

Apendix I: Thesis alignment with the Sustainable Development Goals (SDGs)

SDG	High	Medium	Low	Not applicable
SDG 1. No poverty				✓
SDG 2. Zero hunger				✓
SDG 3. Good health and well-being			✓	
SDG 4. Quality education				✓
SDG 5. Gender equality				✓
SDG 6. Clean water and sanitation				✓
SDG 7. Affordable and clean energy				✓
SDG 8. Decent work and economic growth				✓
SDG 9. Industry, innovation and infrastructure	✓			
SDG 10. Reduced inequalities				✓
SDG 11. Sustainable cities and communities				✓
SDG 12. Responsible consumption and production				✓
SDG 13. Climate action			✓	
SDG 14. Life below water				✓
SDG 15. Life on land			✓	
SDG 16. Peace, justice and strong institutions				✓
SDG 17. Partnership for the goals				✓

- SDG 9 - Industry, Innovation & Infrastructure:** This project not only is developed around the DBTL cycle context but also applies the synthetic biology methodology and values. The development of computational studies and a mathematical model that represent the behaviour of the biosensor embodies innovation and the application of technological tools in order to do so. This same idea applied in this project can be used in other industrial sectors, such as the pharmaceutical, where the

use of genetic twins is habitual. Thus, the successful implementation of algorithms and genetic twins in order to model our biosensor can stimulate the application of these into other fields, thus fostering industrial innovation through the application of tools that enable scalable and cost-efficient production.

Appendix II: Matlab scripts

Datagenerating file

In this file, the function Datagenerating is created, which is responsible for performing calibrations (ODblank and Fblank) and estimating metrics such as MEFL, MEFLcell, Particles and the $\mu(t)$ function.

```
1 function out = Datagenerating(evol, m, startpoint)
2
3 stepTime = evol.time_OD(2) - evol.time_OD(1); % Interval between time points in minutes
4
5 INDUC = char(evol.medios.strains(m).data.Inducc);
6 if size(INDUC, 1) > 1
7     inductions = str2num(INDUC(:, :));
8     inductions(1) = 0.00001; % Change the first induction, which is 0, to a very small value
9 end
10
11 for j = 1:size(INDUC, 1)
12
13     % Calculate OD blank + NAR for each INDUC
14     OD_blank_In = evol.medios.OD(startpoint:end, j);
15
16     % Calculate F blank + NAR for each INDUC
17     F_blank_In = evol.medios.F(startpoint:end, j);
18
19     % Calculate calibrated OD and Fluorescence
20     ODblank = evol.medios.strains(m).data(j).OD(startpoint:end, :) - OD_blank_In;
21     Fblank = evol.medios.strains(m).data(j).F(startpoint:end, :) - F_blank_In;
22
23     % Check the number of replicas for calibration
24     if ~isequal(size(ODblank), size(Fblank))
25         error = size(ODblank, 2) - size(Fblank, 2);
26         if error < 0
27             ODblank = [ODblank, ODblank(:, 1) .* ones(length(ODblank), -error)];
28         else
29             Fblank = [Fblank, Fblank(:, 1) .* ones(length(Fblank), error)];
30         end
31     end
32 end
```

```

1  % Save calibrated data
2  evol.medios.strains(m).data(j).ODblank = ODblank;
3  evol.medios.strains(m).data(j).OD_blank_In = OD_blank_In;
4  evol.medios.strains(m).data(j).Fblank = Fblank;
5  evol.medios.strains(m).data(j).F_blank_In = F_blank_In;
6
7  [R, Rlog, Particles, MEF] = Calibrate_MEFL_GFPmut3b(ODblank, Fblank);
8
9  evol.medios.strains(m).data(j).Particles = Particles;
10 evol.medios.strains(m).data(j).MEFL = MEF;
11 evol.medios.strains(m).data(j).MEFLcell = R;
12 evol.medios.strains(m).data(j).LogMEFcell = Rlog;
13
14 % Growth rate
15 actual_OD = mean(evol.medios.strains(m).data(j).ODblank, 2);
16 actual_PartI = mean(evol.medios.strains(m).data(j).Particles, 2);
17 spec_growth = Mu_estimate(actual_PartI, stepTime); % OD or Particles
18 evol.medios.strains(m).data(j).mu = spec_growth;
19
20 % Production F1
21 actual_R = median(evol.medios.strains(m).data(j).MEFLcell, 2);
22 dpdt = Mu_estimate(actual_R, stepTime);
23
24 out = evol;
25 end
26
27 ESTIMATION OF THE MU FUNCTION:
28
29 function [estimated_mu] = Mu_estimate(Input_time_series, sampling_time)
30 % Estimates the specific growth rate
31 % We obtain the derivative as
32 %  $dy(k)/dt = (-y(k+2) + 8*y(k+1) - 8*y(k-1) + y(k-2))/12/dt$ 
33 % and then apply a Butterworth filter:
34 % [b,a] = butter(2,4/33);
35 % data_ff = filtfilt(b,a,data);
36 % Input_time_series is the sampled data of absorbance or Particles
37 % Sampling is assumed to be in minutes
38 % estimated_mu gives the estimate of the specific growth rate in hours-1
39
40 % Extend the vector on the left and right extremes:
41 x0_value = Input_time_series(1);
42 xf_value = Input_time_series(end);
43 temp_data = [x0_value; x0_value; Input_time_series; xf_value; xf_value];
44 der_temp_data = (-temp_data(5:end, :) + 8 * temp_data(4:end-1, :) - 8 * temp_data(2:end-3, :) +
45     temp_data(1:end-4, :)) / (12 * sampling_time);
46
47 % Divide by the Input_time_series plus a small offset (1e-6) to avoid dividing by zero
48 raw_estimated_mu = der_temp_data ./ (Input_time_series + 1e-6);
49
50 % Replace divisions by zero with NaN
51 raw_estimated_mu(Input_time_series == 0) = NaN;
52 b = [0.0928, 0.1857, 0.0928];
53 a = [1, -0.9728, 0.3441];
54 mu = filtfilt(b, a, raw_estimated_mu);
55
56 if mu <= 0
57     estimated_mu = 1e-4;
58 else
59     estimated_mu = mu;
60 end
61 end
62 end

```

Experimental data analysis and Hill function plotting

The particular code presented was the one utilized for Experiment 9.

```

1  clc; close all; clearvars;
2  load Experiment9_R4R3gfp_R4R4gfp_NAR_27022024.mat
3  startpoint = 1;
4  strain = char(evol.medios.strains.Id);
5
6  % Generating data
7  for m = 1:size(strain, 1)
8      evol = Datagenerating(evol, m, startpoint);
9  end
10
11 % Plotting
12 culture_media = char(evol.medios.Id);
13 for cm = 1:size(culture_media, 1)
14     strain = char(evol.medios(cm).strains.Id);
15     stepTime = evol.time_OD(2) - evol.time_OD(1); % Interval in minutes
16     timePRE = evol.npre * stepTime / 60 - startpoint * stepTime / 60;
17     timeEXP = (evol.time_OD(startpoint:end) - evol.time_OD(startpoint)) / 60; % Convert to hours
18
19     for m = 1:size(strain, 1)
20         figure('Name', [strain(m, :) ' in ' culture_media(cm, :)]);
21         INDUC = char(evol.medios(cm).strains(m).data.Inducc);
22         total_ax = 4;
23
24         for j = 1:size(INDUC, 1)
25             for n = 1:total_ax
26                 hs = subplot(total_ax, 1, n);
27                 if n == 1
28                     plot(timeEXP, evol.medios(cm).strains(m).data(j).ODblank, 'Color', map(j, :),
29                         'LineWidth', 2)
30                     hold on;
31                     if j == size(INDUC, 1)
32                         legend(INDUC)
33                         xlabel('Time (h)');
34                         ylabel('OD_{600}');
35                         title(strain(m, :));
36                         xlim([0 20])
37                         grid on;
38                         pos = get(hs, 'position');
39                         annotation("line", (pos(1) + timePRE / timeEXP(end) * (pos(3))) * [1 1], ...
40                             [pos(2) pos(2) + pos(4)], 'LineWidth', 1, 'LineStyle', "--");
41                     end
42                 elseif n == 2
43                     plot(timeEXP, evol.medios(cm).strains(m).data(j).MEFL, 'Color', map(j, :),
44                         'LineWidth', 2)
45                     hold on;
46                     if j == size(INDUC, 1)
47                         legend(INDUC)
48                         xlabel('Time (h)');
49                         ylabel('MEFL');
50                         xlim([0 20])
51                         grid on;
52                         pos = get(hs, 'position');
53                         annotation("line", (pos(1) + timePRE / timeEXP(end) * (pos(3))) * [1 1], ...
54                             [pos(2) pos(2) + pos(4)], 'LineWidth', 1, 'LineStyle', "--");
55                     end

```

```

1         elseif n == 3
2             plot(timeEXP, evol.medios(cm).strains(m).data(j).MEFLcell, 'Color', map(j, :),
3                 'LineWidth', 2)
4             hold on;
5             if j == size(INDUC, 1)
6                 legend(INDUC)
7                 xlabel('Time (h)');
8                 ylabel('FOD (MEFL/Particles)');
9                 xlim([0 20])
10                grid on;
11                pos = get(hs, 'position');
12                annotation("line", (pos(1) + timePRE / timeEXP(end) * (pos(3))) * [1 1], ...
13                    [pos(2) pos(2) + pos(4)], 'LineWidth', 1, 'LineStyle', "--");
14            end
15        elseif n == 4
16            plot(timeEXP, evol.medios(cm).strains(m).data(j).mu, 'Color', map(j, :),
17                'LineWidth', 1);
18            hold on;
19            if j == size(INDUC, 1)
20                legend(INDUC)
21                xlabel('Time (h)');
22                ylabel('Growth rate (1/min)');
23                xlim([0 20])
24                title(strain(m, :));
25                grid on;
26                pos = get(hs, 'position');
27                annotation("line", (pos(1) + timePRE / timeEXP(end) * (pos(3))) * [1 1], ...
28                    [pos(2) pos(2) + pos(4)], 'LineWidth', 1, 'LineStyle', "--");
29            end
30        elseif n == 5
31            semilogy(timeEXP, (log(2) ./ evol.medios(cm).strains(m).data(j).mu), 'Color',
32                map(j, :), 'LineWidth', 1);
33            hold on;
34            if j == size(INDUC, 1)
35                legend(INDUC)
36                xlabel('Time (h)');
37                ylabel('Doubling time (min)');
38                xlim([0 20])
39                grid on;
40                pos = get(hs, 'position');
41                annotation("line", (pos(1) + timePRE / timeEXP(end) * (pos(3))) * [1 1], ...
42                    [pos(2) pos(2) + pos(4)], 'LineWidth', 1, 'LineStyle', "--");
43            end
44        end
45    end
46
47    % Hill function
48    t1 = 6; t2 = 8; % Times for Hill Function
49    setime = round(t1 * 60 / 5);
50    endtime = round(t2 * 60 / 5);
51    strain_GFP = strain(1:3, :);
52    HillFunction_plotter(evol, map, strain_GFP, setime, endtime);
53
54    %Normalized Hill function
55    NormalizedHillFunc(evol, map, strain_GFP(:, :), setime, endtime);

```

Hill function calculation

```

1 function expression = HillFunction_plotter(evol, map, strain, setime, endtime)
2
3 Marker_Counter = 1;
4 Markers = {'o','v','s','>','d','^','p'};
5 hf = figure('Name', 'Hill function');
6 ha = axes;
7 hp = [];
8
9 Vcell = 2e-15; % Liters
10 nA = 6.02214076e23; % Avogadro's number
11 molec_to_uM = 1e6 / (Vcell * nA); % uM Conversion factor
12
13 for m = 1:size(strain, 1)
14     INDUC = char(evol.medios.strains(m).data.Inducc);
15
16     if size(INDUC, 1) > 1
17         SizeAHL = size(INDUC, 1);
18         expression = zeros(1, size(INDUC, 1));
19         R_std = zeros(1, size(INDUC, 1));
20
21         inductions = str2num(INDUC(:, :));
22         inductions(1) = 0.0001; % Inductor = 0 nM
23
24         for j = 1:SizeAHL
25             % FOD
26             FOD = evol.medios.strains(m).data(j).MEFLcell(setime:endtime, :); % to start in j = 1 at
                AHL = 0 nM
27             R = FOD;
28
29             expression(1, j) = mean(mean(R, 2), 1);
30             R_var = mean(var(R, 0, 1)) + var(mean(R, 1)); % Law of total variance
31             R_std(1, j) = sqrt(R_var);
32             hp1 = errorbar(ha, inductions(j), expression(1, j), R_std(1, j), strcat('--',
                Markers{Marker_Counter}), ...
33                 'MarkerSize', 8, 'MarkerFaceColor', map(j, :), 'Color', map(j, :), 'MarkerEdgeColor',
                'k', 'LineWidth', 1);
34             hold on;
35         end
36         hp = [hp, hp1];
37         plot(inductions, expression, '--k');
38     end
39     Marker_Counter = Marker_Counter + 1;
40 end
41 set(gca, "XLim", [-1 inductions(end)]);
42 colormap(hf, map(1:SizeAHL, :));
43 xlabel('Inductor (nM)'); ylabel('GFP MEFL/Particles'); % ylabel({...,'MEFL_{cell}'});
44 title('Hill function')
45 lgd = legend(hp, strain(:, :), 'Location', 'northeastoutside');
46 title(lgd, 'Strain');
47 grid on;
48 end

```

Normalized Hill function

```

1 function varargout = NormalizedHillFunc(evol, map, strain, setime, endtime)
2
3 Vcell = 2e-15; % Liters
4 nA = 6.02214076e23; % Avogadro's number
5 molec_to_uM = 1 / (Vcell * nA) * 1e6; % uM Conversion factor
6 molec_to_nM = 1 / (Vcell * nA) * 1e9;
7 map1 = map(:, 1:3);
8
9 total_ax = 2;
10
11 for m = 1:size(strain, 1)
12     INDUC = char(evol.medios.strains(m).data.Inducc);
13     figure('Name', strain(m, :));
14     if size(INDUC, 1) > 1
15         expression = zeros(1, size(INDUC, 1));
16         R_std = zeros(1, size(INDUC, 1));
17
18         inductions = str2num(INDUC(:, :));
19         inductions(1) = 0.0001; % Inductor = 0 nM
20
21         for j = 1:size(INDUC, 1)
22             R = evol.medios.strains(m).data(j).MEFLcell(setime:endtime, :);
23             expression(1, j) = median(mean(R, 2), 1);
24             R_var = mean(var(R, 0, 1)) + var(mean(R, 1)); % Law of total variance
25             R_std(1, j) = sqrt(R_var);
26         end
27
28         % Normalization
29         Minvalue = min(expression);
30         Maxvalue = max(expression);
31         Nexpression = expression ./ Maxvalue;
32
33         for k = 1:length(Nexpression)
34             errorbar(inductions(k), Nexpression(1, k), 0 * R_std(1, k), 'd', 'MarkerSize', 6,
35                 'MarkerFaceColor', map(k, :), ...
36                 'Color', map(k, :), 'MarkerEdgeColor', 'k', 'LineWidth', 1);
37             er.Color = [0 0 0];
38             er.LineStyle = 'none';
39             hold on;
40         end
41         plot(inductions, Nexpression, '--k');
42         xlabel('Inductor-TF (uM)'); ylabel('Promoter activity (normalized)');
43         title(['GFP: ', strain(m, :)]);
44         grid on;
45     end
46 end
47
48 end

```

Organizing by constructions and calculating its mean OD_{600} , MEFL, MEFLcell and variance

```
1 clear
2 cd 'Getting all blanks' % File with Datafiles of the experiments
3 files = dir
4 names = {files.name};
5 names_exp = names(14:end);
6
7 Order experiments according their genetic constructions
8 conversion_list = [3 4 5 6 7 8 9 10 12];
9 for i = 3:length(names_exp)
10     j = conversion_list(i);
11     load(names_exp{i}); % Load experiment files
12     for k = 1:2 %n = strains
13         func_name = sprintf('Datagenerating_%d', j);
14         evol = feval(func_name, evol, k, 1);
15         exp_name = sprintf('exp_%d', j);
16         if k == 1
17             s1.(exp_name) = evol.medios.strains(1)
18         else
19             s2.(exp_name) = evol.medios.strains(2)
20         end
21     end
22 end
23
24 %Rename
25 R4R4_16_E6 = s1.exp_6;
26 R3R3_2_E7 = s1.exp_7;
27 R4R3_7_E7 = s2.exp_7;
28 R3R3_2_E8 = s1.exp_8;
29 R3R4_5_E8 = s2.exp_8;
30 R4R3_7_E9 = s1.exp_9;
31 R4R4_16_E9 = s2.exp_9;
32 R3R4_5_E10 = s1.exp_10;
33 R4R4_16_E10 = s2.exp_10;
34 R3R4_2_E12 = s1.exp_12;
35 R4R4_4_E12 = s2.exp_12;
36
37 R3R3 = [R3R3_2_E7, R3R3_2_E8];
38 R3R4 = [R3R4_5_E10, R3R4_2_E12];
39 R4R3 = [R4R3_7_E7, R4R3_7_E9];
40 R4R4 = [R4R4_16_E6, R4R4_16_E9, R4R4_16_E10, R4R4_4_E12];
41
42 construc{1,1} = R3R3;
43 construc{1,2} = R3R4;
44 construc{2,1} = R4R3;
45 construc{2,2} = R4R4;
```

```

1 OBTAIN METRICS AND STORE IT IN ARRAYS AND TABLES.
2
3 % R structure contains 3 sub-structures, each repeated 4 times, corresponding to each construction.
   Each sub-structure includes:
4 % 1. The overall mean for that construction.
5 % 2. The mean for each experiment within that construction.
6 % 3. The mean in overall time of each construction.
7 % Within these sub-structures, data is organized into tables containing four main fields: OD,
   Fluorescence, MEFL and MEFLcell.
8
9 % Cells R_OD, R_F, R_MEFL, R_Meflc and R_Mu contain 4 arrays, one for each construction. In each
   array, there are 242 rows which represent the time, varying number of columns corresponding
   to the experiments performed for the construction (since not all constructions have the same
   number of experiments) and 11 fields for the 11 inductions performed. According to which R_
   cell is accessed, OD, fluorescence or MEFLcell values will be provided.
10
11 R = {};
12 % V= {};
13 conv_list = [3, 4];
14 R_OD = {};
15 R_F = {};
16 R_Meflc = {};
17 R_Mu = {};
18 R_Mefl = {};
19
20 R_od = zeros(242, 11);
21 R_f = zeros(242, 11);
22 R_m_c = zeros(242, 11);
23 R_m = zeros(242, 11);
24 T = table();
25
26 for i = 1:size(construc, 1) % Access rows of construc
27     for j = 1:size(construc, 2) % Access columns of construc
28         % Arrays to store OD, F and MEFL values over time, exp and inductions
29         OD_b = zeros(242, length(construc{i, j}), 11);
30         F_b = zeros(242, length(construc{i, j}), 11);
31         MEFLc = zeros(242, length(construc{i, j}), 11);
32         Mefl = zeros(242, length(construc{i, j}), 11);
33         mu = zeros(242, length(construc{i, j}), 11);
34         % Columns of each construction varies (different N of exp included)
35         % R3R3: 2 exp, R3R4: 3 exp, R4R3: 2 exp, R4R4: 4 exp
36         % Arrays to store means of the experiments for each induction and time
37         m_OD = zeros(242, 11);
38         m_F = zeros(242, 11);
39         m_MEFL = zeros(242, 11);
40         m_MEFLcell = zeros(242, 11);
41         % Arrays to store means of time for each induction
42         mit_OD = zeros(11, 1);
43         mit_F = zeros(11, 1);
44         mit_MEFL = zeros(11, 1);
45         mit_MEFLc = zeros(11, 1);
46
47         for k = 1:length(construc{i, j}) % Exps done for each genetic construct
48             for l = 1:length(construc{i, j}(k).data) % Inductions
49                 OD_b(:, k, l) = mean(construc{i, j}(k).data(1).ODblank(1:242, :), 2);
50                 F_b(:, k, l) = mean(construc{i, j}(k).data(1).Fblank(1:242, :), 2);
51                 MEFL(:, k, l) = mean(real(construc{i, j}(k).data(1).MEFLcell(1:242, :)), 2);
52                 MEFLc(:, k, l) = mean(real(construc{i, j}(k).data(1).MEFLcell(1:242, :)), 2);
53                 mu(:, k, l) = mean(real(construc{i, j}(k).data(1).mu(1:242, :)), 2);
54             end
55         end
56
57         R_OD{i, j} = OD_b; % Cell stores values
58         R_F{i, j} = F_b;
59         R_MEFL{i, j} = MEFL;
60         R_Meflc{i, j} = MEFLc;
61         R_Mu{i, j} = mu;

```

```

1     f = conv_list(i);
2     p = conv_list(j);
3     res = sprintf('Res_exp_%d_%d', f, p);
4
5     % Create tables and store in the structure
6     for h = 1:length(construc{i, j})
7         R_od = OD_b(:, h, :);
8         R_f = F_b(:, h, :);
9         R_m = MEFL(:, h, :);
10        R_m_c = MEFLc(:, h, :);
11
12        T = table();
13        T.OD = reshape(R_od, 242, 11);
14        T.Fluorescence = reshape(R_f, 242, 11);
15        T.MEFL = reshape(R_m, 242, 11);
16        T.MEFLcell = reshape(R_m_c, 242, 11);
17        T.Properties.VariableNames = {'OD', 'Fluorescence', 'MEFL', 'MEFLcell'};
18
19        R.(res){h, 1} = T;
20        % Table for the mean value of experiments for each induction
21        m_OD(:, :) = mean(R_OD{i, j}, 2);
22        m_F(:, :) = mean(R_F{i, j}, 2);
23        m_MEFL(:, :) = mean(R_MEFL{i, j}, 2);
24        m_MEFLcell(:, :) = mean(R_Meflcell{i, j}, 2);
25        res = sprintf('Res_%d_%d', f, p);
26        R.(res) = table(m_OD, m_F, m_MEFL, m_MEFLcell, 'VariableNames', {'OD',
27            'Fluorescence', 'MEFL', 'MEFLcell'});
28
29        % Table for the mean value of each induction over time
30        m1t_OD(:, 1) = mean(m_OD, 1);
31        m1t_F(:, 1) = mean(m_F, 1);
32        m1t_MEFL(:, 1) = mean(m_MEFL, 1);
33        m1t_MEFLc(:, 1) = mean(m_MEFLcell, 1);
34        res = sprintf('Res_1time_%d_%d', f, p);
35        R.(res) = table(m1t_OD, m1t_F, m1t_MEFL, m1t_MEFLc, 'VariableNames', {'OD',
36            'Fluorescence', 'MEFL', 'MEFLcell'});
37    end
38 end
39 cd ..

```

Calculate performance metrics in specific time interval and its variances

```
1 % OBTAIN DATA IN TIME INTERVAL 150-175 AND ITS VARIANCES.
2 % To analyze the model, we selected the time interval from 150 to 175 based on the plots of the
   genetic constructions over time.
3
4 % Data Storage:
5 % Measurements is a cell array that stores 3 arrays: one for OD data, one for fluorescence data,
   and one for MEFLcell data.
6 % Each array has dimensions (242, 11, 4) = (time, inductions, constructions).
7
8 % Variance Calculation:
9 % Arrays v_optd, v_fce, and v_mce store variance values over time interval 150-175 for each
   induction in each construction. Their size is (11, 4) = (inductions, constructions).
10 % Cells V_exp_OD, V_exp_F, and V_exp_M store variances among experiments for each induction of each
   construction in the time interval 150-175. Each cell contains four arrays (one for each
   construction). These arrays store OD, fluorescence, or MEFLcell variance values according to
   their suffix.
11 % Finally, the law of total variance is applied. The structure V_Total contains 3 cells: VT_OD,
   VT_F, and VT_M. Each cell stores 4 arrays (one for each construction) with dimensions (11, 1)
   = (inductions, total variance value). The term total variance value refers to the sum of the
   variance among the chosen time interval (var(mean)) and the variance among experiments of a
   specific construction (mean(var)).
12
13 m_OD = zeros(242, 11, 4);
14 Fluor = zeros(242, 11, 4);
15 Mcell = zeros(242, 11, 4);
16
17 fields = {'Res_3_3', 'Res_3_4', 'Res_4_3', 'Res_4_4'};
18
19 for i = 1:length(fields)
20     field_name = fields{i};
21     m_OD(:, :, i) = R.(field_name).OD;
22     Fluor(:, :, i) = R.(field_name).Fluorescence;
23     Mcell(:, :, i) = R.(field_name).MEFLcell;
24 end
25
26 % Store measurements in a cell array
27 Measurements = {m_OD, Fluor, Mcell};
28
29 optd = zeros(11, 4); % Rows: inductions, columns: genetic constructions
30 fce = zeros(11, 4);
31 mce = zeros(11, 4);
32
33 v_optd = zeros(11, 4); % Rows: inductions, columns: genetic constructions
34 v_fce = zeros(11, 4);
35 v_mce = zeros(11, 4);
```

```

1 % Calculate means and variances for each construction and induction
2 for i = 1:size(Measurements, 1) % Rows
3     for j = 1:size(Measurements, 2) % Columns
4         type = Measurements{i, j};
5         for k = 1:size(type, 3) % Constructions
6             for l = 1:size(type, 2) % Inductions
7                 if j == 1
8                     optd(l, k) = mean(type(150:175, l, k));
9                     v_optd(l, k) = var(type(150:175, l, k)); % Variance along 150:175
10                elseif j == 2
11                    fce(l, k) = mean(type(150:175, l, k));
12                    v_fce(l, k) = var(type(150:175, l, k));
13                elseif j == 3
14                    mce(l, k) = mean(type(150:175, l, k));
15                    v_mce(l, k) = var(type(150:175, l, k));
16                end
17            end
18        end
19    end
20 end
21
22 % Variance of experiments in the construction (time 150:175). OD, F, and MEFL
23
24 V_exp_OD = cell(size(R_OD));
25 V_exp_F = cell(size(R_F));
26 V_exp_M = cell(size(R_Meflc));
27
28
29 fields = {R_OD, R_F, R_Meflc};
30 var_cells = {V_exp_OD, V_exp_F, V_exp_M};
31
32 % Calculate experimental variances
33 for h = 1:length(fields)
34     field_name = fields{h};
35     var_name = var_cells{h};
36     for i = 1:size(field_name, 1) % Rows
37         for j = 1:size(field_name, 2) % Columns
38             for k = 1:size(field_name{i, j}, 3) % Inductions
39                 var_data = var(field_name{i, j}(150:175, :, k), 0, 2);
40                 mean_var = mean(var_data);
41                 var_name{i, j}(k, 1) = mean_var;
42             end
43         end
44     end
45
46     % Assign calculated variances to respective cells
47     if h == 1
48         V_exp_OD = var_name;
49     elseif h == 2
50         V_exp_F = var_name;
51     elseif h == 3
52         V_exp_M = var_name;
53     end
54 end
55
56 % Law of total variance
57
58 V_Total = struct();
59 V_Total.VT_OD = cell(1, 1);
60 V_Total.VT_F = cell(1, 1);
61 V_Total.VT_M = cell(1, 1);

```

```

1 % Calculate total variance
2 l = 1;
3 for i = 1:size(V_exp_OD, 1) % Rows
4     for j = 1:size(V_exp_OD, 2) % Columns
5         for k = 1:size(V_exp_OD{i, j}, 1) % Inductions
6             Vt_OD(k,:) = sqrt(V_exp_OD{i, j}(k) + v_optd(k, l));
7             Vt_F(k,:) = sqrt(V_exp_F{i, j}(k) + v_fce(k, l));
8             Vt_MEFL_R3R3(k,:) = sqrt(V_exp_M{i, j}(k) + v_mce(k, l));
9         end
10        V_Total.VT_OD{i, j} = Vt_OD;
11        V_Total.VT_F{i, j} = Vt_F;
12        V_Total.VT_M{i, j} = Vt_MEFL_R3R3;
13        l = l + 1;
14    end
15 end
16
17 % SAVE DATA IN FILES
18
19 % Data for R3R3
20 GFP_expression_var(1,:) = V_Total.VT_M{1,1}; % Variance of MEFLcell
21 for i = 1:size(R.Res_3_3.MEFLcell, 2) % MEFLcell expression in t = 150-175
22     GFP_expression(:,i) = mean(R.Res_3_3.MEFLcell(150:175, i));
23 end
24 save('Data_R3R3.mat', 'GFP_expression', 'GFP_expression_var');
25
26 % Data for R4R4
27 GFP_expression_var(1,:) = V_Total.VT_M{2,2}; % Variance of MEFLcell
28 for i = 1:size(R.Res_4_4.MEFLcell, 2) % MEFLcell expression in t = 150-175
29     GFP_expression(:,i) = mean(R.Res_4_4.MEFLcell(150:175, i));
30 end
31 save('Data_R4R4.mat', 'GFP_expression', 'GFP_expression_var');
32
33 % Data for R3R4
34 GFP_expression_var(1,:) = V_Total.VT_M{1,2}; % Variance of MEFLcell
35 for i = 1:size(R.Res_3_4.MEFLcell, 2) % MEFLcell expression in t = 150-175
36     GFP_expression(:,i) = mean(R.Res_3_4.MEFLcell(150:175, i));
37 end
38 save('Data_R3R4.mat', 'GFP_expression', 'GFP_expression_var');
39
40 % Data for R4R3
41 GFP_expression_var(1,:) = V_Total.VT_M{2,1}; % Variance of MEFLcell
42 for i = 1:size(R.Res_4_3.MEFLcell, 2) % MEFLcell expression in t = 150-175
43     GFP_expression(:,i) = mean(R.Res_4_3.MEFLcell(150:175, i));
44 end
45 save('Data_R4R3.mat', 'GFP_expression', 'GFP_expression_var');
46
47 % GROWTH RATE
48
49 Mean_mu = struct();
50 Mean_mu.R3R3 = zeros(11, 1);
51 Mean_mu.R3R4 = zeros(11, 1);
52 Mean_mu.R4R3 = zeros(11, 1);
53 Mean_mu.R4R4 = zeros(11, 1);
54
55 V_mu = struct();
56 V_mu.exp = zeros(11, 4);
57 V_mu.time = zeros(11, 4);
58
59 % Calculate mean growth rate
60 Mean_mu.R3R3(:, 1) = squeeze(mean(mean(R_Mu{1,1}(150:175, :, :), 1), 2));
61 Mean_mu.R3R4(:, 1) = squeeze(mean(mean(R_Mu{1,2}(150:175, :, :), 1), 2));
62 Mean_mu.R4R3(:, 1) = squeeze(mean(mean(R_Mu{2,1}(150:175, :, :), 1), 2));
63 Mean_mu.R4R4(:, 1) = squeeze(mean(mean(R_Mu{2,2}(150:175, :, :), 1), 2));

```

```

1 % Calculate variance
2 l = 1;
3 for i = 1:size(R_Mu, 1) % Rows
4     for j = 1:size(R_Mu, 2) % Columns
5         for k = 1:size(R_Mu{i,j}, 3) % Inductions
6             V_mu.exp(k, l) = mean(var(R_Mu{i,j}(150:175, :, k), 0, 2));
7             V_mu.time(k, l) = var(mean(R_Mu{i,j}(150:175, :, k), 2), 0, 1);
8             V_total_mu(k, l) = sqrt(V_mu.exp(k, l) + V_mu.time(k, l));
9         end
10        l = l + 1;
11    end
12 end
13
14 % Plot mean growth rate
15 figure;
16 plot(induc, Mean_mu.R3R3, '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
17 hold on;
18 plot(induc, Mean_mu.R3R4, '-s', 'LineWidth', 1.5, 'MarkerSize', 3);
19 plot(induc, Mean_mu.R4R3, '-^', 'LineWidth', 1.5, 'MarkerSize', 3);
20 plot(induc, Mean_mu.R4R4, '-d', 'LineWidth', 1.5, 'MarkerSize', 3);
21
22 title('Growth rate in constructions, t = 150:175');
23 xlabel('Induction');
24 ylabel('Mu function');
25 legend({'R3R3', 'R3R4', 'R4R3', 'R4R4'}, 'Location', 'northeastoutside');
26
27 grid on;
28 hold off;
29
30 % Plot variance of growth rate
31 figure;
32 plot(induc, V_total_mu(:, 1), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
33 hold on;
34 plot(induc, V_total_mu(:, 2), '-s', 'LineWidth', 1.5, 'MarkerSize', 3);
35 plot(induc, V_total_mu(:, 3), '-^', 'LineWidth', 1.5, 'MarkerSize', 3);
36 plot(induc, V_total_mu(:, 4), '-d', 'LineWidth', 1.5, 'MarkerSize', 3);
37
38 title('Variance of Growth rate in constructions, t = 150:175');
39 xlabel('Induction');
40 ylabel('Variance of mu function');
41 legend({'R3R3', 'R3R4', 'R4R3', 'R4R4'}, 'Location', 'northeastoutside');
42
43 grid on;
44 hold off;

```

Plot constructions

```
1 load("Results.mat")
2
3 time = 0.0543:0.083:20.0573;
4
5 INDUCC=char(evol.medios.strains(1).data.Inducc);
6 induc=str2num(INDUCC(:,:));
7 induc(1)=0.0001;
8
9 PLOT MEAN MEFLcell FOR EACH CONSTRUCTION OVER TIME INTERVAL 150-175
10
11 figure;
12
13 % First subplot
14 subplot(2, 2, 1);
15 plot(induc, R.Res_3_3.MEFLcell(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
16 title('R3R3');
17 xlabel('Induction');
18 ylabel('MEFL_{cell} value');
19 grid on;
20 hold on;
21
22 colormap(parula(length(legend_labels)));
23 colors = colormap;
24 line_objs = findobj(gca, 'Type', 'Line');
25 for i = 1:length(line_objs)
26     set(line_objs(i), 'Color', colors(i,:));
27 end
28
29 % Second subplot
30 subplot(2, 2, 2);
31 semilogy(induc, R.Res_3_4.MEFLcell(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
32 title('R3R4');
33 xlabel('Induction');
34 ylabel('MEFL_{cell} value');
35 grid on;
36 hold on;
37
38 colormap(parula(length(legend_labels)));
39 colors = colormap;
40 line_objs = findobj(gca, 'Type', 'Line');
41 for i = 1:length(line_objs)
42     set(line_objs(i), 'Color', colors(i,:));
43 end
44
45 % Third subplot
46 subplot(2, 2, 3);
47 plot(induc, R.Res_4_3.MEFLcell(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
48 title('R4R3');
49 xlabel('Induction');
50 ylabel('MEFL_{cell} value');
51 grid on;
52 hold on;
53
54 colormap(parula(length(legend_labels)));
55 colors = colormap;
56 line_objs = findobj(gca, 'Type', 'Line');
57 for i = 1:length(line_objs)
58     set(line_objs(i), 'Color', colors(i,:));
59 end
```

```

1 % Fourth subplot
2 subplot(2, 2, 4);
3 plot(induc, R.Res_4_4.MEFLcell(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
4 title('R4R4');
5 xlabel('Induction');
6 ylabel('MEFL_{cell} value');
7 grid on;
8 hold on;
9
10 colormap(parula(length(legend_labels)));
11 colors = colormap;
12 line_objs = findobj(gca, 'Type', 'Line');
13 for i = 1:length(line_objs)
14     set(line_objs(i), 'Color', colors(i,:));
15 end
16
17 subplot(2, 2, 1);
18 set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);
19
20 subplot(2, 2, 2);
21 set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);
22
23 subplot(2, 2, 3);
24 set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);
25
26 subplot(2, 2, 4);
27 set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);
28
29 PLOT MEAN MEFL FOR EACH CONSTRUCTION OVER TIME INTERVAL 150-175
30
31 figure;
32
33 % First subplot
34 subplot(2, 2, 1);
35 plot(induc, R.Res_3_3.MEFL(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
36 title('R3R3');
37 xlabel('Induction');
38 ylabel('MEFL_{cell} value');
39 grid on;
40 hold on;
41
42 colormap(parula(length(legend_labels)));
43 colors = colormap;
44 line_objs = findobj(gca, 'Type', 'Line');
45 for i = 1:length(line_objs)
46     set(line_objs(i), 'Color', colors(i,:));
47 end
48
49 % Second subplot
50 subplot(2, 2, 2);
51 semilogy(induc, R.Res_3_4.MEFL(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
52 title('R3R4');
53 xlabel('Induction');
54 ylabel('MEFL value');
55 grid on;
56 hold on;
57
58 colormap(parula(length(legend_labels)));
59 colors = colormap;
60 line_objs = findobj(gca, 'Type', 'Line');
61 for i = 1:length(line_objs)
62     set(line_objs(i), 'Color', colors(i,:));
63 end

```

```

1 % Third subplot
2 subplot(2, 2, 3);
3 plot(induc, R.Res_4_3.MEFL(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
4 title('R4R3');
5 xlabel('Induction');
6 ylabel('MEFL value');
7 grid on;
8 hold on;
9
10 colormap(parula(length(legend_labels)));
11 colors = colormap;
12 line_objs = findobj(gca, 'Type', 'Line');
13 for i = 1:length(line_objs)
14     set(line_objs(i), 'Color', colors(i,:));
15 end
16
17 % Fourth subplot
18 subplot(2, 2, 4);
19 plot(induc, R.Res_4_4.MEFL(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
20 title('R4R4');
21 xlabel('Induction');
22 ylabel('MEFL value');
23 grid on;
24 hold on;
25
26 colormap(parula(length(legend_labels)));
27 colors = colormap;
28 line_objs = findobj(gca, 'Type', 'Line');
29 for i = 1:length(line_objs)
30     set(line_objs(i), 'Color', colors(i,:));
31 end
32
33 subplot(2, 2, 1);
34 set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);
35
36 subplot(2, 2, 2);
37 set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);
38
39 subplot(2, 2, 3);
40 set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);
41
42 subplot(2, 2, 4);
43 set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);
44
45 PLOT MEAN OD FOR EACH CONSTRUCTION OVER TIME
46
47 figure;
48
49 % First subplot
50 subplot(2, 2, 1);
51 plot(time, R.Res_3_3.OD, '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
52 title('R3R3');
53 xlabel('Time [h]');
54 ylabel('OD_{600} value');
55 grid on;
56 hold on;
57
58 colormap(parula(length(legend_labels)));
59 colors = colormap;
60 line_objs = findobj(gca, 'Type', 'Line');
61 for i = 1:length(line_objs)
62     set(line_objs(i), 'Color', colors(i,:));
63 end

```

```

1 % Second subplot
2 subplot(2, 2, 2);
3 plot(time, R.Res_3_4.OD, '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
4 title('R3R4');
5 xlabel('Time [h]');
6 ylabel('OD_{600} value');
7 grid on;
8 hold on;
9
10 colormap(parula(length(legend_labels)));
11 colors = colormap;
12 line_objs = findobj(gca, 'Type', 'Line');
13 for i = 1:length(line_objs)
14     set(line_objs(i), 'Color', colors(i,:));
15 end
16
17 % Third subplot
18 subplot(2, 2, 3);
19 plot(time, R.Res_4_3.OD, '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
20 title('R4R3');
21 xlabel('Time [h]');
22 ylabel('OD_{600} value');
23 grid on;
24 hold on;
25
26 colormap(parula(length(legend_labels)));
27 colors = colormap;
28 line_objs = findobj(gca, 'Type', 'Line');
29 for i = 1:length(line_objs)
30     set(line_objs(i), 'Color', colors(i,:));
31 end
32
33 % Fourth subplot
34 subplot(2, 2, 4);
35 plot(time, R.Res_4_4.OD, '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
36 title('R4R4');
37 xlabel('Time [h]');
38 ylabel('OD_{600} value');
39 grid on;
40 hold on;
41
42 colormap(parula(length(legend_labels)));
43 colors = colormap;
44 line_objs = findobj(gca, 'Type', 'Line');
45 for i = 1:length(line_objs)
46     set(line_objs(i), 'Color', colors(i,:));
47 end
48
49 subplot(2, 2, 1);
50 set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);
51
52 subplot(2, 2, 2);
53 set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);
54
55 subplot(2, 2, 3);
56 set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);
57
58 subplot(2, 2, 4);
59 set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);

```

Simulation

```

1 clear
2
3 % Load data from R3R3, R4R3, and R4R4
4 D = struct();
5 D.GFP_exp_var = zeros(3, 11);
6 D.GFP_exp = zeros(3, 11);
7
8 l = 1;
9 for j = 3:4
10     for i = 3:4
11         data_name = sprintf('Data_R%dR%d.mat', j, i);
12         load(data_name)
13         if strcmp(data_name, 'Data_R3R4.mat')
14             continue
15         else
16             D.GFP_exp_var(1,:) = GFP_expression_var;
17             D.GFP_exp(1,:) = GFP_expression;
18             Data_exp{j-2,i-2}.m = GFP_expression;
19             Data_exp{j-2,i-2}.v = GFP_expression_var;
20         end
21         l = l + 1;
22     end
23 end
24
25 % Fixed model parameters
26 p.kP1 = 0.56; % Transcription rate constant
27 p.dm = 0.399; % mRNA degradation rate
28 p.dP = 0.0002; % Protein degradation rate
29 p.CN = 40; % Copy number of the plasmid
30 p.mu = log(2) / 30; % Growth rate constant
31
32 % Induction
33 p.NAR_induction = [0, 30, 75, 100, 125, 145, 175, 250, 400, 700, 1400];
34
35 p.Data = Data_exp;
36
37 % Bounds for parameters to estimate
38 lower_bounds = [2, 0.1, 0.1, 0.1, 1, 1, 1, 0.005, 0.001, 0.009, 8, 0.9, 600];
39 upper_bounds = [2, 5, 30, 5, 1, 1, 1, 0.02, 0.02, 0.2, 9, 1.4, 730];
40
41 % Optimization
42 options = optimoptions('ga', 'Display', 'iter', 'MaxGenerations', 100000);
43 [best_param, Jmin_value] = ga(@(param) Jprom(param, p), 13, [], [], [], [], lower_bounds,
44     upper_bounds, [], options);
45
46 % Assign optimized values to model parameters
47 p.nFde0 = best_param(1); % Hill coefficient
48 kFde0_v = best_param(2:4);
49 kdNar_v = best_param(5:7);
50 beta_v = best_param(8:10);
51 RBS = best_param(11:12);
52 p.kFde = best_param(13);

```

```

1 % Simulate and plot results
2 nar = 0:1400;
3 figure;
4 k = 0;
5 names = {'R3R3', 'R4R3', 'R4R4'};
6 for RBS_TFi = 1:2
7     p.RBS_TF = RBS(RBS_TFi);
8     for RBS_BSi = 1:2
9         p.RBS_BS = RBS(RBS_BSi);
10        if (RBS_TFi == 1 && RBS_BSi == 2)
11            continue;
12        end
13        k = k + 1;
14        p.kFde0 = kFde0_v(k);
15        p.kdNar = kdNar_v(1);
16        p.beta = beta_v(k);
17        GFP = evaluate_pFde0(nar, p);
18        GFP_exp = p.Data{RBS_TFi, RBS_BSi}.m;
19        GFP_var = p.Data{RBS_TFi, RBS_BSi}.v;
20
21        subplot(3, 1, k);
22        semilogy(nar, GFP, 'LineWidth', 1);
23        hold on;
24        errorbar(p.NAR_induction, GFP_exp, GFP_var, '-d', 'LineWidth', 1, 'MarkerSize', 1.5);
25        hold off;
26
27        title(['Expression of GFP ' names{k}]);
28        xlabel('Induction');
29        ylabel('GFP');
30        grid on;
31    end
32 end
33
34 % Jprom function: calculates mean squared error between experimental and model GFP
35 function RMLSE = Jprom(param, p)
36     % Model parameters = current optimization values
37     p.nFde0 = param(1);
38     kFde0_v = param(2:4);
39     kdNar_v = param(5:7);
40     beta_v = param(8:10);
41     RBS = param(11:12);
42     p.kFde = param(13);
43
44     RMLSE_parciales = [];
45     NAR = p.NAR_induction;
46     k = 0;
47     for RBS_TFi = 1:2
48         RBS_TF = RBS(RBS_TFi);
49         for RBS_BSi = 1:2
50             RBS_BS = RBS(RBS_BSi);
51             if (RBS_TFi == 1 && RBS_BSi == 2)
52                 continue;
53             end
54             k = k + 1;
55
56             kFde0 = kFde0_v(k);
57             kdNar = kdNar_v(1);
58             beta = beta_v(k);

```

```

1      % Steady state calculation
2      p.Fder_ss = RBS_TF * p.CN * p.kP1 / ((p.dm + p.mu) * (p.dP + p.mu));
3      Fder = p.Fder_ss;
4
5      % pFde0 calculation
6      pFde0 = beta + (1 - beta) * NAR.^p.nFde0 ./ ...
7          (p.kFde * (kdNar * p.CN / Fder).^p.nFde0 + NAR.^p.nFde0);
8
9      % KEXP calculation
10     KEXP = RBS_BS * p.CN * kFde0 / (p.dm + p.mu);
11
12     % GFP calculation
13     GFP = KEXP .* pFde0 / (p.dP + p.mu);
14
15     % Calculate partial RMLSE
16     GFP_exp = p.Data{RBS_TFi, RBS_BSi}.m;
17     J = sqrt(sum((log10(GFP_exp + 1) - log10(GFP + 1)).^2));
18     if (RBS_TFi == 2 && RBS_BSi == 2)
19         J = J * 0;
20     end
21     if (RBS_TFi == 1 && RBS_BSi == 1)
22         J = J * 1.5;
23     end
24     RMLSE_parciales = [RMLSE_parciales J];
25 end
26 end
27 RMLSE = mean(RMLSE_parciales);
28 end
29
30 % evaluate_pFde0 function: predicts GFP expression given NAR concentration and parameters
31 function [GFP] = evaluate_pFde0(NAR, p)
32     p.Fder_ss = p.RBS_TF * p.CN * p.kP1 / ((p.dm + p.mu) * (p.dP + p.mu));
33     pFde0 = p.beta + (1 - p.beta) * NAR.^p.nFde0 ./ ...
34         (p.kFde * (p.kdNar * p.CN / p.Fder_ss).^p.nFde0 + NAR.^p.nFde0);
35
36     KEXP = p.RBS_BS * p.CN * p.kFde0 / (p.dm + p.mu);
37     GFP = KEXP .* pFde0 / (p.dP + p.mu);
38 end

```