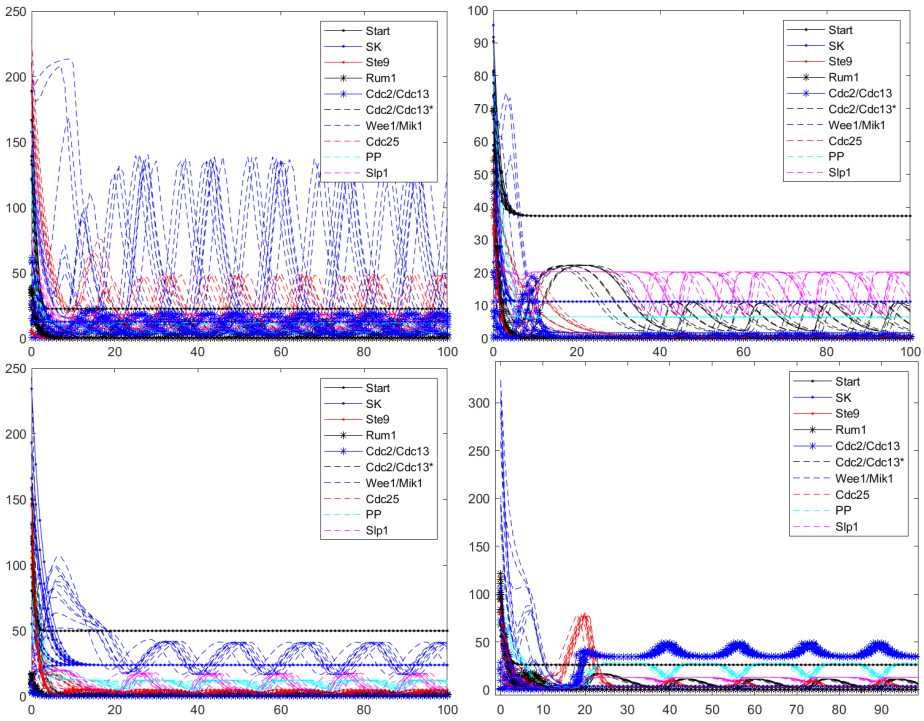


Figure 4: Multistability solutions from RACIPE simulation
X-axis → 1: Monostable solutions, 2: Bistable solutions, ...

.

To test my hypothesis, I wrote down a MATLAB program to extract the parameter sets sampled by RACIPE (present in the '.prs' file) giving rise to the "decastable" solutions and simulate in the same manner via coupled ODEs using shifted Hill equations and plotted the dynamics of those network for random 10 (close to biologically relevant) initial conditions. I plotted the dynamics of the circuit w.r.t time for 4 random parameter sets and found that indeed most of those parameter sets gave rise to oscillating cycles as visible in figure 5 plots.



.

Figure 5: Dynamics of the regulators of the network given the parameter sets generating decastable solutions in RACIPE. Y-axis is the concentration of the ten regulators and X-axis is time steps.

Next, I tried to observe the RACIPE solutions and find any possible cluster and for that applied hierarchical clustering on that solution data and plotted the histogram given in figure 6.

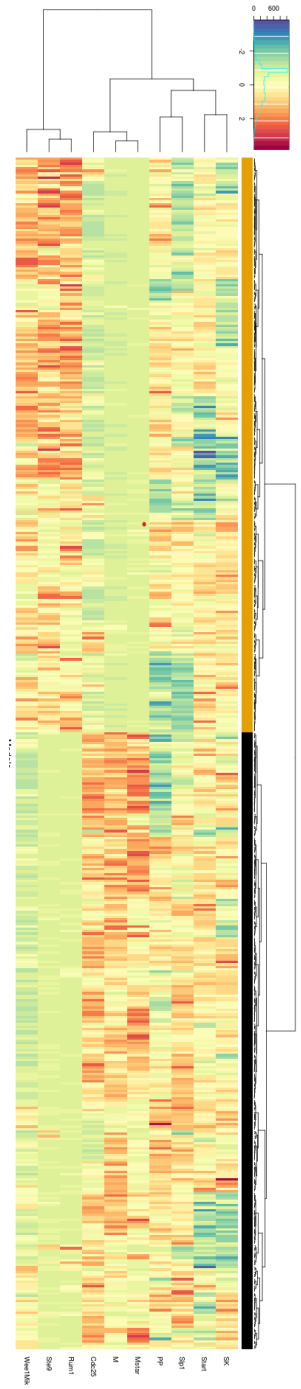
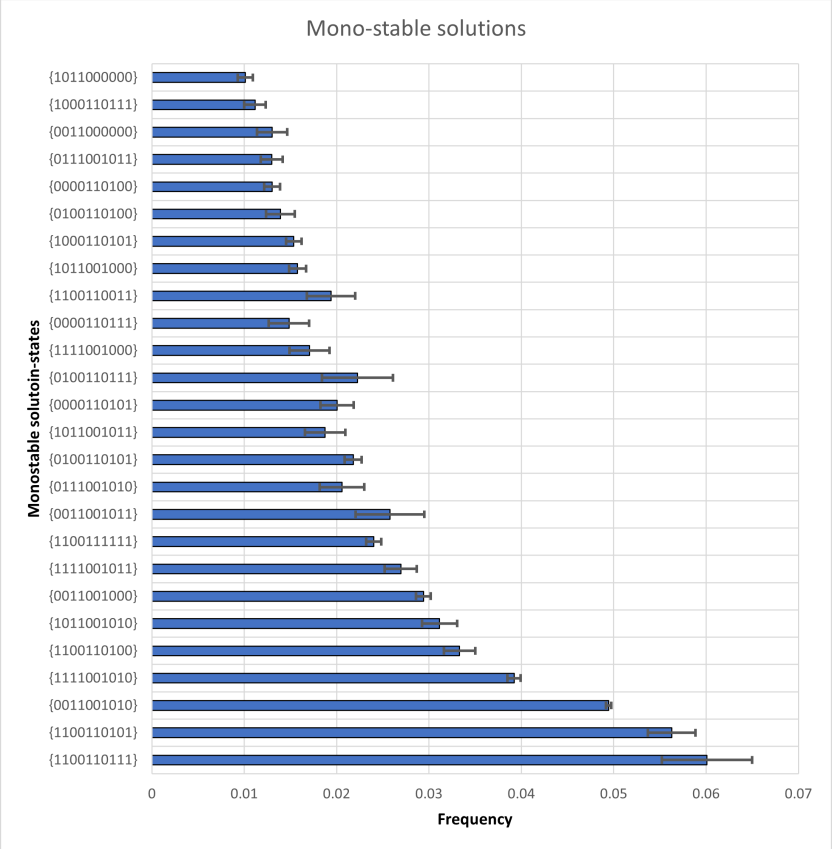


Figure 6: Hierarchical clustering of RACIPE simulation data
M := $Cdc2/Cdc13$, M-star := $Cdc2/Cdc13^*$

As I could not find any clear clustering present in the solution, I wrote a MATLAB program to individally check each solutions of the monostable and bistable conditions and plot how many parameter-sets give rise to each of the solution-state (representing attractors of the systems). I have plotted the statistically significant states that are often found in the vast solution spce of the RACIPE solution due to its highly random nature of sampling inital conditions and parameter sets. So, for that purpose, I have first converted the solutions give in RACIPE files (I performed three independent RACIPE simulations to attain statistically significant results) from log2 scale to normal scale, performed G/K normalization and Z-normalization on the data, to make it more presentable, and then kept tab on each and every solution that fall in one of the solution-state bucket created by my code. I plotted the most frequently found mono and bi-stable steady state solutions as evident in figure 7. In these plots, we found all the attractors found in Boolean simulation and many more. Although, the G1 biological state condition is not the most frequent state in the RACIPE simulations, it is still there and statistically much more significant than around thousands others attractors present in the RACIPE solutions.



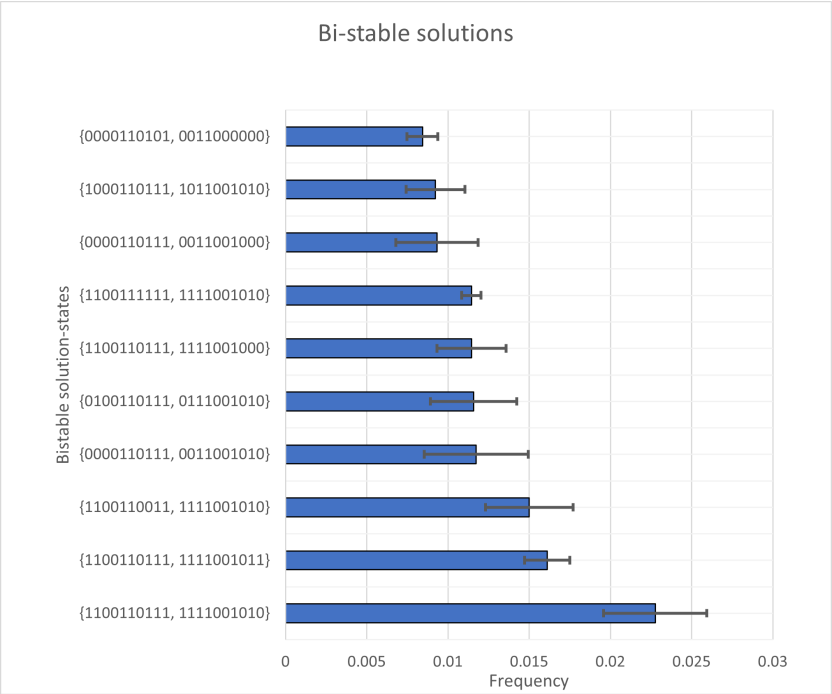


Figure 7: Attractors of the cell-cycle network

.

The naming convention of the above states are as follows: 1 *denotes HIGH, and* 0 *denotes LOW state. And those ten consecutive sequences of ones and zeros basically shows the high/low concentration of* $Start, SK, Ste9, Cdc2/Cdc13, Cdc2/Cdc13^*, Wee1/Mik1, Cdc25, PP,$ *and* $Slp1$ *in that exact order.*

Shortcomings of these models and Future work: Despite showing such detailed information of the fissoin yeast cell-cycle network, there are some inherent shortcomings of these methods and also some lack of expertise from my side as well.

First, I need to perform some more analysis of the cycles shown by the decastable parameter states to see how close those oscillations are to the biologically observed oscillations of those regulators during the cell-cycle.

Next, I need to do some more literature review to find the biological significance of those monostable and bistable steady-state solutions found in both Boolean and RACIPE solutions.

At last, although Boolean, despite being such computatoinally effienct method, was capable of providing stark amount of details of this cell-cycle network, RACIPE is probably a bad choice of software for a gene regulatory network whose primary job is to maintain an oscillation and not converging to some stable states. This required me to write a lot of additional programs to tackle

some of those inherent RACIPE specific problems.

References

[1] M. I. Davidich and S. Bornholdt. Boolean network model predicts cell cycle sequence of fission yeast. *PLOS ONE*, 3(2):1–8, 02 2008.

[2] B. Huang, M. Lu, D. Jia, E. Ben-Jacob, H. Levine, and J. N. Onuchic. Interrogating the topological robustness of gene regulatory circuits by randomization. *PLOS Computational Biology*, 13(3):1–21, 03 2017.

[3] B. Novak, Z. Pataki, A. Ciliberto, and J. J. Tyson. Mathematical model of the cell division cycle of fission yeast. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 11(1):277–286, 2001.

Appendix: Code

June 17, 2021

Reproducibility: Although, all the codes and files are attached to this document, the best way to reproduce the results in this article is to acces the Github repo and read through the README file for better understanding of the usability of the functions and the pipelines to be used.

Github Repository Link given below:

`https://github.com/Souvadra/CH248_Project`

Topo file and RACIPE simulations:

Circuit Topology

Source	Target	Type
Start	Start	2
Start	SK	1
SK	SK	2
SK	Ste9	2
SK	Rum1	2
Ste9	Cdc2/Cdc13	2
Cdc2/Cdc13	Ste9	2
Ste9	Cdc2/Cdc13*	2
Cdc2/Cdc13*	Ste9	2
Rum1	Cdc2/Cdc13	2
Cdc2/Cdc13	Rum1	2
Rum1	Cdc2/Cdc13*	2
Cdc2/Cdc13*	Rum1	2
Cdc2/Cdc13	Wee1/Mik1	2
Cdc2/Cdc13	Cdc25	1
PP	PP	2
PP	Ste9	1
PP	Rum1	1
PP	Wee1/Mik1	1
PP	Cdc25	2
Slp1	PP	1
Slp1	Slp1	2
Slp1	Cdc2/Cdc13	2
Slp1	Cdc2/Cdc13*	2
Cdc2/Cdc13*	Slp1	1
Wee1/Mik1	Cdc2/Cdc13*	2
Cdc25	Cdc2/Cdc13*	1

Running RACIPE Download and compile RACIPE, copy the write the .topo file of the circuit in the same directory of RACIPE and then type:

```
./RACIPE <file>.topo -num_paras <number_of_parameter_sets> -num_ode
<number_of_initial_conditions_for_each_parameter_set> -threads <number_of_processors>
```

G/K Normlalization, Z-normalization:

```
function GK_normalization(path,components_num,external_signal)

%% Reading the paths and making string array of solution files ------------
% and parameters file separately ----------------------------------------

path = strrep(path,'\','/');
x = strcat(path,"/*solution_*.dat");
y = strcat(path,"/*parameters.dat");

F = dir(x);
F2 = dir(y);

%%-------------------------------------------------------------------------
%% Replacing backward slash with forward slashes  for parameters file -----

for i = 1:length(F2)

    s2 = strcat(F2(i).folder,"/",F2(i).name);
    str2(i,1) = strrep(s2,'\','/');
```

```
end

%%-------------------------------------------------------------------------
%% Reading the parameters file, remove all coloumns after the degradation -
% rate values, divide production rates coloumn by degradation rates coloumn
% and then remove all coloumns after the G/K values ----------------------

%external_signal = str2double(external_signal); %Souvadra's addition
%components_num = str2double(components_num);   %Souvadra's addition

F1 = dlmread(str2(1));
F1(:,2+2*(components_num+external_signal)+1:end) = [];

 for j = 3:(3+components_num-1)

    F1(:,j) = F1(:,j)./F1(:,j+components_num+external_signal);

 end


F1(:,2+components_num+1:end) = [];

%%-------------------------------------------------------------------------
%% Replacing backward slash with forward slashes  for solution files ------

str = strings(length(F),1);

for i = 1:length(F)

    s = strcat(F(i).folder,"/",F(i).name);
    str(i,1) = strrep(s,'\','/');

end

%%-------------------------------------------------------------------------
%% Reading the solution files into matrix and performing G/K normalization
% and further the concatenation of all solutions and calculating the ------
% z-scores for plotting the scatter diagram -------------------------------

Mn = zeros(1,components_num);

for i = 1:length(str)

    A = dlmread(str(i));
    B = A(:,1);
    A = 2.^A;
    a = size(A);

    for k = 1:a(1,1)

       for m = 1:length(F1)

       if B(k,1) == F1(m,1)

                for j = 3:components_num + external_signal:a(1,2)

                    for l = 1:components_num

                        A(k,j+l-1) = A(k,j+l-1)./F1(k,2+l);

                    end

                end
       end

       end

    end

    A = log2(A);
    A(:,1) = B;
    newstr = split(str(i,1),"_");
    size(newstr,1);
    new = strings(1);
    for i = 1:size(newstr,1)-1
        new = strcat(new,newstr(i,1),"_");
    end
```

```
        new = strcat(new,"gk_",newstr(size(newstr,1),1));
        dlmwrite(new,A,'delimiter','\t');
    end
    %%-------------------------------------------------------------------------
    end
```

Stability State Counting:

Function 1

```
function universal_stability_state_counter(p1,p2,p3,name)
%{
This function can be used to easily plot the numbers of different
stability states in a given ciruit, using the data from the triplicates and
plots the results in the form of a .fig file. This function is made to
handle larger RACIPE simulations will 20th- and 30th-stable solutions and more general
than that of standard 10th-stable solutions used otherwise.
%}
%% Call the helper functinos
    file1 = p1 ;
    file2 = p2 ;
    file3 = p3 ;
    [val1] = stateCounter(file1);
    [val2] = stateCounter(file2);
    [val3] = stateCounter(file3);
    % let me first finish the helper functions
%% Do the required calculatios
    allStates = unique([val1(:,2); val2(:,2); val3(:,2)]);
    val = zeros(length(allStates),1); % val is actually my y axis
    x = zeros(length(allStates),1);
    err = zeros(length(allStates),1);
    i = 0;
    for currState = allStates'
        i = i + 1;
        z = [];
        participants = 0;

        for j1=1:size(val1,1)
            if val1(j1,2) == currState
                z = [z; val1(j1,1)];
                participants = participants + 1;
            end
        end

        for j2=1:size(val2,1)
            if val2(j2,2) == currState
                z = [z; val2(j2,1)];
                participants = participants + 1;
            end
        end

        for j3=1:size(val3,1)
            if val3(j3,2) == currState
                z = [z; val3(j3,1)];
                participants = participants + 1;
            end
        end

        val(i,1) = sum(z)/participants;
        err(i,1) = std(z);
        x(i,1) = int8(currState);
    end

%% Its time to plot the data
    bar(x,val); hold on;
    er = errorbar(x,val,err,err);
    er.Color = [0 0 0];
    er.LineStyle = 'none';

    ylabel('Number of the stability states');
    titl = ['Circuit topology: ',name];
    title(titl);
    saveName = [name,'_stability_state_counts_full.fig'];
    savefig(saveName);
end
```

Function 2

```
%{
    This function outputs the percentage of RACIPE solutions belonging each
    stability state (obviously incorporating the triplicates) in the form of
    a .xls file.
%}
function output_table = MakeStabilityStateCounter(path,name)
%% Call the helper functions
vec1 = stabilityStateCounter(path + "/1");
vec2 = stabilityStateCounter(path + "/2");
vec3 = stabilityStateCounter(path + "/3");
%% take mean and std-dev
vec = [vec1 vec2, vec3];
vec = vec';
Mean = mean(vec);
Std = std(vec);
Mean = Mean';
Std = Std';
output_table = [Mean Std];
%% time for plotting the function
x_axis = ["mono", "bi", "tri", "other"];
x = 1:4;

figure
bar(x,Mean); hold on;
er = errorbar(x,Mean,Std,Std);
er.Color = [0 0 0];
er.LineStyle = 'none';

set(gca,'xticklabel',x_axis)
ylabel('Percentage Stability States');
Name = "Circuit topology: "+ string(name);
title(Name);

savefig(string(name) + "-stability-state-percentage.fig");
Name = string(name) + "stability-state-percentage" + ".xls";
writematrix(output_table,Name)
end
```

Function 3

```
function output_table = stabilityStateCounter(path)
%% Reading the data and arranging them properly
path = strrep(path,'\','/');
x = strcat(path,"/*solution_gk_*.dat");
solution_path_dir = dir(x);
solution_paths = strings(length(solution_path_dir),1);
for i = 1:length(solution_path_dir)

    s = strcat(solution_path_dir(i).folder,"/",solution_path_dir(i).name);
    solution_paths(i,1) = strrep(s,'\','/');

end
xxx = char(solution_paths(2));
while xxx(length(xxx)-5) == '1'
    solution_paths = [solution_paths; solution_paths(2)];
    solution_paths(2) = [];
    xxx = char(solution_paths(2));
end
xxx = char(solution_paths(3));
if xxx(length(xxx)-5) == '2'
    solution_paths = [solution_paths; solution_paths(3)];
    solution_paths(3) = [];
end
%% Select each stability state file and work the calculation separetely
regex_pattern = '\_\d+';
output_table = zeros(4,1);
for i = 1:length(solution_paths)
    match = regexp(solution_paths(i), regex_pattern, 'match');
    match = strsplit(match, '_');
    match = match(2);
    stability_state =  str2num(match);

    solution_values_all = dlmread(solution_paths(i));
    total_number = size(solution_values_all,1);

    if stability_state < 4
        output_table(stability_state,1) = total_number;
    else
```

```
        output_table(4) = output_table(4) + total_number;
    end
end
output_table = output_table./sum(output_table);
end
```

Function 4

```
function output_table = stabilityStateCounter(path)
%% Reading the data and arranging them properly
path = strrep(path,'\','/');
x = strcat(path,"/*solution_gk_*.dat");
solution_path_dir = dir(x);
solution_paths = strings(length(solution_path_dir),1);
for i = 1:length(solution_path_dir)

    s = strcat(solution_path_dir(i).folder,"/",solution_path_dir(i).name);
    solution_paths(i,1) = strrep(s,'\','/');

end
xxx = char(solution_paths(2));
while xxx(length(xxx)-5) == '1'
    solution_paths = [solution_paths; solution_paths(2)];
    solution_paths(2) = [];
    xxx = char(solution_paths(2));
end
xxx = char(solution_paths(3));
if xxx(length(xxx)-5) == '2'
    solution_paths = [solution_paths; solution_paths(3)];
    solution_paths(3) = [];
end
%% Select each stability state file and work the calculation separetely
regex_pattern = '\_\d+';
output_table = zeros(4,1);
for i = 1:length(solution_paths)
    match = regexp(solution_paths(i), regex_pattern, 'match');
    match = strsplit(match, '_');
    match = match(2);
    stability_state =  str2num(match);

    solution_values_all = dlmread(solution_paths(i));
    total_number = size(solution_values_all,1);

    if stability_state < 4
        output_table(stability_state,1) = total_number;
    else
        output_table(4) = output_table(4) + total_number;
    end
end
output_table = output_table./sum(output_table);
end
```

State Frequency Calculation:

Function 1

```
function [y, Fn1, Fn] = err_bar_maker(path,sol_num,components_num,ex_sig)

%% Reading the data into matrices and arraging them -----------------------

C = dlmread(path);
C(:,1:2) = [];

for i = 1:sol_num
    C(:,(i*components_num)+1:(i*components_num)+ex_sig) = [];
end

Cm = mean(C);
Csd = std(C);
Cn = (C - Cm)./(Csd);

a = size(Cn);
Cn = Cn > 0 ;

%%-------------------------------------------------------------------------
%% Determining the phase of the solution obtained ---------------
```

```
% A1 = [0 0 0];
% A2 = [1 0 0];
% A3 = [0 1 0];
% A4 = [0 0 1];
% A5 = [1 1 0];
% A6 = [1 0 1];
% A7 = [0 1 1];
% A8 = [1 1 1];

v1 = repmat(0,components_num,1);
v2 = repmat(1,components_num,1);
v = horzcat(v1',v2');

n_sh = components_num;
lim_sh = 1;
for ii=(1:n_sh)
    lim_sh = lim_sh + (2^(ii-1));
end

A3 = zeros(1,n_sh);
for ii=(1:lim_sh-1)
    c_sh = ii;
    vector_sh = [];
    while c_sh > 0
        if c_sh == 1
            vector_sh = [vector_sh, c_sh];
            c_sh = 0;
        else
            rem_sh = mod(c_sh,2);
            vector_sh = [vector_sh, rem_sh];
            c_sh = (c_sh - rem_sh) / 2;
        end
    end
    l_sh = n_sh - length(vector_sh);
    for jjj=(1:l_sh)
        vector_sh = [vector_sh, 0];
    end
    vector_sh = fliplr(vector_sh);
    A3 = [A3 ; vector_sh];
end

 a3 = size(A3);

Dn = zeros(a(1,1),components_num*sol_num);

for i=1:components_num:components_num*sol_num
    for j = 1:a(1,1)
        for k = 1:a3(1,1)
            if isequal(Cn(j,i:i+components_num-1),A3(k,:))
                Dn(j,i)=k;
            end
        end
    end
end

%%-------------------------------------------------------------------------
%% Sorting the solutions for ease of counting ----------------------------

Zn = zeros(a(1,1),sol_num);

for i = 1:sol_num
    Zn(:,i) = Dn(:,((components_num*(i-1)+1)));
end

Zn = sort(Zn,2);
Zn = sortrows(Zn);
%%-------------------------------------------------------------------------
%% Explicitly putting the naming system (Binary-ish) - Souvadra
naming = [];
L = size(A3);
L = L(1,1);
B = size(A3);
B = B(1,2);
for i=(1:L)
    sSH = "";
    for j=(1:B)
        sSH = sSH + A3(i,j);
    end
```

```
    naming = [naming; sSH];
end

Zn_naming = [];
BB = size(Zn);
BB = BB(1,2);
for i=(1:length(Zn))
    eSH = [];
    for j=(1:BB)
        aSH = naming(Zn(i,j));
        eSH = [eSH aSH];
    end
    Zn_naming = [Zn_naming; eSH];
end

BBB = size(Zn_naming);
BBB = BBB(1,2);
for i=(1:length(Zn_naming))
    for j=(2:BBB)
    Zn_naming(i,1) = Zn_naming(i,1)+Zn_naming(i,j);
    end
end
Zn_naming = Zn_naming(:,1);
%Fn_naming = unique(Zn_naming);

%%-------------------------------------------------------------------------
%% Making the X-tick labels ---------------------------------------------

% writing the matrix to txt file, then remove delimiters followed by
% reading the txt file so that the rows are horizontally combines i.e. the
% three coloumn single digit elements are converted to a single coloumn
% three digit number. Finally the vector is converted to a string array.

dlmwrite('data.txt',Zn,'delimiter','');
Fn = dlmread('data.txt');
Var_SH = Fn;
delete data.txt;
Fn = [Fn, Zn_naming]; % Souvadra's addition
Fn = unique(Fn,'rows');
f = size(Fn);
Fn1 = Fn(:,1); % Souvadra's addition

%%-------------------------------------------------------------------------
%% Souvadra's alternative to the happeing issue
y = zeros(f(1,1),1);
for i=(1:length(Var_SH))
    for j=(1:f(1,1))
        if Var_SH(i,1) == double(Fn(j,1))
            y(j,1) = y(j,1) + 1;
        end
    end
end
y = y ./ sum(y); % giving percentage result rather than the full value
%%-------------------------------------------------------------------------
end
```

Function 2

```
function matrix_sh = make_an_errorbar_conditions_apply(p1,p2,p3,sol_num,
components_num,ext_signals_num,condition,name)

%% Reading all the solution files into matrices ---------------------------

file1 = p1 ;
file2 = p2 ;
file3 = p3 ;

[X1, F1, F1sh] = err_bar_maker(file1,sol_num,components_num,ext_signals_num);
[X2, F2, F2sh] = err_bar_maker(file2,sol_num,components_num,ext_signals_num);
[X3, F3, F3sh] = err_bar_maker(file3,sol_num,components_num,ext_signals_num);

%%-------------------------------------------------------------------------
%% Making all data vectors of same length --------------------------------

maxlen = [length(X1) length(X2) length(X3)];
maxlen = max(maxlen);
X1(end+1:maxlen) = 0;
X2(end+1:maxlen) = 0;
```

```
X3(end+1:maxlen) = 0;
F1(end+1:maxlen) = "0";
F2(end+1:maxlen) = "0";
F3(end+1:maxlen) = "0";

%%-------------------------------------------------------------------------
%% Making the Categories ------------------------------------------------

C = unique([F1;F2;F3]);
C = double(C);
c = size(C);

%%-------------------------------------------------------------------------
%% Make the data from different simulations match based on the category ---

data = zeros(c(1,1),3);

for i = 1:c(1,1)
    for j = 1:maxlen

        if C(i,1)==double(F1(j,1))
            data(i,1) = X1(j,1);
        end

    end
end
for i = 1:c(1,1)
    for j = 1:maxlen

        if C(i,1)==double(F2(j,1))
            data(i,2) = X2(j,1);
        end

    end
end
for i = 1:c(1,1)
    for j = 1:maxlen

        if C(i,1)==double(F3(j,1))
            data(i,3) = X3(j,1);
        end

    end
end

data_mod = data;

%%-------------------------------------------------------------------------
%% Remove the rows containing all zeros from the C and data matrix --------

a = sum(data,2);
a1 = zeros(length(a),1);

for i = 1:c(1,1)

    if a(i) == 0
        a1(i) = i;
    end

end

a1 = a1(a1 ~= 0);
data_mod = horzcat(data_mod,C);
data_mod(a1,:) = [] ;

data_mod = sortrows(data_mod,'descend');

%%-------------------------------------------------------------------------
%% Calculation errors required for barplot -------------------------------

data_final = data_mod(:,1:3)';
mean_data = mean(data_final);
std_data = std(data_final);

cnn = string(data_mod(:,4));

%%-------------------------------------------------------------------------
%% Conditioning of the data ----------------------------------------------
```

```
% required_indexes = find(mean_data>50);
B = mean_data > condition ;
new_mean_data = mean_data(B);
new_std_data = std_data(B);
new_cnn = cnn(B);


%%-------------------------------------------------------------------------
%% Plotting the bar plot with errorbars ----------------------------------
% Souvadra is commentinf this block of code
cn = categorical(new_cnn);

grid on
bar(cn,new_mean_data);
hold on

er = errorbar(cn,new_mean_data,new_std_data);
er.Color = [0 0 0];
er.LineStyle = 'none';

hold off

%%-------------------------------------------------------------------------
%% Souvadra's modificaiton, Let's not plot the data and store it instead in
%% .csv file
%cn = categorical(new_cnn)
QSH = [F1sh; F2sh; F3sh];
QSH = unique(QSH,'rows');
Map = containers.Map(QSH(:,1),QSH(:,2));
nameSH = strings(length(new_mean_data),1);
for i=(1:length(new_mean_data)) % THis was initially 'length(QSH)'
    variable = {new_cnn(i,1)};
    mappedTo = string(values(Map,variable));
    mappedTo = "<" + mappedTo + ">";
    %nameSH = [nameSH ; mappedTo];
    nameSH(i,1) = mappedTo;
end

Name = name + ".xls";
matrix_sh = [new_cnn, nameSH, new_mean_data', new_std_data']
writematrix(matrix_sh,Name)
%%-------------------------------------------------------------------------
end
```

Plotting the Dynamics of the network:

Function 1

```
% A simple representation of Hill equation
function H = hill(X,X0,lambda,n)
% H+
H1 = ((X/X0)^n)/(1+(X/X0)^n);
% H-
H2 = 1/(1+(X/X0)^n);

% Hill function =  H- plus (lambda)* H+
H = H2 + (lambda)*H1;
if lambda > 1
     H=H/lambda;
end

end

% function H = hill(X,X0,lambda,n)
% H = lambda + (1.0 - lambda) * (1.0/(1.0 + (X/X0)^n));
% if lambda > 1
%     H = H / lambda;
% end
%
% if H < 0
%     H
% end
% end
```

Function 2

```
function [prs_file_info, parameters] = parameter_generator(path)
```

```
%% Reading the topo file and generating the variables ----------------------

path = strrep(path,'\','/');
path = strcat(path,"/*.prs");
file_dir = dir(path);
prs_path = strcat(file_dir(1).folder,"/",file_dir(1).name);
prs_path = strrep(prs_path,'\','/');
prs_file_info = tdfread(prs_path);

parameters = prs_file_info.Parameter ;

%%-------------------------------------------------------------------------
```

Function 3

```
function parameter_sets_for_simulating = parameter_set_segregator(path,
components_num,external_signal,sol_num,categories)

[topo_file_info, parameter_names] = parameter_generator(path);

path = strrep(path,'\','/');
x = strcat(path,"/*solution_gk_*.dat");
y = strcat(path,"/*parameters.dat");

solution_path_dir = dir(x);
parameter_value_file_dir = dir(y);

paramater_value_path = strcat(parameter_value_file_dir(1).folder,"/",
parameter_value_file_dir(1).name);
paramater_value_path = strrep(paramater_value_path,'\','/');
parameter_values = readtable(paramater_value_path);

%% The paths copied will have backward slashes so have to be replaced with
% backward slashes ----------------------------------------------------------

solution_paths = strings(length(solution_path_dir),1);

for i = 1:length(solution_path_dir)

    s = strcat(solution_path_dir(i).folder,"/",solution_path_dir(i).name);
    solution_paths(i,1) = strrep(s,'\','/');

end

%%-------------------------------------------------------------------------

%% reordering the solution_paths to keep solution_10 at last rather than at the
second place
% -- Souvadra
xxx = char(solution_paths(2));
while xxx(length(xxx)-5) == '1'
    solution_paths = [solution_paths; solution_paths(2)];
    solution_paths(2) = [];
    xxx = char(solution_paths(2));
end
xxx = char(solution_paths(3));
if xxx(length(xxx)-5) == '2'
    solution_paths = [solution_paths; solution_paths(3)];
    solution_paths(3) = [];
end
%%

if sol_num == 1
    solution_values_all = dlmread(solution_paths(sol_num,1));
else, solution_values_all = dlmread(solution_paths(sol_num,1));
end

solution_values_all_copy = solution_values_all;
solution_values_all(:,1:2) = [];

for i = 1:sol_num

    solution_values_all(:,(i*components_num)+1:(i*components_num)+external_signal) = [];

end

solution_values_all_mean = mean(solution_values_all);
```

```
solution_values_all_sd = std(solution_values_all);
solution_values_all_normalized = (solution_values_all -
solution_values_all_mean)./(solution_values_all_sd);

a = size(solution_values_all_normalized);

%%-------------------------------------------------------------------------
%% Making the elements of the matrix binary to represent the states -------

% for i = 1:components_num*sol_num
%     for j = 1:a(1,1)
%         if solution_values_all_normalized(j,i) < 0
%             solution_values_all_normalized(j,i) = 0;
%         else, solution_values_all_normalized(j,i) = 1;
%         end
%     end
% end

solution_values_all_normalized = solution_values_all_normalized > 0;

%%-------------------------------------------------------------------------
%% Determining the phase of the solution obtained ---------------

v1 = repmat(0,components_num,1);
v2 = repmat(1,components_num,1);
v = horzcat(v1',v2');

% A1 = nchoosek(v,components_num);
% A1 = unique(A1,'rows');
% A1size = size(A1);

% This code-block is written by Souvadra
n_sh = components_num;
lim_sh = 1;
for ii=(1:n_sh)
    lim_sh = lim_sh + (2^(ii-1));
end

A3 = zeros(1,n_sh);
for ii=(1:lim_sh-1)
    c_sh = ii;
    vector_sh = [];
    while c_sh > 0
        if c_sh == 1
            vector_sh = [vector_sh, c_sh];
            c_sh = 0;
        else
            rem_sh = mod(c_sh,2);
            vector_sh = [vector_sh, rem_sh];
            c_sh = (c_sh - rem_sh) / 2;
        end
    end
    l_sh = n_sh - length(vector_sh);
    for jjj=(1:l_sh)
        vector_sh = [vector_sh, 0];
    end
    vector_sh = fliplr(vector_sh);
    A3 = [A3 ; vector_sh];
end

% A3 = zeros(1,components_num);
%
% for i = 1:A1size(1,1)
%
%     A2 = perms(A1(i,:));
%     A2 = unique(A2,'rows');
%     A3 = vertcat(A3,A2);
%
% end
%
% A3(1,:) = [];
a3 = size(A3);

solution_values_all_categorized = zeros(a(1,1),components_num*sol_num);

for i=1:components_num:components_num*sol_num
    for j = 1:a(1,1)
        for k = 1:a3(1,1)
```

```
            if isequal(solution_values_all_normalized(j,i:i+components_num-1),A3(k,:))
                solution_values_all_categorized(j,i)=k;
            end
        end
    end
end

clear A1 A1size A2 A3 a2 a3 v v1 v2 x y i j k s

%%-------------------------------------------------------------------------
%% Sorting the solutions for ease of counting ----------------------------

solutions_all_categorized_no_null = zeros(a(1,1),sol_num);

for i = 1:sol_num

    solutions_all_categorized_no_null(:,i) = solution_values_all_categorized(:,((
        components_num*(i-1)+1)));

end

% Zn = sort(Zn,2);
% Zn = sortrows(Zn);

clear i

%%-------------------------------------------------------------------------
%% Making the X-tick labels ----------------------------------------------

% writing the matrix to txt file, then remove delimiters followed by
% reading the txt file so that the rows are horizontally combines i.e. the
% three coloumn single digit elements are converted to a single coloumn
% three digit number. Finally the vector is converted to a string array.

dlmwrite('data.txt',solutions_all_categorized_no_null,'delimiter','');
solutions_all_categorized_final = dlmread('data.txt');
delete data.txt;
% Fn = unique(Fn);
% f = size(Fn);
% Fnn = string(Fn);

%%-------------------------------------------------------------------------

solutions_all_categorized_final = horzcat(solutions_all_categorized_final,
solution_values_all_copy(:,1));

if sol_num == 1
    for i = 1:size(solutions_all_categorized_final,1)
        if solutions_all_categorized_final(i,1)==categories(1,1)
            solution_indices_required(i) = solutions_all_categorized_final(i,2);
        end
    end
% if sol_num == 1
%     for i = 1:size(solutions_all_categorized_final,1)
%         if solutions_all_categorized_final(i,1)==categories(1,1)
%             solution_indices_required(i) = solutions_all_categorized_final(i,2);
%         end
%     end
elseif sol_num == 2
    for i = 1:size(solutions_all_categorized_final,1)
        if solutions_all_categorized_final(i,1)==categories(1,1) ||
            solutions_all_categorized_final(i,1)==categories(1,2)
            solution_indices_required(i) = solutions_all_categorized_final(i,2);
        end
    end
elseif sol_num == 3
    for i = 1:size(solutions_all_categorized_final,1)
        if solutions_all_categorized_final(i,1)==categories(1,1) ||
            solutions_all_categorized_final(i,1)==categories(1,2) ||
                solutions_all_categorized_final(i,1)==categories(1,3) ||
                    solutions_all_categorized_final(i,1)==categories(1,4) ||
                        solutions_all_categorized_final(i,1)==categories(1,5) ||
                            solutions_all_categorized_final(i,1)==categories(1,6)
            solution_indices_required(i) = solutions_all_categorized_final(i,2);
        end
    end
end
```

```
solution_indices_required = solution_indices_required';
solution_indices_required = nonzeros(solution_indices_required);

%%-------------------------------------------------------------------------

% parameters = zeros(size(solution_indices_required,1)+1,size(parameter_names,2));

for i = 1:size(parameter_names,1)
    parameters(1,i) = convertCharsToStrings(parameter_names(i,1:size(parameter_names,2)));
end

parameter_values_newfilenames = cellstr(parameters);

parameter_values.Properties.VariableNames = ["S_no" "States_number"
parameter_values_newfilenames];

clear i

%%-------------------------------------------------------------------------

% parameter_sets_for_simulating = table(size(solution_indices_required,1),
size(parameter_values,2));
xxx = parameter_values.S_no;
%LLL = size(parameter_values);
for i = 1:numel(solution_indices_required)
    for j = 1:numel(parameter_values.Prod_of_A) % Souvadra's commenting

    %for j = 1:LLL(1,1)
     if solution_indices_required(i,1) == xxx(j)
         parameter_sets_for_simulating(i,:)=parameter_values(j,:);
     end

   end
end

%%-------------------------------------------------------------------------

end
```

Function 4

```
function dydt = dynamics_simulation(t,y,parameter_set)

dydt = zeros(10,1);
Start = y(1);
SK = y(2);
Ste9 = y(3);
Rum1 = y(4);
Cdc2byCdc13 = y(5);
Cdc2byCdc13Star = y(6);
Wee1byMik1 = y(7);
Cdc25 = y(8);
PP = y(9);
Slp1 = y(10);


ga = parameter_set.("Prod_of_Start               ");
gb = parameter_set.("Prod_of_SK                  ");
gc = parameter_set.("Prod_of_Ste9                ");
gd = parameter_set.("Prod_of_Rum1                ");
ge = parameter_set.("Prod_of_Cdc2/Cdc13          ");
gf = parameter_set.("Prod_of_Cdc2/Cdc13*         ");
gg = parameter_set.("Prod_of_Wee1/Mik1           ");
gh = parameter_set.("Prod_of_Cdc25               ");
gi = parameter_set.("Prod_of_PP                  ");
gj = parameter_set.("Prod_of_Slp1                ");

ka = parameter_set.("Deg_of_Start                ");
kb = parameter_set.("Deg_of_SK                   ");
kc = parameter_set.("Deg_of_Ste9                 ");
kd = parameter_set.("Deg_of_Rum1                 ");
ke = parameter_set.("Deg_of_Cdc2/Cdc13           ");
kf = parameter_set.("Deg_of_Cdc2/Cdc13*          ");
kg = parameter_set.("Deg_of_Wee1/Mik1            ");
kh = parameter_set.("Deg_of_Cdc25                ");
ki = parameter_set.("Deg_of_PP                   ");
kj = parameter_set.("Deg_of_Slp1                 ");
```

```
H_Start_Start = hill(Start, parameter_set.("Trd_of_StartToStart          "),
parameter_set.("Inh_of_StartToStart          "),
parameter_set.("Num_of_StartToStart          "));

H_SK_SK = hill(SK, parameter_set.("Trd_of_SKToSK                "),
parameter_set.("Inh_of_SKToSK                "),
parameter_set.("Num_of_SKToSK                "));
H_Start_SK = hill(Start, parameter_set.("Trd_of_StartToSK             "),
parameter_set.("Act_of_StartToSK             "),
parameter_set.("Num_of_StartToSK             "));

H_Cdc2byCdc13Star_Ste9 = hill(Cdc2byCdc13Star,
parameter_set.("Trd_of_Cdc2/Cdc13*ToSte9     "),
parameter_set.("Inh_of_Cdc2/Cdc13*ToSte9     "),
parameter_set.("Num_of_Cdc2/Cdc13*ToSte9     "));
H_SK_Ste9 = hill(SK, parameter_set.("Trd_of_SKToSte9              "),
parameter_set.("Inh_of_SKToSte9              "),
parameter_set.("Num_of_SKToSte9              "));
H_Cdc2byCdc13_Ste9 = hill(Cdc2byCdc13,
parameter_set.("Trd_of_Cdc2/Cdc13ToSte9      "),
parameter_set.("Inh_of_Cdc2/Cdc13ToSte9      "),
parameter_set.("Num_of_Cdc2/Cdc13ToSte9      "));
H_PP_Ste9 = hill(PP, parameter_set.("Trd_of_PPToSte9              "),
parameter_set.("Act_of_PPToSte9              "),
parameter_set.("Num_of_PPToSte9              "));

H_SK_Rum1 = hill(SK, parameter_set.("Trd_of_SKToRum1              "),
parameter_set.("Inh_of_SKToRum1              "),
parameter_set.("Num_of_SKToRum1              "));
H_Cdc2byCdc13_Rum1 = hill(Cdc2byCdc13,
parameter_set.("Trd_of_Cdc2/Cdc13ToRum1      "),
parameter_set.("Inh_of_Cdc2/Cdc13ToRum1      "),
parameter_set.("Num_of_Cdc2/Cdc13ToRum1      "));
H_Cdc2byCdc13Star_Rum1 = hill(Cdc2byCdc13Star,
parameter_set.("Trd_of_Cdc2/Cdc13*ToRum1     "),
parameter_set.("Inh_of_Cdc2/Cdc13*ToRum1     "),
parameter_set.("Num_of_Cdc2/Cdc13*ToRum1     "));
H_PP_Rum1 = hill(PP, parameter_set.("Trd_of_PPToRum1              "),
parameter_set.("Act_of_PPToRum1              "),
parameter_set.("Num_of_PPToRum1              "));

H_Rum1_Cdc2byCdc13 = hill(Rum1, parameter_set.("Trd_of_Rum1ToCdc2/Cdc13      "),
parameter_set.("Inh_of_Rum1ToCdc2/Cdc13      "),
parameter_set.("Num_of_Rum1ToCdc2/Cdc13      "));
H_Ste9_Cdc2byCdc13 = hill(Ste9, parameter_set.("Trd_of_Ste9ToCdc2/Cdc13      "),
parameter_set.("Inh_of_Ste9ToCdc2/Cdc13      "),
parameter_set.("Num_of_Ste9ToCdc2/Cdc13      "));
H_Slp1_Cdc2byCdc13 = hill(Slp1, parameter_set.("Trd_of_Slp1ToCdc2/Cdc13      "),
parameter_set.("Inh_of_Slp1ToCdc2/Cdc13      "),
parameter_set.("Num_of_Slp1ToCdc2/Cdc13      "));

H_Cdc2byCdc13_Wee1byMik1 = hill(Cdc2byCdc13,
parameter_set.("Trd_of_Cdc2/Cdc13ToWee1/Mik1 "),
parameter_set.("Inh_of_Cdc2/Cdc13ToWee1/Mik1 "),
parameter_set.("Num_of_Cdc2/Cdc13ToWee1/Mik1 "));
H_PP_Wee1byMik1 = hill(PP, parameter_set.("Trd_of_PPToWee1/Mik1         "),
parameter_set.("Act_of_PPToWee1/Mik1         "),
 parameter_set.("Num_of_PPToWee1/Mik1         "));

H_Cdc2byCdc13_Cdc25 = hill(Cdc2byCdc13, parameter_set.("Trd_of_Cdc2/Cdc13ToCdc25     "),
parameter_set.("Act_of_Cdc2/Cdc13ToCdc25     "),
parameter_set.("Num_of_Cdc2/Cdc13ToCdc25     "));
H_PP_Cdc25 = hill(PP, parameter_set.("Trd_of_PPToCdc25             "),
parameter_set.("Inh_of_PPToCdc25             "),
parameter_set.("Num_of_PPToCdc25             "));

H_Cdc2byCdc13Star_Slp1 = hill(Cdc2byCdc13Star,
parameter_set.("Trd_of_Cdc2/Cdc13*ToSlp1     "),
parameter_set.("Act_of_Cdc2/Cdc13*ToSlp1     "),
parameter_set.("Num_of_Cdc2/Cdc13*ToSlp1     "));
H_Slp1_Slp1 = hill(Slp1, parameter_set.("Trd_of_Slp1ToSlp1            "),
parameter_set.("Inh_of_Slp1ToSlp1            "),
parameter_set.("Num_of_Slp1ToSlp1            "));

H_PP_PP = hill(PP, parameter_set.("Trd_of_PPToPP                "),
parameter_set.("Inh_of_PPToPP                "),
parameter_set.("Num_of_PPToPP                "));
H_Slp1_PP = hill(Slp1, parameter_set.("Trd_of_Slp1ToPP              "),
```

```
    parameter_set.("Act_of_Slp1ToPP              "),
    parameter_set.("Num_of_Slp1ToPP              "));

    H_Slp1_Cdc2byCdc13Star = hill(Slp1,
    parameter_set.("Trd_of_Slp1ToCdc2/Cdc13*     "),
    parameter_set.("Inh_of_Slp1ToCdc2/Cdc13*     "),
    parameter_set.("Num_of_Slp1ToCdc2/Cdc13*     "));
    H_Wee1byMik1_Cdc2byCdc13Star = hill(Wee1byMik1,
    parameter_set.("Trd_of_Wee1/Mik1ToCdc2/Cdc13*"),
    parameter_set.("Inh_of_Wee1/Mik1ToCdc2/Cdc13*"),
    parameter_set.("Num_of_Wee1/Mik1ToCdc2/Cdc13*"));
    H_Cdc25_Cdc2byCdc13Star = hill(Cdc25, parameter_set.("Trd_of_Cdc25ToCdc2/Cdc13*    "),
    parameter_set.("Act_of_Cdc25ToCdc2/Cdc13*    "),
    parameter_set.("Num_of_Cdc25ToCdc2/Cdc13*    "));
    H_Ste9_Cdc2byCdc13Star = hill(Ste9, parameter_set.("Trd_of_Ste9ToCdc2/Cdc13*     "),
    parameter_set.("Inh_of_Ste9ToCdc2/Cdc13*     "),
    parameter_set.("Num_of_Ste9ToCdc2/Cdc13*     "));
    H_Rum1_Cdc2byCdc13Star = hill(Rum1, parameter_set.("Trd_of_Rum1ToCdc2/Cdc13*     "),
    parameter_set.("Inh_of_Rum1ToCdc2/Cdc13*     "),
    parameter_set.("Num_of_Rum1ToCdc2/Cdc13*     "));


    dydt(1) = ga * H_Start_Start - ka*Start;
    dydt(2) = gb * H_Start_SK * H_SK_SK - kb*SK;
    dydt(3) = gc * H_Cdc2byCdc13Star_Ste9 * H_SK_Ste9 * H_Cdc2byCdc13_Ste9 * H_PP_Ste9 - kc*Ste9;
    dydt(4) = gd * H_SK_Rum1 * H_Cdc2byCdc13_Rum1 * H_Cdc2byCdc13Star_Rum1 * H_PP_Rum1 - kd*Rum1;
    dydt(5) = ge * H_Rum1_Cdc2byCdc13 * H_Ste9_Cdc2byCdc13 * H_Slp1_Cdc2byCdc13 - ke*Cdc2byCdc13;
    dydt(6) = gf * H_Slp1_Cdc2byCdc13Star * H_Wee1byMik1_Cdc2byCdc13Star * H_Cdc25_Cdc2byCdc13Star *
            H_Ste9_Cdc2byCdc13Star * H_Rum1_Cdc2byCdc13Star - kf*Cdc2byCdc13Star;
    dydt(7) = gi * H_Cdc2byCdc13_Wee1byMik1 * H_PP_Wee1byMik1 - ki*Wee1byMik1;
    dydt(8) = gj *H_Cdc2byCdc13_Cdc25 * H_PP_Cdc25 - kj*Cdc25;
    dydt(9) = gg * H_PP_PP * H_Slp1_PP - kg*PP;
    dydt(10) = gh * H_Cdc2byCdc13Star_Slp1 * H_Slp1_Slp1 - kh*Slp1;
    %dydt
end
```

Main Function of this section

```
    % ODE Solver RACIPE solution files
    clear
    path = '../data/fission_yeast_cc_normal/1';
    components_num = 10;
    external_signal = 0;
    sol_num = 10;

    %% Finding the solution files from the given paths
    [topo_file_info, parameter_names] = parameter_generator(path);

    path = strrep(path,'\','/');
    x = strcat(path,"/*solution_gk_*.dat");
    y = strcat(path,"/*parameters.dat");

    solution_path_dir = dir(x);
    parameter_value_file_dir = dir(y);

    paramater_value_path = strcat(parameter_value_file_dir(1).folder,"/",parameter_value_file_dir(1).name);
    paramater_value_path = strrep(paramater_value_path,'\','/');
    parameter_values = readtable(paramater_value_path);


    solution_paths = strings(length(solution_path_dir),1);

    for i = 1:length(solution_path_dir)

        s = strcat(solution_path_dir(i).folder,"/",solution_path_dir(i).name);
        solution_paths(i,1) = strrep(s,'\','/');

    end
    %% Extracting the parameter set files I want from the parameter_values table
    for i = 1:size(parameter_names,1)
        parameters(1,i) = convertCharsToStrings(parameter_names(i,1:size(parameter_names,2)));
    end

    parameter_values_newfilenames = cellstr(parameters);

    parameter_values.Properties.VariableNames = ["S_no" "States_number" parameter_values_newfilenames];

    clear i
```

```
j = 1;
for i=1:size(parameter_values,1)
    if parameter_values.States_number(i) == sol_num
        parameter_sets_for_simulating(j,:) = parameter_values(i,:);
        j=j+1;
    end
end

%% Now its time to solve ans ODE and plot the solutions
run_time = 0:1:100; % run time for ode solver
num_initials = 10; %100;

for ii = 22:42   %size(parameter_sets_for_simulating,1)
    ga = parameter_sets_for_simulating.("Prod_of_Start               ")(ii);
    gb = parameter_sets_for_simulating.("Prod_of_SK                  ")(ii);
    gc = parameter_sets_for_simulating.("Prod_of_Ste9                ")(ii);
    gd = parameter_sets_for_simulating.("Prod_of_Rum1                ")(ii);
    ge = parameter_sets_for_simulating.("Prod_of_Cdc2/Cdc13          ")(ii);
    gf = parameter_sets_for_simulating.("Prod_of_Cdc2/Cdc13*         ")(ii);
    gg = parameter_sets_for_simulating.("Prod_of_Wee1/Mik1           ")(ii);
    gh = parameter_sets_for_simulating.("Prod_of_Cdc25               ")(ii);
    gi = parameter_sets_for_simulating.("Prod_of_PP                  ")(ii);
    gj = parameter_sets_for_simulating.("Prod_of_Slp1                ")(ii);

    ka = parameter_sets_for_simulating.("Deg_of_Start                ")(ii);
    kb = parameter_sets_for_simulating.("Deg_of_SK                   ")(ii);
    kc = parameter_sets_for_simulating.("Deg_of_Ste9                 ")(ii);
    kd = parameter_sets_for_simulating.("Deg_of_Rum1                 ")(ii);
    ke = parameter_sets_for_simulating.("Deg_of_Cdc2/Cdc13           ")(ii);
    kf = parameter_sets_for_simulating.("Deg_of_Cdc2/Cdc13*          ")(ii);
    kg = parameter_sets_for_simulating.("Deg_of_Wee1/Mik1            ")(ii);
    kh = parameter_sets_for_simulating.("Deg_of_Cdc25                ")(ii);
    ki = parameter_sets_for_simulating.("Deg_of_PP                   ")(ii);
    kj = parameter_sets_for_simulating.("Deg_of_Slp1                 ")(ii);

   figure
   for jj = 1:num_initials
       [ii,jj]
       Startjj = (ga/ka)*(1+rand);
       SKjj = (gb/kb)*(rand);
       Ste9jj = (gc/kc)*(1+rand);
       Rum1jj = (gd/kd)*(1+rand);
       Cdc2byCdc13jj = (ge/ke)*(rand);
       Cdc2byCdc13Starjj = (gf/kf)*(rand);
       Wee1byMik1jj = (gg/kg)*(1+rand);
       Cdc25jj = (gh/kh)*(rand);
       PPjj = (gi/ki)*(rand);
       Slp1jj = (gj/kj)*(rand);

       I = [Startjj SKjj Ste9jj Rum1jj Cdc2byCdc13jj Cdc2byCdc13Starjj Wee1byMik1jj Cdc25jj PPjj Slp1jj];

       [t,y] = ode45(@(t,y)dynamics_simulation(t,y,parameter_sets_for_simulating(ii,:)),run_time,I);
       markers = {'k.-','b.-','r.-','k*-','b*-','k--','b--','r--','c--','m--'};
       for idx=1:size(y,2)
           plot(t,y(:,idx),markers{idx}); hold on
       end
   end
   legend('Start','SK','Ste9','Rum1','Cdc2/Cdc13','Cdc2/Cdc13*','Wee1/Mik1','Cdc25','PP','Slp1');
   hold off
end
```