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 μ(t) function.

```
function out = Datagenerating(evol, m, startpoint)

stepTime = evol.time_OD(2) - evol.time_OD(1); % Interval between time points in minutes

INDUC = char(evol.medios.strains(m).data.Inducc);
if size(INDUC, 1) > 1
    inductions = str2num(INDUC(:,:));
    inductions(1) = 0.00001; % Change the first induction, which is 0, to a very small value
end

for j = 1:size(INDUC, 1)

    % Calculate OD blank + NAR for each INDUC
    OD_blank_In = evol.medios.OD(startpoint:end, j);

    % Calculate F blank + NAR for each INDUC
    F_blank_In = evol.medios.F(startpoint:end, j);

    % Calculate calibrated OD and Fluorescence
    ODblank = evol.medios.strains(m).data(j).OD(startpoint:end, :) - OD_blank_In;
    Fblank = evol.medios.strains(m).data(j).F(startpoint:end, :) - F_blank_In;

    % Check the number of replicas for calibration
    if ~isequal(size(ODblank), size(Fblank))
        error = size(ODblank, 2) - size(Fblank, 2);
        if error < 0
            ODblank = [ODblank, ODblank(:, 1) .* ones(length(ODblank), -error)];
        else
            Fblank = [Fblank, Fblank(:, 1) .* ones(length(Fblank), error)];
        end
    end
```

```
    % Save calibrated data
    evol.medios.strains(m).data(j).ODblank = ODblank;
    evol.medios.strains(m).data(j).OD_blank_In = OD_blank_In;
    evol.medios.strains(m).data(j).Fblank = Fblank;
    evol.medios.strains(m).data(j).F_blank_In = F_blank_In;

    [R, Rlog, Particles, MEF] = Calibrate_MEFL_GFPmut3b(ODblank, Fblank);

    evol.medios.strains(m).data(j).Particles = Particles;
    evol.medios.strains(m).data(j).MEFL = MEF;
    evol.medios.strains(m).data(j).MEFLcell = R;
    evol.medios.strains(m).data(j).LogMEFcell = Rlog;

    % Growth rate
    actual_OD = mean(evol.medios.strains(m).data(j).ODblank, 2);
    actual_Parti = mean(evol.medios.strains(m).data(j).Particles, 2);
    spec_growth = Mu_estimate(actual_Parti, stepTime); % OD or Particles
    evol.medios.strains(m).data(j).mu = spec_growth;

    % Production F1
    actual_R = median(evol.medios.strains(m).data(j).MEFLcell, 2);
    dpdt = Mu_estimate(actual_R, stepTime);

    out = evol;
end

ESTIMATION OF THE MU FUNCTION:

function [estimated_mu] = Mu_estimate(Input_time_series, sampling_time)
    % Estimates the specific growth rate
    % We obtain the derivative as
    % dy(k)/dt = (-y(k+2) + 8*y(k+1) - 8*y(k-1) + y(k-2))/12/dt
    % and then apply a Butterworth filter:
    % [b,a] = butter(2,4/33);
    % data_ff = filtfilt(b,a,data);
    % Input_time_series is the sampled data of absorbance or Particles
    % Sampling is assumed to be in minutes
    % estimated_mu gives the estimate of the specific growth rate in hours^{-1}

    % Extend the vector on the left and right extremes:
    x0_value = Input_time_series(1);
    xf_value = Input_time_series(end);
    temp_data = [x0_value; x0_value; Input_time_series; xf_value; xf_value];
    der_temp_data = (-temp_data(5:end, :) + 8 * temp_data(4:end-1, :) - 8 * temp_data(2:end-3, :) +
         temp_data(1:end-4, :)) / (12 * sampling_time);

    % Divide by the Input_time_series plus a small offset (1e-6) to avoid dividing by zero
    raw_estimated_mu = der_temp_data ./ (Input_time_series + 1e-6);

    % Replace divisions by zero with NaN
    raw_estimated_mu(Input_time_series == 0) = NaN;
    b = [0.0928, 0.1857, 0.0928];
    a = [1, -0.9728, 0.3441];
    mu = filtfilt(b, a, raw_estimated_mu);

    if mu <= 0
       estimated_mu = 1e-4;
    else
       estimated_mu = mu;
    end
end

end
```

Experimental data analysis and Hill function plotting

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

```
clc; close all; clearvars;
load Experiment9_R4R3gfp_R4R4gfp_NAR_27022024.mat
startpoint = 1;
strain = char(evol.medios.strains.Id);

% Generating data
for m = 1:size(strain, 1)
    evol = Datagenerating(evol, m, startpoint);
end

% Plotting
culture_media = char(evol.medios.Id);
for cm = 1:size(culture_media, 1)
    strain = char(evol.medios(cm).strains.Id);
    stepTime = evol.time_OD(2) - evol.time_OD(1); % Interval in minutes
    timePRE = evol.npre * stepTime / 60 - startpoint * stepTime / 60;
    timeEXP = (evol.time_OD(startpoint:end) - evol.time_OD(startpoint)) / 60; % Convert to hours

    for m = 1:size(strain, 1)
        figure('Name', [strain(m, :) ' in ' culture_media(cm, :)]);
        INDUC = char(evol.medios(cm).strains(m).data.Inducc);
        total_ax = 4;

        for j = 1:size(INDUC, 1)
            for n = 1:total_ax
                hs = subplot(total_ax, 1, n);
                if n == 1
                    plot(timeEXP, evol.medios(cm).strains(m).data(j).ODblank, 'Color', map(j, :),
                        'LineWidth', 2)
                    hold on;
                    if j == size(INDUC, 1)
                        legend(INDUC)
                        xlabel('Time (h)');
                        ylabel('OD_{600}');
                        title(strain(m, :));
                        xlim([0 20])
                        grid on;
                        pos = get(hs, 'position');
                        annotation("line", (pos(1) + timePRE / timeEXP(end) * (pos(3))) * [1 1], ...
                            [pos(2) pos(2) + pos(4)], 'LineWidth', 1, 'LineStyle', "--");
                    end

                elseif n == 2
                    plot(timeEXP, evol.medios(cm).strains(m).data(j).MEFL, 'Color', map(j, :),
                        'LineWidth', 2)
                    hold on;
                    if j == size(INDUC, 1)
                        legend(INDUC)
                        xlabel('Time (h)');
                        ylabel('MEFL');
                        xlim([0 20])
                        grid on;
                        pos = get(hs, 'position');
                        annotation("line", (pos(1) + timePRE / timeEXP(end) * (pos(3))) * [1 1], ...
                            [pos(2) pos(2) + pos(4)], 'LineWidth', 1, 'LineStyle', "--");
                    end
```

```
                elseif n == 3
                    plot(timeEXP, evol.medios(cm).strains(m).data(j).MEFLcell, 'Color', map(j, :),
                         'LineWidth', 2)
                    hold on;
                    if j == size(INDUC, 1)
                        legend(INDUC)
                        xlabel('Time (h)');
                        ylabel('FOD (MEFL/Particles)');
                        xlim([0 20])
                        grid on;
                        pos = get(hs, 'position');
                        annotation("line", (pos(1) + timePRE / timeEXP(end) * (pos(3))) * [1 1], ...
                            [pos(2) pos(2) + pos(4)], 'LineWidth', 1, 'LineStyle', "--");
                    end
                elseif n == 4
                    plot(timeEXP, evol.medios(cm).strains(m).data(j).mu, 'Color', map(j, :),
                         'LineWidth', 1);
                    hold on;
                    if j == size(INDUC, 1)
                        legend(INDUC)
                        xlabel('Time (h)');
                        ylabel('Growth rate (1/min)');
                        xlim([0 20])
                        title(strain(m, :));
                        grid on;
                        pos = get(hs, 'position');
                        annotation("line", (pos(1) + timePRE / timeEXP(end) * (pos(3))) * [1 1], ...
                            [pos(2) pos(2) + pos(4)], 'LineWidth', 1, 'LineStyle', "--");
                    end
                elseif n == 5
                    semilogy(timeEXP, (log(2) ./ evol.medios(cm).strains(m).data(j).mu), 'Color',
                         map(j, :), 'LineWidth', 1);
                    hold on;
                    if j == size(INDUC, 1)
                        legend(INDUC)
                        xlabel('Time (h)');
                        ylabel('Doubling time (min)');
                        xlim([0 20])
                        grid on;
                        pos = get(hs, 'position');
                        annotation("line", (pos(1) + timePRE / timeEXP(end) * (pos(3))) * [1 1], ...
                            [pos(2) pos(2) + pos(4)], 'LineWidth', 1, 'LineStyle', "--");
                    end
                end
            end
        end
    end
end

% Hill function
t1 = 6; t2 = 8; % Times for Hill Function
setime = round(t1 * 60 / 5);
endtime = round(t2 * 60 / 5);
strain_GFP = strain(1:3, :);
HillFunction_plotter(evol, map, strain_GFP, setime, endtime);

%Normalized Hill function
NormalizedHillFunc(evol, map, strain_GFP(:,:), setime, endtime);
```

Hill function calculation

```matlab
function expression = HillFunction_plotter(evol, map, strain, setime, endtime)

Marker_Counter = 1;
Markers = {'o','v','s','>','d','^','p'};
hf = figure('Name', 'Hill function');
ha = axes;
hp = [];

Vcell = 2e-15; % Liters
nA = 6.02214076e23; % Avogadro's number
molec_to_uM = 1e6 / (Vcell * nA); % uM Conversion factor

for m = 1:size(strain, 1)
    INDUC = char(evol.medios.strains(m).data.Inducc);

    if size(INDUC, 1) > 1
        SizeAHL = size(INDUC, 1);
        expression = zeros(1, size(INDUC, 1));
        R_std = zeros(1, size(INDUC, 1));

        inductions = str2num(INDUC(:, :));
        inductions(1) = 0.0001; % Inductor = 0 nM

        for j = 1:SizeAHL
            % FOD
            FOD = evol.medios.strains(m).data(j).MEFLcell(setime:endtime, :); % to start in j = 1 at
                 AHL = 0 nM
            R = FOD;

            expression(1, j) = mean(mean(R, 2), 1);
            R_var = mean(var(R, 0, 1)) + var(mean(R, 1)); % Law of total variance
            R_std(1, j) = sqrt(R_var);
            hp1 = errorbar(ha, inductions(j), expression(1, j), R_std(1, j), strcat('--',
                 Markers{Marker_Counter}), ...
                'MarkerSize', 8, 'MarkerFaceColor', map(j, :), 'Color', map(j, :), 'MarkerEdgeColor',
                     'k', 'LineWidth', 1);
            hold on;
        end
        hp = [hp, hp1];
        plot(inductions, expression, '--k');
    end
    Marker_Counter = Marker_Counter + 1;
end
set(gca, "XLim", [-1 inductions(end)]);
colormap(hf, map(1:SizeAHL, :));
xlabel('Inductor (nM)'); ylabel('GFP MEFL/Particles'); % ylabel({...,'(MEFL_{cell})'});
title('Hill function')
lgd = legend(hp, strain(:, :), 'Location', 'northeastoutside');
title(lgd, 'Strain');
grid on;
end
```

Normalized Hill function

```
function varargout = NormalizedHillFunc(evol, map, strain, setime, endtime)

Vcell = 2e-15; % Liters
nA = 6.02214076e23; % Avogadro's number
molec_to_uM = 1 / (Vcell * nA) * 1e6; % uM Conversion factor
molec_to_nM = 1 / (Vcell * nA) * 1e9;
map1 = map(:, 1:3);

total_ax = 2;

for m = 1:size(strain, 1)
    INDUC = char(evol.medios.strains(m).data.Inducc);
    figure('Name', strain(m, :));
    if size(INDUC, 1) > 1
        expression = zeros(1, size(INDUC, 1));
        R_std = zeros(1, size(INDUC, 1));

        inductions = str2num(INDUC(:, :));
        inductions(1) = 0.0001; % Inductor = 0 nM

        for j = 1:size(INDUC, 1)
            R = evol.medios.strains(m).data(j).MEFLcell(setime:endtime, :);
            expression(1, j) = median(mean(R, 2), 1);
            R_var = mean(var(R, 0, 1)) + var(mean(R, 1)); % Law of total variance
            R_std(1, j) = sqrt(R_var);
        end

        % Normalization
        Minvalue = min(expression);
        Maxvalue = max(expression);
        Nexpression = expression ./ Maxvalue;

        for k = 1:length(Nexpression)
            errorbar(inductions(k), Nexpression(1, k), 0 * R_std(1, k), 'd', 'MarkerSize', 6,
                 'MarkerFaceColor', map(k, :), ...
                'Color', map(k, :), 'MarkerEdgeColor', 'k', 'LineWidth', 1);
            er.Color = [0 0 0];
            er.LineStyle = 'none';
            hold on;
        end
        plot(inductions, Nexpression, '--k');
        xlabel('Inductor-TF (uM)'); ylabel('Promoter activity (normalized)');
        title(['GFP: ', strain(m, :)]);
        grid on;
    end

end

end
```

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

```
    clear
cd 'Getting all blanks' % File with Datafiles of the experiments
files = dir
names = {files.name};
names_exp = names(14:end);

Order experiments according their genetic constructions
conversion_list = [3 4 5 6 7 8 9 10 12];
for i = 3:length(names_exp)
    j = conversion_list(i);
    load(names_exp{i}); % Load experiment files
    for k = 1:2 %n = strains
        func_name = sprintf('Datagenerating_%d', j);
        evol = feval(func_name, evol, k, 1);
        exp_name = sprintf('exp_%d', j);
        if k == 1
            s1.(exp_name) = evol.medios.strains(1)
        else
            s2.(exp_name) = evol.medios.strains(2)
        end
    end
end

    %Rename
R4R4_16_E6 = s1.exp_6;
R3R3_2_E7 = s1.exp_7;
R4R3_7_E7 = s2.exp_7;
R3R3_2_E8 = s1.exp_8;
R3R4_5_E8 = s2.exp_8;
R4R3_7_E9 = s1.exp_9;
R4R4_16_E9 = s2.exp_9;
R3R4_5_E10 = s1.exp_10;
R4R4_16_E10 = s2.exp_10;
R3R4_2_E12 = s1.exp_12;
R4R4_4_E12 = s2.exp_12;

R3R3 = [R3R3_2_E7, R3R3_2_E8];
R3R4 = [R3R4_5_E10, R3R4_2_E12];
R4R3 = [R4R3_7_E7, R4R3_7_E9];
R4R4 = [R4R4_16_E6, R4R4_16_E9, R4R4_16_E10, R4R4_4_E12];

construc{1,1} = R3R3;
construc{1,2} = R3R4;
construc{2,1} = R4R3;
construc{2,2} = R4R4;
```

```
OBTAIN METRICS AND STORE IT IN ARRAYS AND TABLES.

% R structure contains 3 sub-structures, each repeated 4 times, corresponding to each construction.
    Each sub-structure includes:
% 1. The overall mean for that construction.
% 2. The mean for each experiment within that construction.
% 3. The mean in overall time of each construction.
% Within these sub-structures, data is organized into tables containing four main fields: OD,
    Fluorescence, MEFL and MEFLcell.

% 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, variating 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 accesed, OD, fluorescence or MEFLcell values will be provided.

R = {};
% V= {};
conv_list = [3, 4];
R_OD = {};
R_F = {};
R_Meflc = {};
R_Mu = {};
R_Mefl = {};

R_od = zeros(242, 11);
R_f = zeros(242, 11);
R_m_c = zeros(242, 11);
R_m = zeros(242, 11);
T = table();

for i = 1:size(construc, 1) % Access rows of construc
   for j = 1:size(construc, 2) % Access columns of construc
      % Arrays to store OD, F and MEFL values over time, exp and inductions
      OD_b = zeros(242, length(construc{i, j}), 11);
      F_b = zeros(242, length(construc{i, j}), 11);
      MEFLc = zeros(242, length(construc{i, j}), 11);
      Mefl = zeros(242, length(construc{i, j}), 11);
      mu = zeros(242, length(construc{i, j}), 11);
      % Columns of each construction varies (different N of exp included)
      % R3R3: 2 exp, R3R4: 3 exp, R4R3: 2 exp, R4R4: 4 exp
      % Arrays to store means of the experiments for each induction and time
      m_OD = zeros(242, 11);
      m_F = zeros(242, 11);
      m_MEFL = zeros(242, 11);
      m_MEFLcell = zeros(242, 11);
      % Arrays to store means of time for each induction
      m1t_OD = zeros(11, 1);
      m1t_F = zeros(11, 1);
      m1t_MEFL = zeros(11, 1);
      m1t_MEFLc = zeros(11, 1);

      for k = 1:length(construc{i, j}) % Exps done for each genetic construct
         for l = 1:length(construc{i, j}(k).data) % Inductions
            OD_b(:, k, l) = mean(construc{i, j}(k).data(l).ODblank(1:242, :), 2);
            F_b(:, k, l) = mean(construc{i,j}(k).data(l).Fblank(1:242, :), 2);
            MEFL(:, k, l) = mean(real(construc{i, j}(k).data(l).MEFLcell(1:242, :)), 2);
            MEFLc(:, k, l) = mean(real(construc{i, j}(k).data(l).MEFLcell(1:242, :)), 2);
            mu(:, k, l) = mean(real(construc{i, j}(k).data(l).mu(1:242, :)), 2);
         end
      end

      R_OD{i, j} = OD_b; % Cell stores values
      R_F{i, j} = F_b;
      R_MEFL{i, j} = MEFL;
      R_Meflc{i, j} = MEFLc;
      R_Mu{i, j} = mu;
```

```
        f = conv_list(i);
        p = conv_list(j);
        res = sprintf('Res_exp_%d_%d', f, p);

        % Create tables and store in the structure
        for h = 1:length(construc{i, j})
            R_od = OD_b(:, h, :);
            R_f = F_b(:, h, :);
            R_m = MEFL(:, h, :);
            R_m_c = MEFLc(:, h, :);

            T = table();
            T.OD = reshape(R_od, 242, 11);
            T.Fluorescence = reshape(R_f, 242, 11);
            T.MEFL = reshape(R_m, 242, 11);
            T.MEFLcell = reshape(R_m_c, 242, 11);
            T.Properties.VariableNames = {'OD', 'Fluorescence', 'MEFL','MEFLcell'};

            R.(res){h, 1} = T;
        % Table for the mean value of experiments for each induction
        m_OD(:,:) = mean(R_OD{i, j}, 2);
        m_F(:,:) = mean(R_F{i, j}, 2);
        m_MEFL(:,:) = mean(R_MEFL{i, j}, 2);
        m_MEFLcell(:,:) = mean(R_Meflc{i, j}, 2);
        res = sprintf('Res_%d_%d', f, p);
        R.(res) = table(m_OD, m_F, m_MEFL, m_MEFLcell, 'VariableNames', {'OD',
             'Fluorescence','MEFL', 'MEFLcell'});

        % Table for the mean value of each induction over time
        m1t_OD(:,1) = mean(m_OD, 1);
        m1t_F(:,1) = mean(m_F, 1);
        m1t_MEFL(:,1) = mean(m_MEFL, 1);
        m1t_MEFLc(:,1) = mean(m_MEFLcell, 1);
        res = sprintf('Res_1time_%d_%d', f, p);
        R.(res) = table(m1t_OD, m1t_F, m1t_MEFL, m1t_MEFLc, 'VariableNames', {'OD',
             'Fluorescence','MEFL', 'MEFLcell'});

    end
end

cd ..
```

Calculate performance metrics in specific time interval and its variances

```
% OBTAIN DATA IN TIME INTERVAL 150-175 AND ITS VARIANCES.
% To analyze the model, we selected the time interval from 150 to 175 based on the plots of the genetic constructions over time.

% Data Storage:
% Measurements is a cell array that stores 3 arrays: one for OD data, one for fluorescence data, and one for MEFLcell data.
% Each array has dimensions (242, 11, 4) = (time, inductions, constructions).

% Variance Calculation:
% 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).
% 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.
% 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)).

m_OD = zeros(242, 11, 4);
Fluor = zeros(242, 11, 4);
Mcell = zeros(242, 11, 4);

fields = {'Res_3_3', 'Res_3_4', 'Res_4_3', 'Res_4_4'};

for i = 1:length(fields)
    field_name = fields{i};
    m_OD(:,:,i) = R.(field_name).OD;
    Fluor(:,:,i) = R.(field_name).Fluorescence;
    Mcell(:,:,i) = R.(field_name).MEFLcell;
end

% Store measurements in a cell array
Measurements = {m_OD, Fluor, Mcell};

optd = zeros(11, 4); % Rows: inductions, columns: genetic constructions
fce = zeros(11, 4);
mce = zeros(11, 4);

v_optd = zeros(11, 4); % Rows: inductions, columns: genetic constructions
v_fce = zeros(11, 4);
v_mce = zeros(11, 4);
```

```
% Calculate means and variances for each construction and induction
for i = 1:size(Measurements, 1) % Rows
    for j = 1:size(Measurements, 2) % Columns
        type = Measurements{i, j};
        for k = 1:size(type, 3) % Constructions
            for l = 1:size(type, 2) % Inductions
                if j == 1
                    optd(l, k) = mean(type(150:175, l, k));
                    v_optd(l, k) = var(type(150:175, l, k)); % Variance along 150:175
                elseif j == 2
                    fce(l, k) = mean(type(150:175, l, k));
                    v_fce(l, k) = var(type(150:175, l, k));
                elseif j == 3
                    mce(l, k) = mean(type(150:175, l, k));
                    v_mce(l, k) = var(type(150:175, l, k));
                end
            end
        end
    end
end

% Variance of experiments in the construction (time 150:175). OD, F, and MEFL

V_exp_OD = cell(size(R_OD));
V_exp_F = cell(size(R_F));
V_exp_M = cell(size(R_Meflc));


fields = {R_OD, R_F, R_Meflc};
var_cells = {V_exp_OD, V_exp_F, V_exp_M};

% Calculate experimental variances
for h = 1:length(fields)
    field_name = fields{h};
    var_name = var_cells{h};
    for i = 1:size(field_name, 1) % Rows
        for j = 1:size(field_name, 2) % Columns
            for k = 1:size(field_name{i, j}, 3) % Inductions
                var_data = var(field_name{i, j}(150:175, :, k), 0, 2);
                mean_var = mean(var_data);
                var_name{i, j}(k, 1) = mean_var;
            end
        end
    end

    % Assign calculated variances to respective cells
    if h == 1
        V_exp_OD = var_name;
    elseif h == 2
        V_exp_F = var_name;
    elseif h == 3
        V_exp_M = var_name;
    end
end

% Law of total variance

V_Total = struct();
V_Total.VT_OD = cell(1, 1);
V_Total.VT_F = cell(1, 1);
V_Total.VT_M = cell(1, 1);
```

```
% Calculate total variance
l = 1;
for i = 1:size(V_exp_OD, 1) % Rows
    for j = 1:size(V_exp_OD, 2) % Columns
        for k = 1:size(V_exp_OD{i, j}, 1) % Inductions
            Vt_OD(k,:) = sqrt(V_exp_OD{i, j}(k) + v_optd(k, l));
            Vt_F(k,:) = sqrt(V_exp_F{i, j}(k) + v_fce(k, l));
            Vt_MEFL_R3R3(k,:) = sqrt(V_exp_M{i, j}(k) + v_mce(k, l));
        end
        V_Total.VT_OD{i, j} = Vt_OD;
        V_Total.VT_F{i, j} = Vt_F;
        V_Total.VT_M{i, j} = Vt_MEFL_R3R3;
        l = l + 1;
    end
end

% SAVE DATA IN FILES

% Data for R3R3
GFP_expression_var(1,:) = V_Total.VT_M{1,1}; % Variance of MEFLcell
for i = 1:size(R.Res_3_3.MEFLcell, 2) % MEFLcell expression in t = 150-175
    GFP_expression(:,i) = mean(R.Res_3_3.MEFLcell(150:175, i));
end
save('Data_R3R3.mat', 'GFP_expression', 'GFP_expression_var');

% Data for R4R4
GFP_expression_var(1,:) = V_Total.VT_M{2,2}; % Variance of MEFLcell
for i = 1:size(R.Res_4_4.MEFLcell, 2) % MEFLcell expression in t = 150-175
    GFP_expression(:,i) = mean(R.Res_4_4.MEFLcell(150:175, i));
end
save('Data_R4R4.mat', 'GFP_expression', 'GFP_expression_var');

% Data for R3R4
GFP_expression_var(1,:) = V_Total.VT_M{1,2}; % Variance of MEFLcell
for i = 1:size(R.Res_3_4.MEFLcell, 2) % MEFLcell expression in t = 150-175
    GFP_expression(:,i) = mean(R.Res_3_4.MEFLcell(150:175, i));
end
save('Data_R3R4.mat', 'GFP_expression', 'GFP_expression_var');

% Data for R4R3
GFP_expression_var(1,:) = V_Total.VT_M{2,1}; % Variance of MEFLcell
for i = 1:size(R.Res_4_3.MEFLcell, 2) % MEFLcell expression in t = 150-175
    GFP_expression(:,i) = mean(R.Res_4_3.MEFLcell(150:175, i));
end
save('Data_R4R3.mat', 'GFP_expression', 'GFP_expression_var');

% GROWTH RATE

Mean_mu = struct();
Mean_mu.R3R3 = zeros(11, 1);
Mean_mu.R3R4 = zeros(11, 1);
Mean_mu.R4R3 = zeros(11, 1);
Mean_mu.R4R4 = zeros(11, 1);

V_mu = struct();
V_mu.exp = zeros(11, 4);
V_mu.time = zeros(11, 4);

% Calculate mean growth rate
Mean_mu.R3R3(:, 1) = squeeze(mean(mean(R_Mu{1,1}(150:175, :, :), 1), 2));
Mean_mu.R3R4(:, 1) = squeeze(mean(mean(R_Mu{1,2}(150:175, :, :), 1), 2));
Mean_mu.R4R3(:, 1) = squeeze(mean(mean(R_Mu{2,1}(150:175, :, :), 1), 2));
Mean_mu.R4R4(:, 1) = squeeze(mean(mean(R_Mu{2,2}(150:175, :, :), 1), 2));
```

```
% Calculate variance
l = 1;
for i = 1:size(R_Mu, 1) % Rows
    for j = 1:size(R_Mu, 2) % Columns
        for k = 1:size(R_Mu{i,j}, 3) % Inductions
            V_mu.exp(k, l) = mean(var(R_Mu{i,j}(150:175, :, k), 0, 2));
            V_mu.time(k, l) = var(mean(R_Mu{i,j}(150:175, :, k), 2), 0, 1);
            V_total_mu(k, l) = sqrt(V_mu.exp(k, l) + V_mu.time(k, l));
        end
        l = l + 1;
    end
end

% Plot mean growth rate
figure;
plot(induc, Mean_mu.R3R3, '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
hold on;
plot(induc, Mean_mu.R3R4, '-s', 'LineWidth', 1.5, 'MarkerSize', 3);
plot(induc, Mean_mu.R4R3, '-^', 'LineWidth', 1.5, 'MarkerSize', 3);
plot(induc, Mean_mu.R4R4, '-d', 'LineWidth', 1.5, 'MarkerSize', 3);

title('Growth rate in constructions, t = 150:175');
xlabel('Induction');
ylabel('Mu function');
legend({'R3R3', 'R3R4', 'R4R3', 'R4R4'}, 'Location', 'northeastoutside');

grid on;
hold off;

% Plot variance of growth rate
figure;
plot(induc, V_total_mu(:, 1), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
hold on;
plot(induc, V_total_mu(:, 2), '-s', 'LineWidth', 1.5, 'MarkerSize', 3);
plot(induc, V_total_mu(:, 3), '-^', 'LineWidth', 1.5, 'MarkerSize', 3);
plot(induc, V_total_mu(:, 4), '-d', 'LineWidth', 1.5, 'MarkerSize', 3);

title('Variance of Growth rate in constructions, t = 150:175');
xlabel('Induction');
ylabel('Variance of mu function');
legend({'R3R3', 'R3R4', 'R4R3', 'R4R4'}, 'Location', 'northeastoutside');

grid on;
hold off;
```

Plot constructions

```
load("Results.mat")

time = 0.0543:0.083:20.0573;

INDUCC=char(evol.medios.strains(1).data.Inducc);
induc=str2num(INDUCC(:,:));
induc(1)=0.0001;

PLOT MEAN MEFLcell FOR EACH CONSTRUCTION OVER TIME INTERVAL 150-175

figure;

% First subplot
subplot(2, 2, 1);
plot(induc, R.Res_3_3.MEFLcell(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
title('R3R3');
xlabel('Induction');
ylabel('MEFL_{cell} value');
grid on;
hold on;

colormap(parula(length(legend_labels)));
colors = colormap;
line_objs = findobj(gca, 'Type', 'Line');
for i = 1:length(line_objs)
    set(line_objs(i), 'Color', colors(i,:));
end

% Second subplot
subplot(2, 2, 2);
semilogy(induc, R.Res_3_4.MEFLcell(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
title('R3R4');
xlabel('Induction');
ylabel('MEFL_{cell} value');
grid on;
hold on;

colormap(parula(length(legend_labels)));
colors = colormap;
line_objs = findobj(gca, 'Type', 'Line');
for i = 1:length(line_objs)
    set(line_objs(i), 'Color', colors(i,:));
end

% Third subplot
subplot(2, 2, 3);
plot(induc, R.Res_4_3.MEFLcell(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
title('R4R3');
xlabel('Induction');
ylabel('MEFL_{cell} value');
grid on;
hold on;

colormap(parula(length(legend_labels)));
colors = colormap;
line_objs = findobj(gca, 'Type', 'Line');
for i = 1:length(line_objs)
    set(line_objs(i), 'Color', colors(i,:));
end
```

```
% Fourth subplot
subplot(2, 2, 4);
plot(induc, R.Res_4_4.MEFLcell(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
title('R4R4');
xlabel('Induction');
ylabel('MEFL_{cell} value');
grid on;
hold on;

colormap(parula(length(legend_labels)));
colors = colormap;
line_objs = findobj(gca, 'Type', 'Line');
for i = 1:length(line_objs)
    set(line_objs(i), 'Color', colors(i,:));
end

subplot(2, 2, 1);
set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);

subplot(2, 2, 2);
set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);

subplot(2, 2, 3);
set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);

subplot(2, 2, 4);
set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);

PLOT MEAN MEFL FOR EACH CONSTRUCTION OVER TIME INTERVAL 150-175

figure;

% First subplot
subplot(2, 2, 1);
plot(induc, R.Res_3_3.MEFL(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
title('R3R3');
xlabel('Induction');
ylabel('MEFL_{cell} value');
grid on;
hold on;

colormap(parula(length(legend_labels)));
colors = colormap;
line_objs = findobj(gca, 'Type', 'Line');
for i = 1:length(line_objs)
    set(line_objs(i), 'Color', colors(i,:));
end

% Second subplot
subplot(2, 2, 2);
semilogy(induc, R.Res_3_4.MEFL(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
title('R3R4');
xlabel('Induction');
ylabel('MEFL value');
grid on;
hold on;

colormap(parula(length(legend_labels)));
colors = colormap;
line_objs = findobj(gca, 'Type', 'Line');
for i = 1:length(line_objs)
    set(line_objs(i), 'Color', colors(i,:));
end
```

```
% Third subplot
subplot(2, 2, 3);
plot(induc, R.Res_4_3.MEFL(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
title('R4R3');
xlabel('Induction');
ylabel('MEFL value');
grid on;
hold on;

colormap(parula(length(legend_labels)));
colors = colormap;
line_objs = findobj(gca, 'Type', 'Line');
for i = 1:length(line_objs)
    set(line_objs(i), 'Color', colors(i,:));
end

% Fourth subplot
subplot(2, 2, 4);
plot(induc, R.Res_4_4.MEFL(150:175,:), '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
title('R4R4');
xlabel('Induction');
ylabel('MEFL value');
grid on;
hold on;

colormap(parula(length(legend_labels)));
colors = colormap;
line_objs = findobj(gca, 'Type', 'Line');
for i = 1:length(line_objs)
    set(line_objs(i), 'Color', colors(i,:));
end

subplot(2, 2, 1);
set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);

subplot(2, 2, 2);
set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);

subplot(2, 2, 3);
set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);

subplot(2, 2, 4);
set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);

PLOT MEAN OD FOR EACH CONSTRUCTION OVER TIME

figure;

% First subplot
subplot(2, 2, 1);
plot(time, R.Res_3_3.OD, '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
title('R3R3');
xlabel('Time [h]');
ylabel('OD_{600} value');
grid on;
hold on;

colormap(parula(length(legend_labels)));
colors = colormap;
line_objs = findobj(gca, 'Type', 'Line');
for i = 1:length(line_objs)
    set(line_objs(i), 'Color', colors(i,:));
end
```

```
% Second subplot
subplot(2, 2, 2);
plot(time, R.Res_3_4.OD, '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
title('R3R4');
xlabel('Time [h]');
ylabel('OD_{600} value');
grid on;
hold on;

colormap(parula(length(legend_labels)));
colors = colormap;
line_objs = findobj(gca, 'Type', 'Line');
for i = 1:length(line_objs)
    set(line_objs(i), 'Color', colors(i,:));
end

% Third subplot
subplot(2, 2, 3);
plot(time, R.Res_4_3.OD, '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
title('R4R3');
xlabel('Time [h]');
ylabel('OD_{600} value');
grid on;
hold on;

colormap(parula(length(legend_labels)));
colors = colormap;
line_objs = findobj(gca, 'Type', 'Line');
for i = 1:length(line_objs)
    set(line_objs(i), 'Color', colors(i,:));
end

% Fourth subplot
subplot(2, 2, 4);
plot(time, R.Res_4_4.OD, '-o', 'LineWidth', 1.5, 'MarkerSize', 3);
title('R4R4');
xlabel('Time [h]');
ylabel('OD_{600} value');
grid on;
hold on;

colormap(parula(length(legend_labels)));
colors = colormap;
line_objs = findobj(gca, 'Type', 'Line');
for i = 1:length(line_objs)
    set(line_objs(i), 'Color', colors(i,:));
end

subplot(2, 2, 1);
set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);

subplot(2, 2, 2);
set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);

subplot(2, 2, 3);
set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);

subplot(2, 2, 4);
set(gca, 'Position', get(gca, 'Position') - [0, 0.05, 0, 0]);
```

Simulation

```
clear

% Load data from R3R3, R4R3, and R4R4
D = struct();
D.GFP_exp_var = zeros(3, 11);
D.GFP_exp = zeros(3, 11);

l = 1;
for j = 3:4
    for i = 3:4
        data_name = sprintf('Data_R%dR%d.mat', j, i);
        load(data_name)
        if strcmp(data_name, 'Data_R3R4.mat')
            continue
        else
            D.GFP_exp_var(l,:) = GFP_expression_var;
            D.GFP_exp(l,:) = GFP_expression;
            Data_exp{j-2,i-2}.m = GFP_expression;
            Data_exp{j-2,i-2}.v = GFP_expression_var;
        end
        l = l + 1;
    end
end

% Fixed model parameters
p.kP1 = 0.56; % Transcription rate constant
p.dm = 0.399; % mRNA degradation rate
p.dP = 0.0002; % Protein degradation rate
p.CN = 40; % Copy number of the plasmid
p.mu = log(2) / 30; % Growth rate constant

% Induction
p.NAR_induction = [0, 30, 75, 100, 125, 145, 175, 250, 400, 700, 1400];

p.Data = Data_exp;

% Bounds for parameters to estimate
lower_bounds = [2, 0.1, 0.1, 0.1, 1, 1, 1, 0.005, 0.001, 0.009, 8, 0.9, 600];
upper_bounds = [2, 5, 30, 5, 1, 1, 1, 0.02, 0.02, 0.2, 9, 1.4, 730];

% Optimization
options = optimoptions('ga', 'Display', 'iter', 'MaxGenerations', 100000);
[best_param, Jmin_value] = ga(@(param) Jprom(param, p), 13, [], [], [], [], lower_bounds,
    upper_bounds, [], options);

% Assign optimized values to model parameters
p.nFde0 = best_param(1); % Hill coefficient
kFde0_v = best_param(2:4);
kdNar_v = best_param(5:7);
beta_v = best_param(8:10);
RBS = best_param(11:12);
p.kFde = best_param(13);
```

```
% Simulate and plot results
nar = 0:1400;
figure;
k = 0;
names = {'R3R3', 'R4R3', 'R4R4'};
for RBS_TFi = 1:2
    p.RBS_TF = RBS(RBS_TFi);
    for RBS_BSi = 1:2
        p.RBS_BS = RBS(RBS_BSi);
        if (RBS_TFi == 1 && RBS_BSi == 2)
            continue;
        end
        k = k + 1;
        p.kFde0 = kFde0_v(k);
        p.kdNar = kdNar_v(1);
        p.beta = beta_v(k);
        GFP = evaluate_pFde0(nar, p);
        GFP_exp = p.Data{RBS_TFi, RBS_BSi}.m;
        GFP_var = p.Data{RBS_TFi, RBS_BSi}.v;

        subplot(3, 1, k);
        semilogy(nar, GFP, 'LineWidth', 1);
        hold on;
        errorbar(p.NAR_induction, GFP_exp, GFP_var, '-d', 'LineWidth', 1, 'MarkerSize', 1.5);
        hold off;

        title(['Expression of GFP ' names{k}]);
        xlabel('Induction');
        ylabel('GFP');
        grid on;
    end
end

% Jprom function: calculates mean squared error between experimental and model GFP
function RMLSE = Jprom(param, p)
    % Model parameters = current optimization values
    p.nFde0 = param(1);
    kFde0_v = param(2:4);
    kdNar_v = param(5:7);
    beta_v = param(8:10);
    RBS = param(11:12);
    p.kFde = param(13);

    RMLSE_parciales = [];
    NAR = p.NAR_induction;
    k = 0;
    for RBS_TFi = 1:2
        RBS_TF = RBS(RBS_TFi);
        for RBS_BSi = 1:2
            RBS_BS = RBS(RBS_BSi);
            if (RBS_TFi == 1 && RBS_BSi == 2)
                continue;
            end
            k = k + 1;

            kFde0 = kFde0_v(k);
            kdNar = kdNar_v(1);
            beta = beta_v(k);
```

```matlab
            % Steady state calculation
            p.Fder_ss = RBS_TF * p.CN * p.kP1 / ((p.dm + p.mu) * (p.dP + p.mu));
            FdeR = p.Fder_ss;

            % pFde0 calculation
            pFde0 = beta + (1 - beta) * NAR.^p.nFde0 ./ ...
                (p.kFde * (kdNar * p.CN / FdeR).^p.nFde0 + NAR.^p.nFde0);

            % KEXP calculation
            KEXP = RBS_BS * p.CN * kFde0 / (p.dm + p.mu);

            % GFP calculation
            GFP = KEXP .* pFde0 / (p.dP + p.mu);

            % Calculate partial RMLSE
            GFP_exp = p.Data{RBS_TFi, RBS_BSi}.m;
            J = sqrt(sum((log10(GFP_exp + 1) - log10(GFP + 1)).^2));
            if (RBS_TFi == 2 && RBS_BSi == 2)
                J = J * 0;
            end
            if (RBS_TFi == 1 && RBS_BSi == 1)
                J = J * 1.5;
            end
            RMLSE_parciales = [RMLSE_parciales J];
        end
    end
    RMLSE = mean(RMLSE_parciales);
end

% evaluate_pFde0 function: predicts GFP expression given NAR concentration and parameters
function [GFP] = evaluate_pFde0(NAR, p)
    p.Fder_ss = p.RBS_TF * p.CN * p.kP1 / ((p.dm + p.mu) * (p.dP + p.mu));
    pFde0 = p.beta + (1 - p.beta) * NAR.^p.nFde0 ./ ...
        (p.kFde * (p.kdNar * p.CN / p.Fder_ss).^p.nFde0 + NAR.^p.nFde0);

    KEXP = p.RBS_BS * p.CN * p.kFde0 / (p.dm + p.mu);
    GFP = KEXP .* pFde0 / (p.dP + p.mu);
end
```