

```

% Implement three functions:
% 1. Multi-component concentration trends over time
% 2. Detailed light intensity-response analysis (focus on S and S*)
% 3. Sobol parameter sensitivity analysis (including convergence analysis)

clear all; close all; clc;

%% Model parameter definition - calibrated according to modeling documentation
% CcaS related parameters
params.Vmax_g = 0.1; % Maximum green light response rate s-1
params.Vmax_r = 0.05; % Maximum red light response rate s-1
params.I_g_50 = 50; % Green light intensity for half-maximal response lx
params.I_r_50 = 50; % Red light intensity for half-maximal response lx
params.k_sstar = 0.0005; % CcaS spontaneous inactivation rate constant s-1
params.n_hill = 2; % Hill coefficient

% CcaR phosphorylation parameters
params.V1_R = 0.1; % CcaR phosphorylation maximum rate μM/s
params.V1star_R = 0.05; % CcaR dephosphorylation maximum rate μM/s
params.V2_R = 0.08; % Phosphatase maximum rate μM/s
params.V2star_R = 0.04; % Reverse phosphorylation maximum rate μM/s
params.K_R1 = 10; % Michaelis constant for CcaS on CcaR μM
params.K_R2 = 15; % Michaelis constant for phosphatase on RP μM
params.K_RP1star = 8; % Inhibition constant for CcaS on RP μM
params.K_RP2star = 12; % Inhibition constant for phosphatase on R μM

% CheA parameters
params.Vmax_A = 0.2; % CheA maximum production rate μM/s
params.kdA = 5; % CheA half-activation concentration μM
params.beta_A = 0.005; % CheA degradation rate s-1
params.n_A = 2.5; % CheA Hill coefficient

% CheY phosphorylation parameters
params.V1_Y = 0.15; % CheY phosphorylation maximum rate μM/s
params.V1star_Y = 0.07; % CheY dephosphorylation maximum rate μM/s
params.V2_Y = 0.09; % CheZ maximum rate μM/s
params.V2star_Y = 0.06; % Reverse action maximum rate μM/s
params.K_Y1 = 20; % Michaelis constant for CheA on CheY μM
params.K_Y2 = 25; % Michaelis constant for CheZ on YP μM
params.K_YP1star = 18; % Inhibition constant for CheA on YP μM
params.K_YP2star = 22; % Inhibition constant for CheZ on Y μM

% Light intensity parameters

```

```

params.I_g = 100; % Green light intensity lx
params.I_r = 50; % Red light intensity lx

fprintf('Biological system light control model - Starting simulation...\n');

%% Function 1: Multi-component concentration trends over time
fprintf('\n=== Function 1: Multi-component concentration trends over time ===\n');

% Initial conditions: S* initial concentration is 0, others are random positive values
rng(1); % Fixed random seed
y0_random = abs(randn(7,1)) .* [0; 14; 10; 3; 6; 15; 4]; % S* initial is 0

% Simulation time
tspan = [0 500];

% Solve differential equations
[t, y] = ode15s(@bacterial_system(t,y,params), tspan, y0_random);

% Extract component concentrations
S_star = y(:,1); % Active CcaS
S = y(:,2); % Inactive CcaS
R = y(:,3); % CcaR
R_P = y(:,4); % Phosphorylated CcaR
A = y(:,5); % CheA
Y = y(:,6); % CheY
Y_P = y(:,7); % Phosphorylated CheY

% Plot multi-component concentration trends
fig1 = figure('Position', [100, 100, 1000, 600]);
set(fig1, 'Name', 'Function 1: Multi-component concentration over time');

% CcaS system dynamics
ax1 = subplot(2,2,1);
plot(t, S_star, 'r-', 'LineWidth', 2); hold on;
plot(t, S, 'b-', 'LineWidth', 2);
legend('S* (Active CcaS)', 'S (Inactive CcaS)', 'Location', 'best');
set(get(ax1, 'Title'), 'String', 'CcaS Kinetics');
set(get(ax1, 'XLabel'), 'String', 'Time (s)');
set(get(ax1, 'YLabel'), 'String', 'Concentration ( $\mu$ M)');
set(ax1, 'XGrid', 'on', 'YGrid', 'on');

% CcaR phosphorylation dynamics
ax2 = subplot(2,2,2);
plot(t, R, 'g-', 'LineWidth', 2); hold on;

```

```

plot(t, R_P, 'm-', 'LineWidth', 2);
legend('R (CcaR)', 'R_P (Phosphorylated CcaR)', 'Location', 'best');
set(get(ax2, 'Title'), 'String', 'CcaR Phosphorylation Kinetics');
set(get(ax2, 'XLabel'), 'String', 'Time (s)');
set(get(ax2, 'YLabel'), 'String', 'Concentration ( $\mu\text{M}$ )');
set(ax2, 'XGrid', 'on', 'YGrid', 'on');

% CheY system dynamics
ax3 = subplot(2,2,3);
plot(t, Y, 'c-', 'LineWidth', 2); hold on;
plot(t, Y_P, 'y-', 'LineWidth', 2);
legend('Y (CheY)', 'Y_P (Phosphorylated CheY)', 'Location', 'best');
set(get(ax3, 'Title'), 'String', 'CheY Phosphorylation Kinetics');
set(get(ax3, 'XLabel'), 'String', 'Time (s)');
set(get(ax3, 'YLabel'), 'String', 'Concentration ( $\mu\text{M}$ )');
set(ax3, 'XGrid', 'on', 'YGrid', 'on');

% Key active components comparison
ax4 = subplot(2,2,4);
plot(t, S_star, 'r-', t, R_P, 'm-', t, Y_P, 'y-', 'LineWidth', 1.5);
legend('S*', 'R_P', 'Y_P', 'Location', 'best');
set(get(ax4, 'Title'), 'String', 'Key Active Components Comparison');
set(get(ax4, 'XLabel'), 'String', 'Time (s)');
set(get(ax4, 'YLabel'), 'String', 'Concentration ( $\mu\text{M}$ )');
set(ax4, 'XGrid', 'on', 'YGrid', 'on');

% Display steady-state values
steady_state = y(end,:);
fprintf('Concentrations at system steady state:\n');
fprintf('S* = %.4f  $\mu\text{M}$ , S = %.4f  $\mu\text{M}$ \n', steady_state(1), steady_state(2));
fprintf('R = %.4f  $\mu\text{M}$ , R_P = %.4f  $\mu\text{M}$ \n', steady_state(3), steady_state(4));
fprintf('CheA = %.4f  $\mu\text{M}$ , CheY = %.4f  $\mu\text{M}$ , CheY_P = %.4f  $\mu\text{M}$ \n', ...
steady_state(5), steady_state(6), steady_state(7));

% Mass conservation verification
total_CcaS = S_star + S;
total_CcaR = R + R_P;
total_CheY = Y + Y_P;

fprintf('\nMass conservation verification:\n');
fprintf('CcaS total variation range: %.6f  $\mu\text{M}$ \n', max(total_CcaS) - min(total_CcaS));
fprintf('CcaR total variation range: %.6f  $\mu\text{M}$ \n', max(total_CcaR) - min(total_CcaR));
fprintf('CheY total variation range: %.6f  $\mu\text{M}$ \n', max(total_CheY) - min(total_CheY));
fprintf('CheA has degradation term, concentration from %.4f to %.4f  $\mu\text{M}$ \n', A(1), A(end));

```

```

%% Function 2: Detailed light intensity-response analysis (focus on S and S*)
fprintf('\n=== Function 2: Detailed light intensity-response analysis (focus on S and S*) ===\n');

% Use steady state from Function 1 as initial condition (achieved under light conditions)
y0_steady = steady_state';

% 2.1 Determine appropriate simulation time
fprintf('Determining appropriate simulation time...\n');

% Test several typical light intensity conditions to observe system convergence time
test_conditions = [
0, 0; % No light
100, 0; % High green light
0, 100; % High red light
100, 100; % Antagonistic condition
];

% Simulate longer time for each condition to observe convergence
test_tspan = [0 5000]; % Extended simulation time to observe convergence

fig2 = figure('Position', [100, 100, 1000, 600]);
set(fig2, 'Name', 'Function 2: System convergence time test');
colors = lines(size(test_conditions, 1));

for i = 1:size(test_conditions, 1)
current_params = params;
current_params.I_g = test_conditions(i, 1);
current_params.I_r = test_conditions(i, 2);
[t_test, y_test] = ode15s(@(t,y) bacterial_system(t,y,current_params), ...
test_tspan, y0_steady);
% Plot S* convergence process
subplot(2,2,1);
plot(t_test, y_test(:,1), 'Color', colors(i,:), 'LineWidth', 1.5, ...
'DisplayName', sprintf('G%dR%d', test_conditions(i,1), test_conditions(i,2)));
hold on;
% Plot S convergence process
subplot(2,2,2);
plot(t_test, y_test(:,2), 'Color', colors(i,:), 'LineWidth', 1.5, ...
'DisplayName', sprintf('G%dR%d', test_conditions(i,1), test_conditions(i,2)));
hold on;
% Calculate relative changes to determine convergence
S_star_rel_change = abs(diff(y_test(:,1))) ./ y_test(1:end-1,1);
S_rel_change = abs(diff(y_test(:,2))) ./ y_test(1:end-1,2);

```

```

% Find convergence time (relative change less than 0.1%)
conv_threshold = 0.001;
S_star_conv_idx = find(S_star_rel_change < conv_threshold, 1);
S_conv_idx = find(S_rel_change < conv_threshold, 1);
if ~isempty(S_star_conv_idx)
    fprintf('Condition G%dR%d: S* converged at %.1f seconds\n', ...
        test_conditions(i,1), test_conditions(i,2), t_test(S_star_conv_idx));
else
    fprintf('Condition G%dR%d: S* did not fully converge within simulation time\n', ...
        test_conditions(i,1), test_conditions(i,2));
end
end

ax1 = subplot(2,2,1);
set(get(ax1, 'Title'), 'String', 'S* Convergence Process');
set(get(ax1, 'XLabel'), 'String', 'Time (s)');
set(get(ax1, 'YLabel'), 'String', 'S* Concentration ( $\mu$ M)');
legend('show', 'Location', 'best');
set(ax1, 'XGrid', 'on', 'YGrid', 'on');

ax2 = subplot(2,2,2);
set(get(ax2, 'Title'), 'String', 'S Convergence Process');
set(get(ax2, 'XLabel'), 'String', 'Time (s)');
set(get(ax2, 'YLabel'), 'String', 'S Concentration ( $\mu$ M)');
legend('show', 'Location', 'best');
set(ax2, 'XGrid', 'on', 'YGrid', 'on');

% Based on test results, determine appropriate simulation time
% Observed that most conditions converge within 1000 seconds, conservatively choose 1500 seconds
optimal_simulation_time = 1500;
fprintf('\nSelected simulation time: %d seconds\n', optimal_simulation_time);

% 2.2 Fine light intensity scan (0.2 lx interval)
green_intensities = 0:0.2:200; % Green light intensity, 0.2 lx interval
red_intensities = 0:0.2:200; % Red light intensity, 0.2 lx interval

% Store results
S_star_green = zeros(length(green_intensities), 1);
S_green = zeros(length(green_intensities), 1);
S_star_red = zeros(length(red_intensities), 1);
S_red = zeros(length(red_intensities), 1);
convergence_times_green = zeros(length(green_intensities), 1);
convergence_times_red = zeros(length(red_intensities), 1);

```

```
fprintf('Performing fine light intensity scan (0.2 lx interval, simulating %d seconds)...\\n',
optimal_simulation_time);
```

```
% Green light only effect (red light = 0) - using steady state from Function 1 as initial condition
for i = 1:length(green_intensities)
```

```
current_params = params;
```

```
current_params.I_g = green_intensities(i);
```

```
current_params.I_r = 0;
```

```
% Use optimized simulation time
```

```
tspan_optimized = [0 optimal_simulation_time];
```

```
[t_temp, y_temp] = ode15s(@(t,y) bacterial_system(t,y,current_params), ...
```

```
tspan_optimized, y0_steady);
```

```
S_star_green(i) = y_temp(end, 1);
```

```
S_green(i) = y_temp(end, 2);
```

```
% Calculate convergence time (relative change less than 0.1%)
```

```
S_star_rel_change = abs(diff(y_temp(:,1))) ./ y_temp(1:end-1,1);
```

```
conv_idx = find(S_star_rel_change < 0.001, 1);
```

```
if ~isempty(conv_idx)
```

```
convergence_times_green(i) = t_temp(conv_idx);
```

```
else
```

```
convergence_times_green(i) = optimal_simulation_time;
```

```
end
```

```
% Display progress
```

```
if mod(i, 200) == 0
```

```
fprintf(' Green light scan progress: %d/%d, S* = %.4f, convergence time = %.1f\\n', ...
```

```
i, length(green_intensities), S_star_green(i), convergence_times_green(i));
```

```
end
```

```
end
```

```
% Red light only effect (green light = 0) - using steady state from Function 1 as initial condition
```

```
for i = 1:length(red_intensities)
```

```
current_params = params;
```

```
current_params.I_g = 0;
```

```
current_params.I_r = red_intensities(i);
```

```
% Use optimized simulation time
```

```
tspan_optimized = [0 optimal_simulation_time];
```

```
[t_temp, y_temp] = ode15s(@(t,y) bacterial_system(t,y,current_params), ...
```

```
tspan_optimized, y0_steady);
```

```
S_star_red(i) = y_temp(end, 1);
```

```
S_red(i) = y_temp(end, 2);
```

```
% Calculate convergence time (relative change less than 0.1%)
```

```
S_star_rel_change = abs(diff(y_temp(:,1))) ./ y_temp(1:end-1,1);
```

```
conv_idx = find(S_star_rel_change < 0.001, 1);
```

```
if ~isempty(conv_idx)
```

```

convergence_times_red(i) = t_temp(conv_idx);
else
convergence_times_red(i) = optimal_simulation_time;
end
% Display progress
if mod(i, 200) == 0
fprintf(' Red light scan progress: %d/%d, S* = %.4f, convergence time = %.1fs\n', ...
i, length(red_intensities), S_star_red(i), convergence_times_red(i));
end
end

% Analyze convergence time statistics
fprintf('\nConvergence time analysis:\n');
fprintf('Green light conditions average convergence time: %.1f ± %.1f seconds\n', ...
mean(convergence_times_green), std(convergence_times_green));
fprintf('Red light conditions average convergence time: %.1f ± %.1f seconds\n', ...
mean(convergence_times_red), std(convergence_times_red));
fprintf('Maximum convergence time: %.1f seconds\n', max([convergence_times_green;
convergence_times_red]));

% Verify no-light condition results
fprintf('No-light condition verification (I_g=0, I_r=0):\n');
fprintf('S* = %.6f μM\n', S_star_green(1));
fprintf('S = %.6f μM\n', S_green(1));
fprintf('Convergence time = %.1f seconds\n', convergence_times_green(1));

% Plot fine light intensity response curves
fig3 = figure('Position', [100, 100, 1200, 800]);
set(fig3, 'Name', 'Function 2: Fine light intensity response analysis');

% Green light response - concentration
ax1 = subplot(2,2,1);
plot(green_intensities, S_star_green, 'r-', 'LineWidth', 1.5); hold on;
plot(green_intensities, S_green, 'b-', 'LineWidth', 1.5);
legend('S* Steady State Concentration', 'S Steady State Concentration', 'Location', 'best');
set(get(ax1, 'Title'), 'String', 'Green Light Effect on S and S* (I_r=0)');
set(get(ax1, 'XLabel'), 'String', 'Green Light Intensity (lx)');
set(get(ax1, 'YLabel'), 'String', 'Concentration (μM)');
set(ax1, 'XGrid', 'on', 'YGrid', 'on');

% Mark no-light condition point
plot(0, S_star_green(1), 'ro', 'MarkerSize', 8, 'LineWidth', 2);
plot(0, S_green(1), 'bo', 'MarkerSize', 8, 'LineWidth', 2);

```

```

% Red light response - concentration
ax2 = subplot(2,2,2);
plot(red_intensities, S_star_red, 'r-', 'LineWidth', 1.5); hold on;
plot(red_intensities, S_red, 'b-', 'LineWidth', 1.5);
legend('S* Steady State Concentration', 'S Steady State Concentration', 'Location', 'best');
set(get(ax2, 'Title'), 'String', 'Red Light Effect on S and S* (I_g=0)');
set(get(ax2, 'XLabel'), 'String', 'Red Light Intensity (I_x)');
set(get(ax2, 'YLabel'), 'String', 'Concentration ( $\mu$ M)');
set(ax2, 'XGrid', 'on', 'YGrid', 'on');

% Mark no-light condition point
plot(0, S_star_red(1), 'ro', 'MarkerSize', 8, 'LineWidth', 2);
plot(0, S_red(1), 'bo', 'MarkerSize', 8, 'LineWidth', 2);

% Green light response - convergence time
ax3 = subplot(2,2,3);
plot(green_intensities, convergence_times_green, 'g-', 'LineWidth', 1.5);
set(get(ax3, 'Title'), 'String', 'Green Light Effect on Convergence Time (I_r=0)');
set(get(ax3, 'XLabel'), 'String', 'Green Light Intensity (I_x)');
set(get(ax3, 'YLabel'), 'String', 'Convergence Time (s)');
set(ax3, 'XGrid', 'on', 'YGrid', 'on');

% Red light response - convergence time
ax4 = subplot(2,2,4);
plot(red_intensities, convergence_times_red, 'm-', 'LineWidth', 1.5);
set(get(ax4, 'Title'), 'String', 'Red Light Effect on Convergence Time (I_g=0)');
set(get(ax4, 'XLabel'), 'String', 'Red Light Intensity (I_x)');
set(get(ax4, 'YLabel'), 'String', 'Convergence Time (s)');
set(ax4, 'XGrid', 'on', 'YGrid', 'on');

% 2.3 3D response surface analysis (only analyze S and S*)
fprintf('Performing 3D response surface analysis (S and S*)...\n');

% Create light intensity grid
[I_g_grid, I_r_grid] = meshgrid(0:20:200, 0:20:200);
S_star_grid = zeros(size(I_g_grid));
S_grid = zeros(size(I_g_grid));
convergence_grid = zeros(size(I_g_grid));

% Calculate response for each grid point - using steady state from Function 1 as initial condition
for i = 1:size(I_g_grid, 1)
    for j = 1:size(I_g_grid, 2)
        current_params = params;
        current_params.I_g = I_g_grid(i, j);

```

```

current_params.I_r = I_r_grid(i, j);
% Use optimized simulation time
[t_temp, y_temp] = ode15s(@(t,y) bacterial_system(t,y,current_params), ...
[0 optimal_simulation_time], y0_steady);
S_star_grid(i, j) = y_temp(end, 1);
S_grid(i, j) = y_temp(end, 2);
% Calculate convergence time
S_star_rel_change = abs(diff(y_temp(:,1))) ./ y_temp(1:end-1,1);
conv_idx = find(S_star_rel_change < 0.001, 1);
if ~isempty(conv_idx)
convergence_grid(i, j) = t_temp(conv_idx);
else
convergence_grid(i, j) = optimal_simulation_time;
end
end
fprintf(' Completed %d/%d grid row calculations\n', i, size(I_g_grid, 1));
end

% Plot 3D response surfaces
fig4 = figure('Position', [100, 100, 1500, 400]);
set(fig4, 'Name', 'Function 2: 3D Response Surface Analysis');

% S* 3D response surface
ax1 = subplot(1,3,1);
surf(I_g_grid, I_r_grid, S_star_grid, 'EdgeColor', 'none');
set(get(ax1, 'XLabel'), 'String', 'Green Light Intensity (lx)');
set(get(ax1, 'YLabel'), 'String', 'Red Light Intensity (lx)');
set(get(ax1, 'ZLabel'), 'String', 'S* Concentration ( $\mu$ M)');
set(get(ax1, 'Title'), 'String', 'S* Concentration Response Surface');
colorbar;
view(45, 30);

% S 3D response surface
ax2 = subplot(1,3,2);
surf(I_g_grid, I_r_grid, S_grid, 'EdgeColor', 'none');
set(get(ax2, 'XLabel'), 'String', 'Green Light Intensity (lx)');
set(get(ax2, 'YLabel'), 'String', 'Red Light Intensity (lx)');
set(get(ax2, 'ZLabel'), 'String', 'S Concentration ( $\mu$ M)');
set(get(ax2, 'Title'), 'String', 'S Concentration Response Surface');
colorbar;
view(45, 30);

% Convergence time 3D response surface
ax3 = subplot(1,3,3);

```

```

surf(I_g_grid, I_r_grid, convergence_grid, 'EdgeColor', 'none');
set(get(ax3, 'XLabel'), 'String', 'Green Light Intensity (lx)');
set(get(ax3, 'YLabel'), 'String', 'Red Light Intensity (lx)');
set(get(ax3, 'ZLabel'), 'String', 'Convergence Time (s)');
set(get(ax3, 'Title'), 'String', 'System Convergence Time Response Surface');
colorbar;
view(45, 30);

% Display key data points
fprintf("\nKey response data:\n");
fprintf('%-25s %-12s %-12s %-15s\n', 'Light Condition', 'S* Conc', 'S Conc', 'Conv Time(s)');

% No-light condition
fprintf('%-25s %-12.4f %-12.4f %-15.1f\n', 'No Light (0,0)', ...
S_star_grid(1,1), S_grid(1,1), convergence_grid(1,1));

% High green light condition
[max_green, idx] = max(S_star_green);
fprintf('%-25s %-12.4f %-12.4f %-15.1f\n', sprintf('High Green (%d,0)', green_intensities(idx)), ...
max_green, S_green(idx), convergence_times_green(idx));

% High red light condition
[max_red, idx] = max(S_star_red);
fprintf('%-25s %-12.4f %-12.4f %-15.1f\n', sprintf('High Red (0,%d)', red_intensities(idx)), ...
max_red, S_red(idx), convergence_times_red(idx));

% Antagonistic condition
fprintf('%-25s %-12.4f %-12.4f %-15.1f\n', 'Antagonistic (100,100)', ...
S_star_grid(6,6), S_grid(6,6), convergence_grid(6,6));

fprintf("\nSimulation setup summary:\n");
fprintf('Simulation time: %d seconds\n', optimal_simulation_time);
fprintf('Convergence threshold: relative change < 0.1%%\n');
fprintf('Light intensity scan interval: 0.2 lx\n');

%% Function 3: Sobol parameter sensitivity analysis (including convergence analysis)
fprintf("\n=== Function 3: Sobol parameter sensitivity analysis (including convergence analysis)
===\n");

% Use steady state from Function 1 as initial condition
y0_steady = steady_state';

% Select key parameters for analysis
param_names = {'Vmax_g', 'Vmax_r', 'Vmax_A', 'beta_A', 'V1_Y', 'V1star_Y'};

```

```

n_params = length(param_names);

% Parameter variation range ( $\pm 20\%$ )
param_ranges = [
params.Vmax_g * 0.8, params.Vmax_g * 1.2; % Vmax_g
params.Vmax_r * 0.8, params.Vmax_r * 1.2; % Vmax_r
params.Vmax_A * 0.8, params.Vmax_A * 1.2; % Vmax_A
params.beta_A * 0.8, params.beta_A * 1.2; % beta_A
params.V1_Y * 0.8, params.V1_Y * 1.2; % V1_Y
params.V1star_Y * 0.8, params.V1star_Y * 1.2; % V1star_Y
];

%% 3.1 Convergence analysis
fprintf('Performing Sobol convergence analysis...\n');

% Define sample size gradient
sample_sizes = [200, 400, 600, 800, 1000];
n_sizes = length(sample_sizes);
n_repeats = 2; % 2 repeats for each sample size

% Store convergence analysis results
mean_sensitivity = zeros(n_params, n_sizes);

for size_idx = 1:n_sizes
n_current = sample_sizes(size_idx);
fprintf(' Sample size %d/%d: %d samples\n', size_idx, n_sizes, n_current);
% Store repeat results for current sample size
sensitivity_repeats = zeros(n_repeats, n_params);
for repeat = 1:n_repeats
% Generate parameter samples (Latin Hypercube Sampling)
param_samples = lhsdesign(n_current, n_params);
for i = 1:n_params
param_samples(:,i) = param_ranges(i,1) + ...
(param_ranges(i,2) - param_ranges(i,1)) .* param_samples(:,i);
end
% Calculate output for each parameter sample (CheY_P steady state concentration)
outputs = zeros(n_current, 1);
for j = 1:n_current
current_params = params;
current_params.Vmax_g = param_samples(j,1);
current_params.Vmax_r = param_samples(j,2);
current_params.Vmax_A = param_samples(j,3);
current_params.beta_A = param_samples(j,4);
current_params.V1_Y = param_samples(j,5);

```

```

current_params.VIstar_Y = param_samples(j,6);
[~, y_temp] = ode15s(@(t,y) bacterial_system(t,y,current_params), ...
[0 2000], y0_steady);
outputs(j) = y_temp(end, 7); % CheY_P steady state concentration
end

% Calculate sensitivity metric (absolute value of correlation coefficient)
for i = 1:n_params
R = corrcoef(param_samples(:,i), outputs);
sensitivity_repeats(repeat, i) = abs(R(1,2));
end
end

% Calculate mean
mean_sensitivity(:, size_idx) = mean(sensitivity_repeats, 1);
end

% Plot convergence analysis
fig5 = figure('Position', [100, 100, 800, 500]);
set(fig5, 'Name', 'Function 3: Sobol Convergence Analysis');

colors = lines(n_params);
for i = 1:n_params
plot(sample_sizes, mean_sensitivity(i,:), 'Color', colors(i,:), ...
'LineWidth', 2, 'DisplayName', param_names{i});
hold on;
end

% Determine recommended sample size (based on convergence)
convergence_threshold = 0.01; % 1% change threshold
recommended_sample = sample_sizes(end);

for i = n_sizes-1:-1:2
max_change = 0;
for j = 1:n_params
relative_change = abs(mean_sensitivity(j,i) - mean_sensitivity(j,i-1)) / mean_sensitivity(j,i-1);
if relative_change > max_change
max_change = relative_change;
end
end
if max_change < convergence_threshold
recommended_sample = sample_sizes(i);
fprintf(' Convergence achieved at sample size %d (maximum change: %.2f%%)\n',
recommended_sample, max_change*100);
break;
end
end

```

```

end

% Mark recommended sample size
line([recommended_sample, recommended_sample], ylim, 'Color', 'red', ...
'LineStyle', '--', 'LineWidth', 2, 'DisplayName', ...
sprintf('Recommended sample size: %d', recommended_sample));

set(get(gca, 'XLabel'), 'String', 'Sample Size');
set(get(gca, 'YLabel'), 'String', 'Sensitivity Metric (|Correlation|)');
set(get(gca, 'Title'), 'String', 'Sobol Sensitivity Analysis Convergence');
legend('show', 'Location', 'best');
set(gca, 'XGrid', 'on', 'YGrid', 'on');

fprintf('\nRecommended sample size: %d\n', recommended_sample);

%% 3.2 Final sensitivity analysis using recommended sample size
fprintf('\nPerforming final sensitivity analysis using recommended sample size %d...\n',
recommended_sample);

% Generate final parameter samples
param_samples = lhsdesign(recommended_sample, n_params);
for i = 1:n_params
    param_samples(:,i) = param_ranges(i,1) + ...
    (param_ranges(i,2) - param_ranges(i,1)) .* param_samples(:,i);
end

% Calculate final output
outputs = zeros(recommended_sample, 1);
fprintf('Performing final simulations...\n');

for i = 1:recommended_sample
    current_params = params;
    current_params.Vmax_g = param_samples(i,1);
    current_params.Vmax_r = param_samples(i,2);
    current_params.Vmax_A = param_samples(i,3);
    current_params.beta_A = param_samples(i,4);
    current_params.V1_Y = param_samples(i,5);
    current_params.V1star_Y = param_samples(i,6);
    [~, y_temp] = ode15s(@(t,y) bacterial_system(t,y,current_params), ...
[0 2000], y0_steady);
    outputs(i) = y_temp(end, 7); % CheY_P steady state concentration
    if mod(i, 100) == 0
        fprintf(' Completed %d/%d simulations\n', i, recommended_sample);
    end
end

```

```

end

% Calculate final sensitivity metrics
final_sensitivity = zeros(n_params, 1);
for i = 1:n_params
    R = corrcoef(param_samples(:,i), outputs);
    final_sensitivity(i) = abs(R(1,2));
end

% Plot final sensitivity results
fig6 = figure('Position', [100, 100, 600, 400]);
set(fig6, 'Name', 'Function 3: Sobol Final Sensitivity Analysis');

bar(final_sensitivity);
set(gca, 'XTickLabel', param_names, 'XTickLabelRotation', 45);
set(get(gca, 'XLabel'), 'String', 'Parameter');
set(get(gca, 'YLabel'), 'String', 'Sensitivity Metric (|Correlation|)');
set(get(gca, 'Title'), 'String', sprintf('Sobol Parameter Sensitivity Analysis (Sample Size: %d)',
recommended_sample));
set(gca, 'XGrid', 'on', 'YGrid', 'on');

% Display sensitivity ranking
[~, idx] = sort(final_sensitivity, 'descend');
fprintf('\nFinal parameter sensitivity ranking:\n');
for i = 1:n_params
    fprintf('%d. %s: %.4f\n', i, param_names{idx(i)}, final_sensitivity(idx(i)));
end

fprintf('\n=== All functions completed ===\n');

% Define differential equation system function
function dydt = bacterial_system(t, y, params)
% State variables
S_star = y(1); % Active CcaS
S = y(2); % Inactive CcaS
R = y(3); % CcaR
R_P = y(4); % Phosphorylated CcaR
A = y(5); % CheA
Y = y(6); % CheY
Y_P = y(7); % Phosphorylated CheY
% CcaS light response kinetics
green_response = params.Vmax_g * (params.I_g^params.n_hill) / ...
(params.I_g^params.n_hill + params.I_g_50^params.n_hill);
red_response = params.Vmax_r * (params.I_r^params.n_hill) / ...

```

```

(params.I_r^params.n_hill + params.I_r_50^params.n_hill);
% CcaS activity change
dS_star_dt = green_response * S - red_response * S_star - params.k_sstar * S_star;
dS_dt = -green_response * S + red_response * S_star + params.k_sstar * S_star;
% CcaR phosphorylation kinetics
dR_dt = -params.V1_R * S_star * R / (params.K_R1 + R) + ...
params.V1star_R * R_P / (params.K_RP1star + R_P) + ...
params.V2star_R * R_P / (params.K_RP2star + R_P) - ...
params.V2_R * R / (params.K_R2 + R);
dR_P_dt = params.V1_R * S_star * R / (params.K_R1 + R) - ...
params.V1star_R * R_P / (params.K_RP1star + R_P) - ...
params.V2star_R * R_P / (params.K_RP2star + R_P) + ...
params.V2_R * R / (params.K_R2 + R);
% CheA kinetics
dA_dt = params.Vmax_A * (R_P^params.n_A) / (params.kdA^params.n_A + R_P^params.n_A) - ...
params.beta_A * A;
% CheY phosphorylation kinetics
dY_dt = -params.V1_Y * A * Y / (params.K_Y1 + Y) + ...
params.V1star_Y * Y_P / (params.K_YP1star + Y_P) + ...
params.V2star_Y * Y_P / (params.K_YP2star + Y_P) - ...
params.V2_Y * Y / (params.K_Y2 + Y);
dY_P_dt = params.V1_Y * A * Y / (params.K_Y1 + Y) - ...
params.V1star_Y * Y_P / (params.K_YP1star + Y_P) - ...
params.V2star_Y * Y_P / (params.K_YP2star + Y_P) + ...
params.V2_Y * Y / (params.K_Y2 + Y);
% Assemble derivative vector
dydt = [dS_star_dt; dS_dt; dR_dt; dR_P_dt; dA_dt; dY_dt; dY_P_dt];
end

```

%% Differential equation systems-Strictly follow the modeling document

```
function dydt = bacterial_system(t, y, params)
```

```
% state variable
```

```
S_star = y(1);% Active CcaS concentration
```

```
S = y(2);% Non-reactive CcaS concentration
```

```
R = y(3);% CcaR concentration
```

```
R_P = y(4);% phosphorylated CcaR concentration
```

```
A = y(5);% CheA concentration
```

```
Y = y(6);% CheY concentration
```

```
Y_P = y(7);% phosphorylated CheY concentration
```

```
% Extraction parameters
```

```
Vmax_g = params.Vmax_g;
```

```
Vmax_r = params.Vmax_r;
```

```
I_g = params.I_g;
```

```
I_r = params.I_r;
```

```

I_g_50 = params.I_g_50;
I_r_50 = params.I_r_50;
k_sstar = params.k_sstar;
n_hill = params.n_hill;
V1_R = params.V1_R;
V1star_R = params.V1star_R;
V2_R = params.V2_R;
V2star_R = params.V2star_R;
K_R1 = params.K_R1;
K_R2 = params.K_R2;
K_RP1star = params.K_RP1star;
K_RP2star = params.K_RP2star;
Vmax_A = params.Vmax_A;
kdA = params.kdA;
beta_A = params.beta_A;
n_A = params.n_A;
V1_Y = params.V1_Y;
V1star_Y = params.V1star_Y;
V2_Y = params.V2_Y;
V2star_Y = params.V2star_Y;
K_Y1 = params.K_Y1;
K_Y2 = params.K_Y2;
K_YP1star = params.K_YP1star;
K_YP2star = params.K_YP2star;
% Equation (1): d[S*]/dt
green_activation = (Vmax_g * I_g^n_hill) / (I_g^n_hill + I_g_50^n_hill) * S;
red_inactivation = (Vmax_r * I_r^n_hill) / (I_r^n_hill + I_r_50^n_hill) * S_star;
spontaneous_inactivation = k_sstar * S_star;
dS_star_dt = green_activation - red_inactivation - spontaneous_inactivation;
% Equation (2): d[S]/dt
dS_dt = -green_activation + spontaneous_inactivation + red_inactivation;
Calculation of% CcaR phosphorylation rate (equation 5-8)
% Equation (5): v1[R]
denom1 = 1 + R/K_R1 + R_P/K_RP1star;
v1_R = (V1_R * R / K_R1) / denom1;
% Equation (6): v1*[R_P]
v1star_RP = (V1star_R * R_P / K_RP1star) / denom1;
% Equation (7): v2[R]
denom2 = 1 + R/K_R2 + R_P/K_RP2star;
v2_R = (V2_R * R / K_R2) / denom2;
% Equation (8): v2*[R_P]
v2star_RP = (V2star_R * R_P / K_RP2star) / denom2;
% Equation (3): d[R]/dt
dR_dt = -v1_R + v1star_RP - v2_R + v2star_RP;

```

```

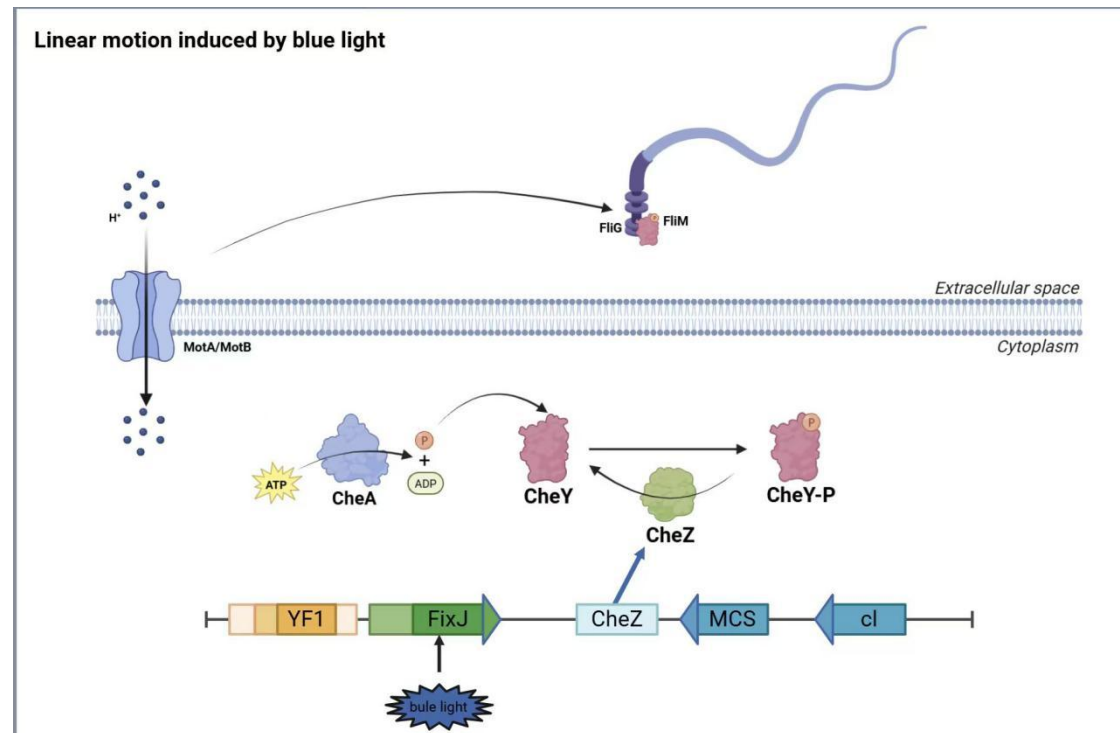
% Equation (4): d[R_P]/dt
dR_P_dt = v1_R - v1star_RP + v2_R - v2star_RP;
% Equation (9): d[A]/dt
dA_dt = (Vmax_A * R_P^n_A) / (kdA^n_A + R_P^n_A) - beta_A * A;
Calculation of phosphorylation rate of CheY (Equation 12-15)
% Equation (12): v1[Y]
denom3 = 1 + Y/K_Y1 + Y_P/K_YP1star;
v1_Y = (V1_Y * Y / K_Y1) / denom3;
% Equation (13): v1*[Y_P]
v1star_YP = (V1star_Y * Y_P / K_YP1star) / denom3;
% Equation (14): v2[Y]
denom4 = 1 + Y/K_Y2 + Y_P/K_YP2star;
v2_Y = (V2_Y * Y / K_Y2) / denom4;
% Equation (15): v2*[Y_P]
v2star_YP = (V2star_Y * Y_P / K_YP2star) / denom4;
% Equation (10): d[Y]/dt
dY_dt = -v1_Y + v1star_YP - v2_Y + v2star_YP;
% Equation (11): d[Y_P]/dt
dY_P_dt = v1_Y - v1star_YP + v2_Y - v2star_YP;
% Return to differential equations
dydt = [dS_star_dt; dS_dt; dR_dt; dR_P_dt; dA_dt; dY_dt; dY_P_dt];
end

```

1、Protein component analysis of E. coli rectal motility under blue light regulation

component	function
YF1	Phosphokinase, blue light inactivates it
FixJ	Response regulator protein, phosphorylated to activate downstream gene transcription
pFixK2	The promoter is activated by phosphorylation of FixJ, which activates downstream gene expression
CheZ	Phosphatase, dephosphorylation of CheY-P, reduced the frequency of cell swimming rotation and promoted straight-line motion

component	function
CheY	Response regulator protein, phosphorylated form (CheY-P) promotes cellular tumbling



Running logic:

Under blue light: YF1 is inactivated, FixJ can not be phosphorylated → inhibit pFixK2 promoter → decrease of cI protein → increase of CheZ expression → dephosphorylation of CheZ CheY-P → decrease of CheY-P → enhance of linear movement of cells.

Catalog of parameters

Parameter name	Biological implications	unit
<i>YF1*</i>	Phosphokinase activity concentration	μM
<i>YF1</i>	Phosphokinase nonactive concentration	μM
<i>F</i>	FixJ concentration	μM
<i>FP</i>	Concentration of phosphorylated FixJ	μM
<i>Z</i>	Concentration of CheZ	μM
<i>Y</i>	Concentration of CheY	μM
<i>YP</i>	Concentration of CheY_P	μM
<i>cI</i>	Concentration of cI	μM

I_B	Blue light intensity	lx
I_{B_50}	The light intensity at which the maximum response rate of 50% blue light is reached	lx
β_{cI}	C I degradation rate constant	s^{-1}
β_Z	CheZ degradation rate constant	s^{-1}
$V_{max_Fp_cI}$	The maximum reaction rate of phosphorylation FixJ activation cI	$\mu M/s$
$V_{max_cI_Z}$	C I inhibits the maximum reaction rate of CheZ	$\mu M/s$
V_{max_B}	Maximum response speed of blue light	$\mu M/s$
k_{dF}	cI half-activation concentration	μM
k_{dcI}	Cherz and cI apparent dissociation constants	μM
K_{F1}	Milk constant of substrate FixJ for active enzyme YF1	μM
K_{F2}	Mie constant of phosphatase phosphorylation of substrate FixJ	μM
K_{FP1}^*	Inhibitory constant of active enzyme CcaS on product phosphorylation FixJ	μM
K_{FP2}^*	Inhibitory constant of phosphatase for product FixJ	μM
K_{Y1}	Mie constant of substrate CheY for enzyme CheA	μM
K_{Y2}	Mie constant of substrate phosphorylation CheY by enzyme CheZ	μM
K_{YP1}^*	Inhibition constant of enzyme CheA on phosphorylated CheY product	μM
K_{YP2}^*	Inhibition constant of enzyme CheZ on product CheY	μM
$v_1[X]$	$(X \rightarrow X_P)$ phosphorylation positive flux	$\mu M/s$
$v_1^*[X_P]$	$(X_P \rightarrow X)$ phosphorylation reverse flux	$\mu M/s$
$v_2[X]$	$(X \rightarrow X_P)$ dephosphorylation reverse flux	$\mu M/s$
$v_2^*[X_P]$	$(X_P \rightarrow X)$ dephosphorylation positive flux	$\mu M/s$

First, when exposed to blue light: The YF1 structure undergoes structural alteration and inactivation, losing its ability to phosphorylate the FixJ protein. At this stage, we believe that the rate of YF1 protein inactivation is influenced by two factors: the intensity of blue light and the concentration of YF1. It's important to note that for light-sensitive proteins, the response to light is extremely rapid, allowing us to formulate the following equation:

$$\frac{d[YF1]}{dt} = \frac{V_{max_B}(I_B)^n}{(I_B)^n + (I_{B_50})^n} [YF1^*] - k_{YF1} [YF1] \quad (n=2) \quad (1)$$

$$\frac{d[YF1^*]}{dt} = -\frac{V_{max_B}(I_B)^n}{(I_B)^n + (I_{B_50})^n} [YF1^*] + k_{YF1} [YF1] \quad (n=2) \quad (2)$$

Next, for FixJ phosphorylation and dephosphorylation, we constructed our equations using steady-state enzyme kinetics formulas reported in the literature:

$$\frac{d[F_P]}{dt} = v_1[F] - v_1^*[F_P] + v_2[F] - v_2^*[F_P] \quad (3)$$

$$\frac{d[F]}{dt} = -v_1[F] + v_1^*[F_P] - v_2[F] + v_2^*[F_P] \quad (4)$$

among :

$$v_1[F] = \frac{\frac{V_1[F]}{K_{F1}}}{1 + \frac{[F]}{K_{F1}} + \frac{[F_P]}{K_{FP1}^*}} \quad (5)$$

$$v_1^*[F_P] = \frac{\frac{V_1^*[F_P]}{K_{FP1}^*}}{1 + \frac{[F]}{K_{F1}} + \frac{[F_P]}{K_{FP1}^*}} \quad (6)$$

$$v_2[F] = \frac{\frac{V_2[F]}{K_{F2}}}{1 + \frac{[F]}{K_{F2}} + \frac{[F_P]}{K_{FP2}^*}} \quad (7)$$

$$v_2^*[F_P] = \frac{\frac{V_2^*[F_P]}{K_{FP2}^*}}{1 + \frac{[F]}{K_{F2}} + \frac{[F_P]}{K_{FP2}^*}} \quad (8)$$

Next, we describe the relationship between phosphorylated FixJ and the repressor protein cI (using Hill equations to describe it):

$$\frac{d[cI]}{dt} = \frac{V_{\max_Fp_cI}[F_p]^n}{K_{dF}^n + [F_p]^n} - \beta_{cI}[cI], \quad (n=1.5) \quad (9)$$

Then the interaction between cI and cheZ:

$$\frac{d[Z]}{dt} = \frac{V_{\max_cI_Z}K_{dcI}^n}{K_{dcI}^n + [cI]^n} - \beta_Z[Z] \quad (n=2.5) \quad (10)$$

Finally, for the transition between cheY and cheY_P:

$$\frac{d[Y]}{dt} = -v_1[Y] + v_1^*[Y_P] - v_2[Y] + v_2^*[Y_P] \quad (11)$$

$$\frac{d[Y_P]}{dt} = v_1[Y] - v_1^*[Y_P] + v_2[Y] - v_2^*[Y_P] \quad (12)$$

among :

$$v_1[Y] = \frac{\frac{V_1[Y]}{K_{Y1}}}{1 + \frac{[Y]}{K_{Y1}} + \frac{[Y_P]}{K_{YP1}^*}} \quad (13)$$

$$v_1^*[Y_P] = \frac{\frac{V_1^*[Y_P]}{K_{YP1}^*}}{1 + \frac{[Y]}{K_{Y1}} + \frac{[Y_P]}{K_{YP1}^*}} \quad (14)$$

$$v_2[Y] = \frac{\frac{V_2[Y]}{K_{Y2}}}{1 + \frac{[Y]}{K_{Y2}} + \frac{[Y_P]}{K_{YP2}^*}} \quad (15)$$

$$v_2^*[Y_P] = \frac{\frac{V_2^*[Y_P]}{K_{YP2}^*}}{1 + \frac{[Y]}{K_{Y2}} + \frac{[Y_P]}{K_{YP2}^*}} \quad (16)$$

Similarly, we carry out numerical simulation based on the above model to assist experimental research. The simulated values of each parameter are as follows

Table 2.1 Parameter value estimation table

Parameter name	Biological implications	unit	Initial values	Source of parameters
<i>YFI*</i>	Phosphokinase activity concentration	μM	0.8	estimate
<i>YFI</i>	Phosphokinase nonactive concentration	μM	0.2	estimate
<i>F</i>	FixJ concentration	μM	8	estimate

FP	Concentration of phosphorylated FixJ	μM	5	estimate
Z	Concentration of Chez	μM	0.1	estimate
Y	Concentration of Chey	μM	8	estimate
YP	Concentration of Chey_P	μM	5	estimate
cI	Concentration of cI	μM	0.5	estimate
I_B	Blue light intensity	lx	100	estimate
I_{B_50}	The light intensity at which the maximum response rate of 50% blue light is reached	lx	50	estimate
β_{cI}	C I degradation rate constant	s^{-1}	0.03	estimate
β_Z	Chez degradation rate constant	s^{-1}	0.02	estimate
$V_{max_Fp_cI}$	The maximum reaction rate of phosphorylation FixJ activation cI	$\mu M/s$	0.4	estimate
$v_{max_cI_Z}$	C I inhibits the maximum reaction rate of CheZ	$\mu M/s$	0.35	estimate
V_{max_B}	Maximum response speed of blue light	$\mu M/s$	0.8	estimate
k_{dF}	cI half-activation concentration	μM	5	estimate
k_{dcI}	Cherz and cI apparent dissociation constants	μM	10	estimate
K_{F1}	Milk constant of substrate FixJ for active enzyme YF1	μM	3	estimate
K_{F2}	Mie constant of phosphatase phosphorylation of substrate FixJ	μM	4	estimate
K_{FP1}^*	Inhibitory constant of active enzyme CcaS on product phosphorylation FixJ	μM	6	estimate
K_{FP2}^*	Inhibitory constant of phosphatase for product FixJ	μM	5	estimate

K_{Y1}	Mie constant of substrate CheY for enzyme CheA	μM	3	estimate
K_{Y2}	Mie constant of substrate phosphorylation CheY by enzyme CheZ	μM	2.5	estimate
K_{YP1}^*	Inhibition constant of enzyme CheA on phosphorylated CheY product	μM	4	estimate
K_{YP2}^*	Inhibition constant of enzyme CheZ on product CheY ($X \rightarrow X_P$)	μM	3.5	estimate
$v_1[X]$	phosphorylation positive flux ($X_P \rightarrow X$)	$\mu M/s$	/	estimate
$v_1^*[X_P]$	phosphorylation reverse flux ($X \rightarrow X_P$)	$\mu M/s$	/	estimate
$v_2[X]$	dephosphorylation reverse flux ($X_P \rightarrow X$)	$\mu M/s$	/	estimate
$v_2^*[X_P]$	dephosphorylation positive flux	$\mu M/s$	/	estimate

The analytical approach for this pathway is fundamentally consistent with the previous study. First, we conducted numerical simulations using the aforementioned parameters to obtain Figure 2.1. Our findings reveal that YF1 and YF1* reached steady states within an extremely short duration (<12s), which aligns with the magnitude of results from literature reviews—photosensitive proteins undergo conformational changes under light stimulation within brief periods, thereby altering their functions. Other key components (CheZ, CheY_P) achieved steady states over approximately 300 seconds after blue light activation, consistent with our expectations. Western blot analysis demonstrated increased levels of these critical components following prolonged illumination. Repeated wet-lab experiments confirmed minimal temporal variations in CheZ and CheY_P concentrations (less than 5%), which corroborates our simulation results.

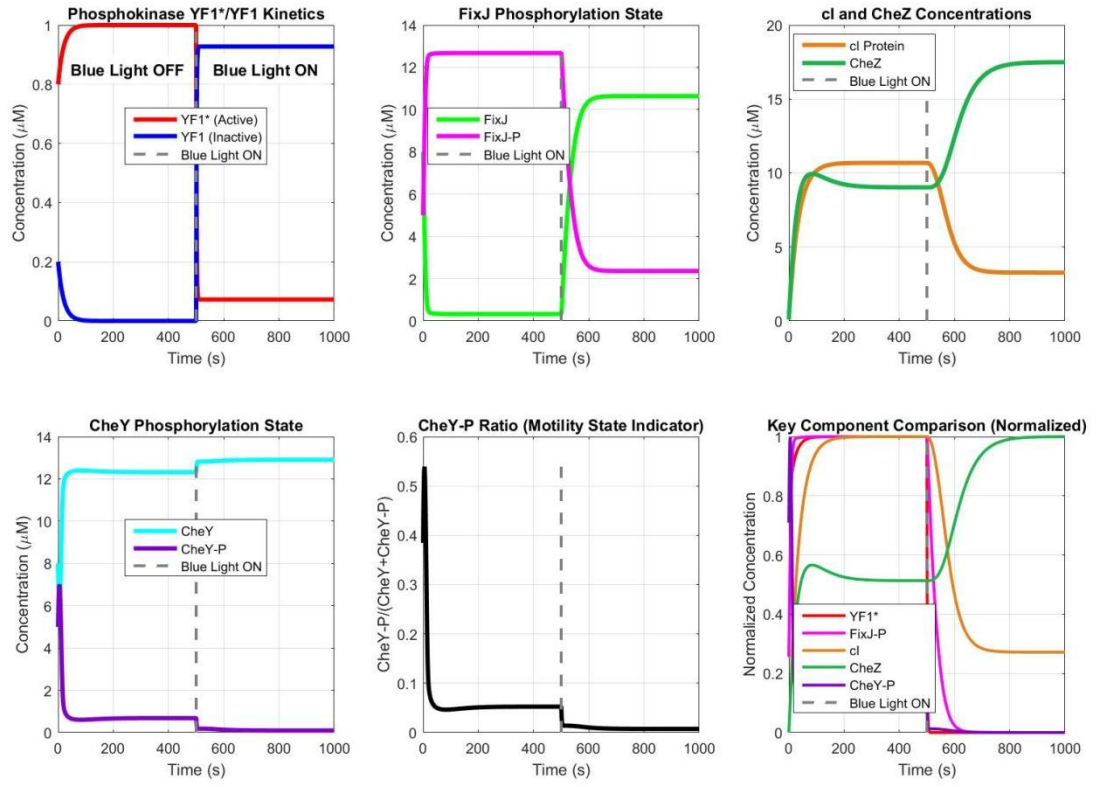


Figure 2.1 Concentration curves of each component over time (dark for $I_B=100$ the first 500s, after 500s)

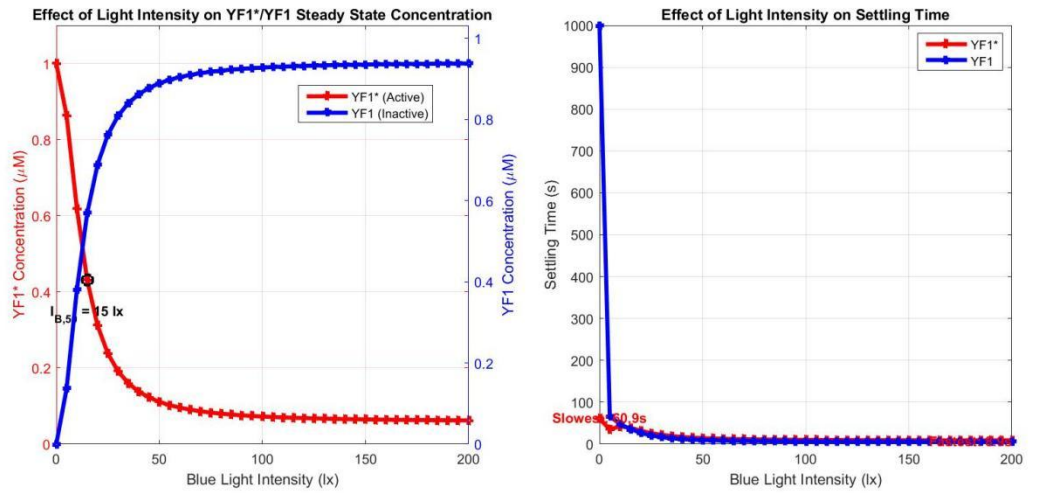


Figure 2.2 Light intensity response analysis

Furthermore, we attempted to analyze the specific effects of light intensity on proteins YF1 and YF1* using the aforementioned parameter values. It should be noted that we have not found direct literature specifying their interaction mechanisms. Through computer simulations capturing protein concentration data after 2000 seconds of blue light exposure (Figure 2.2), it is evident that both YF1 and YF1*

exhibit extreme sensitivity to blue light. At an irradiance of IB=15 lx, they reach half-saturation light intensity. To evaluate the rationality of our illumination duration settings, we conducted steady-state time monitoring across multiple lighting conditions. As shown in the right panel of Figure 2.2, low light intensities (approximately 15 lx) can rapidly activate the proteins within <6 seconds. While it should be emphasized that our simulations utilize estimated parameters which may deviate significantly from real-world conditions, we have nevertheless established a framework and methodology for analytical purposes.

Finally, to identify the parameters with the most significant impact on the model's output, we conducted parameter sensitivity analysis. Seven key parameters (Vmax_B, IB_50, k_YF1, V1_F, V2_Y, Vmax_Fp_cI, Vmax_cI_Z) were selected for $\pm 10\%$ relative perturbations while keeping other parameters constant. After each disturbance, the ODE system was re-integrated during the blue light activation phase (500-1000 s), and the steady-state CheY-P concentration Y_p at $t = 1000s$ was extracted.

The sensitivity index S_i is defined as follows:

$$S_i = \frac{1}{2} \left(\frac{|Y_P^{+10\%} - Y_P^{ref}|}{Y_P^{ref}} + \frac{|Y_P^{-10\%} - Y_P^{ref}|}{Y_P^{ref}} \right) / 10\%$$

among :

Y_P^{ref} : CheY-P steady-state concentration under reference category;

$Y_P^{+10\%}$, $Y_P^{-10\%}$: CheY-P steady-state concentration after increasing or decreasing the parameters by 10%;

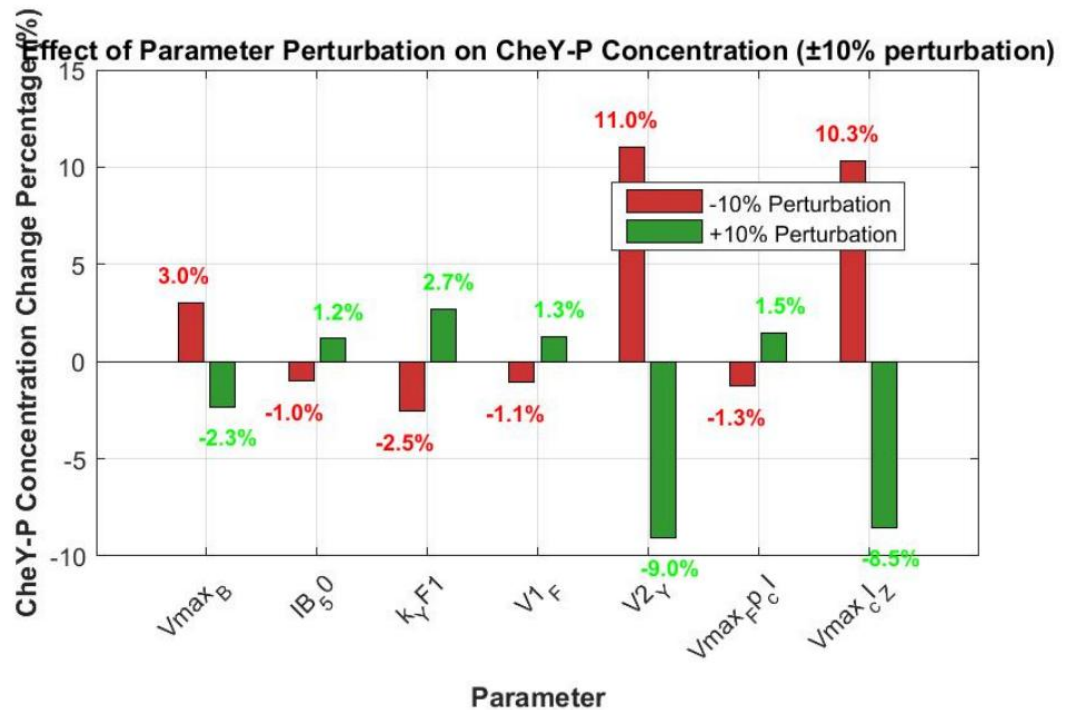


Figure 1.5 Parameter sensitivity analysis (1)

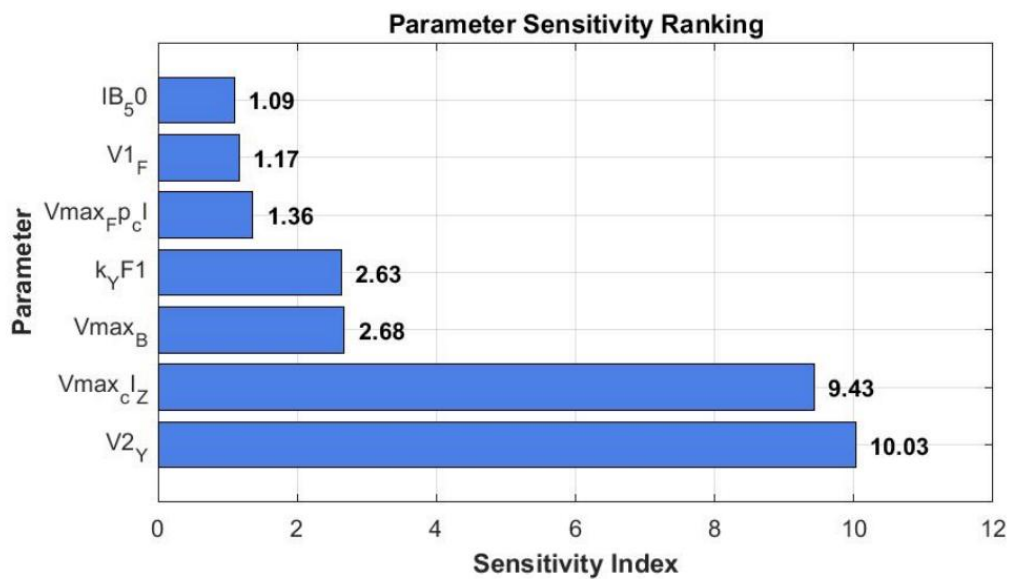


Figure 1.6 Parameter sensitivity parameter ranking (2)

According to the simulation results, we found that the maximum v_{2-Y} reaction rate of CheY phosphorylase and the maximum response rate of blue light were the most sensitive to the biological system, which provided a preferred target for subsequent experimental verification.

Here is the code:%%% Revised Protein Phosphorylation System Simulation-MATLAB 2015b

Compatible Version

clear all; close all; clc;

```

% Add the encoding setting at the beginning of the code
feature('DefaultCharacterSet', 'UTF-8');
slCharacterEncoding('UTF-8');
%% Parameter Definition (extracted from documentation)
% Initial concentrations - adjusted to reflect steady state with blue light off
YF1_star_0 = 0.8; %  $\mu\text{M}$  (more YF1* when blue light off)
YF1_0 = 0.2; %  $\mu\text{M}$  (less YF1 when blue light off)
F_0 = 8; %  $\mu\text{M}$ 
Fp_0 = 5; %  $\mu\text{M}$  (more Fp when blue light off, as YF1* is active)
Z_0 = 0.1; %  $\mu\text{M}$ 
Y_0 = 8; %  $\mu\text{M}$ 
Yp_0 = 5; %  $\mu\text{M}$  (more Yp when blue light off)
cI_0 = 0.5; %  $\mu\text{M}$  (higher cI expression when blue light off)

% System parameters
IB = 100; % lx (blue light on)
IB_50 = 50; % lx
beta_cI = 0.03; %  $\mu\text{M}/\text{s}$ 
beta_Z = 0.02; %  $\mu\text{M}/\text{s}$ 
Vmax_Fp_cI = 0.4; %  $\mu\text{M}/\text{s}$ 
Vmax_cI_Z = 0.35; %  $\mu\text{M}/\text{s}$ 
Vmax_B = 0.8; %  $\text{s}^{-1}$  (increased light response speed)
kdF = 5; %  $\mu\text{M}$  (decreased to make cI more sensitive to Fp)
kdcI = 10; %  $\mu\text{M}$  (decreased to make Z more sensitive to cI)

% Michaelis constants and inhibition constants
KF1 = 3; %  $\mu\text{M}$  (decreased to improve YF1* affinity for FixJ)
KF2 = 4; %  $\mu\text{M}$ 
KFP1_star = 6; %  $\mu\text{M}$ 
KFP2_star = 5; %  $\mu\text{M}$ 
KY1 = 3; %  $\mu\text{M}$ 
KY2 = 2.5; %  $\mu\text{M}$ 
KYP1_star = 4; %  $\mu\text{M}$ 
KYP2_star = 3.5; %  $\mu\text{M}$ 

% Adjusted rate constants to enhance blue light effect
V1_F = 1.5; %  $\mu\text{M}/\text{s}$  (increased YF1*-catalyzed FixJ phosphorylation rate)
V1star_F = 0.3; %  $\mu\text{M}/\text{s}$  (decreased reverse rate)
V2_F = 0.2; %  $\mu\text{M}/\text{s}$  (decreased basal phosphatase activity)
V2star_F = 0.1; %  $\mu\text{M}/\text{s}$ 
V1_Y = 1.2; %  $\mu\text{M}/\text{s}$ 
V1star_Y = 0.9; %  $\mu\text{M}/\text{s}$ 
V2_Y = 0.7; %  $\mu\text{M}/\text{s}$ 
V2star_Y = 0.5; %  $\mu\text{M}/\text{s}$ 

```

```

k_YF1 = 0.05; % s^-1 (decreased YF1 recovery rate to enhance blue light effect)

% Hill coefficients
n_B = 2;
n_Fp_cI = 1.5;
n_cI_Z = 2.5;

%% Function 1: Multi-component Concentration Analysis (including blue light switch experiment) -
Simplified Version
fprintf('=== Function 1: Multi-component Concentration Analysis (Blue Light Switch Experiment)
===\n');

% Time range - simulating blue light switch experiment
tspan1 = [0 500]; % Blue light off phase
tspan2 = [500 1000]; % Blue light on phase

% Initial condition vector (steady state with blue light off)
y0 = [YF1_star_0, YF1_0, F_0, Fp_0, cI_0, Z_0, Y_0, Yp_0];

% Stage 1: Blue light off (IB = 0)
IB_stage1 = 0;
[t1, y1] = ode45(@(t,y) pd_ode_system(t, y, IB_stage1, IB_50, Vmax_B, k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cI, kdF, n_Fp_cI, beta_cI, Vmax_cI_Z, kdcI, n_cI_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star), tspan1, y0);

% Stage 2: Blue light on (IB = 100), using end state of stage 1 as initial condition
IB_stage2 = 100;
[t2, y2] = ode45(@(t,y) pd_ode_system(t, y, IB_stage2, IB_50, Vmax_B, k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cI, kdF, n_Fp_cI, beta_cI, Vmax_cI_Z, kdcI, n_cI_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star), tspan2, y1(end,:));

% Combine results
t = [t1; t2];
y = [y1; y2];

% Extract component concentrations
YF1_star = y(:,1);
YF1 = y(:,2);
F = y(:,3);
Fp = y(:,4);
cI = y(:,5);

```

```

Z = y(:,6);
Y = y(:,7);
Yp = y(:,8);

% Calculate flux ratio as system state indicator
flux_ratio = Yp ./ (Y + Yp + eps); % Avoid division by zero

% Calculate baseline CheY-P concentration (for Function 3)
Yp_base = y2(end, 8); % CheY-P concentration at end of stage 2

% Plot Function 1: Simplified graphical display
fig1 = figure('Position', [100, 100, 1200, 800]);

% Subplot 1: YF1*/YF1 kinetics
ax1 = subplot(2,3,1);
plot(t, YF1_star, 'r-', 'LineWidth', 3);
hold on;
plot(t, YF1, 'b-', 'LineWidth', 3);
plot([500, 500], [0, max([YF1_star; YF1])], 'k--', 'LineWidth', 2, 'Color', [0.5 0.5 0.5]);
set(get(ax1, 'Title'), 'String', 'Kinase YF1*/YF1 Kinetics');
set(get(ax1, 'XLabel'), 'String', 'Time (s)');
set(get(ax1, 'YLabel'), 'String', 'Concentration (μM)');
legend('YF1* (Active)', 'YF1 (Inactive)', 'Blue Light On', 'Location', 'best');
set(ax1, 'XGrid', 'on', 'YGrid', 'on');
xlim([0 1000]);
text(250, max([YF1_star; YF1])*0.85, 'Blue Light Off', 'HorizontalAlignment', 'center', ...
'FontSize', 11, 'FontWeight', 'bold', 'BackgroundColor', 'white');
text(750, max([YF1_star; YF1])*0.85, 'Blue Light On', 'HorizontalAlignment', 'center', ...
'FontSize', 11, 'FontWeight', 'bold', 'BackgroundColor', 'white');

% Subplot 2: FixJ phosphorylation state
ax2 = subplot(2,3,2);
plot(t, F, 'g-', 'LineWidth', 3);
hold on;
plot(t, Fp, 'm-', 'LineWidth', 3);
plot([500, 500], [0, max([F; Fp])], 'k--', 'LineWidth', 2, 'Color', [0.5 0.5 0.5]);
set(get(ax2, 'Title'), 'String', 'FixJ Phosphorylation State');
set(get(ax2, 'XLabel'), 'String', 'Time (s)');
set(get(ax2, 'YLabel'), 'String', 'Concentration (μM)');
legend('FixJ', 'FixJ-P', 'Blue Light On', 'Location', 'best');
set(ax2, 'XGrid', 'on', 'YGrid', 'on');
xlim([0 1000]);

% Subplot 3: cI and CheZ concentrations

```

```

ax3 = subplot(2,3,3);
plot(t, cI, 'Color', [0.9 0.5 0.1], 'LineWidth', 3);
hold on;
plot(t, Z, 'Color', [0.1 0.7 0.3], 'LineWidth', 3);
plot([500, 500], [0, max([cI; Z])], 'k--', 'LineWidth', 2, 'Color', [0.5 0.5 0.5]);
set(get(ax3, 'Title'), 'String', 'cI and CheZ Concentrations');
set(get(ax3, 'XLabel'), 'String', 'Time (s)');
set(get(ax3, 'YLabel'), 'String', 'Concentration (μM)');
legend('cI Protein', 'CheZ', 'Blue Light On', 'Location', 'best');
set(ax3, 'XGrid', 'on', 'YGrid', 'on');
xlim([0 1000]);

```

% Subplot 4: CheY phosphorylation state

```

ax4 = subplot(2,3,4);
plot(t, Y, 'c-', 'LineWidth', 3);
hold on;
plot(t, Yp, 'Color', [0.5 0 0.8], 'LineWidth', 3);
plot([500, 500], [0, max([Y; Yp])], 'k--', 'LineWidth', 2, 'Color', [0.5 0.5 0.5]);
set(get(ax4, 'Title'), 'String', 'CheY Phosphorylation State');
set(get(ax4, 'XLabel'), 'String', 'Time (s)');
set(get(ax4, 'YLabel'), 'String', 'Concentration (μM)');
legend('CheY', 'CheY-P', 'Blue Light On', 'Location', 'best');
set(ax4, 'XGrid', 'on', 'YGrid', 'on');
xlim([0 1000]);

```

% Subplot 5: CheY-P ratio

```

ax5 = subplot(2,3,5);
plot(t, flux_ratio, 'k-', 'LineWidth', 3);
hold on;
plot([500, 500], [0, max(flux_ratio)], 'k--', 'LineWidth', 2, 'Color', [0.5 0.5 0.5]);
set(get(ax5, 'Title'), 'String', 'CheY-P Ratio (Motility State Indicator)');
set(get(ax5, 'XLabel'), 'String', 'Time (s)');
set(get(ax5, 'YLabel'), 'String', 'CheY-P/(CheY+CheY-P)');
set(ax5, 'XGrid', 'on', 'YGrid', 'on');
xlim([0 1000]);

```

% Subplot 6: Key component comparison (normalized)

```

ax6 = subplot(2,3,6);
% Normalize to 0-1 range
YF1_star_norm = (YF1_star - min(YF1_star)) / (max(YF1_star) - min(YF1_star) + eps);
Fp_norm = (Fp - min(Fp)) / (max(Fp) - min(Fp) + eps);
cI_norm = (cI - min(cI)) / (max(cI) - min(cI) + eps);
Z_norm = (Z - min(Z)) / (max(Z) - min(Z) + eps);
Yp_norm = (Yp - min(Yp)) / (max(Yp) - min(Yp) + eps);

```

```

plot(t, YF1_star_norm, 'r-', 'LineWidth', 2);
hold on;
plot(t, Fp_norm, 'm-', 'LineWidth', 2);
plot(t, cI_norm, '-', 'LineWidth', 2, 'Color', [0.9 0.5 0.1]);
plot(t, Z_norm, '-', 'LineWidth', 2, 'Color', [0.1 0.7 0.3]);
plot(t, Yp_norm, '-', 'LineWidth', 2, 'Color', [0.5 0 0.8]);
plot([500, 500], [0, 1], 'k--', 'LineWidth', 2, 'Color', [0.5 0.5 0.5]);
set(get(ax6, 'Title'), 'String', 'Key Component Comparison (Normalized)');
set(get(ax6, 'XLabel'), 'String', 'Time (s)');
set(get(ax6, 'YLabel'), 'String', 'Normalized Concentration');
legend('YF1*', 'FixJ-P', 'cI', 'CheZ', 'CheY-P', 'Blue Light On', 'Location', 'best');
set(ax6, 'XGrid', 'on', 'YGrid', 'on');
xlim([0 1000]);

```

%% Function 2: Light Intensity-Response Analysis (YF1 System) - Optimized Version

```
fprintf('=== Function 2: Light Intensity-Response Analysis (YF1 System) ===\n');
```

% Light intensity range

```
IB_range = 0:5:200; % Light intensity range
```

```
steady_state_YF1_star = zeros(size(IB_range));
```

```
steady_state_YF1 = zeros(size(IB_range));
```

```
settling_time_YF1_star = zeros(size(IB_range));
```

```
settling_time_YF1 = zeros(size(IB_range));
```

% Set steady state determination threshold

```
settling_threshold = 0.01;
```

% Simplified initial conditions

```
y0_YF1 = [YF1_star_0, YF1_0];
```

% Use precise ODE solver

```
options = odeset('RelTol', 1e-6, 'AbsTol', 1e-8);
```

```
for i = 1:length(IB_range)
```

```
IB_current = IB_range(i);
```

% Use simplified YF1 ODE system

```
[t_temp, y_temp] = ode45(@(t,y) simple_YF1_ode(t, y, IB_current, IB_50, Vmax_B, k_YF1, n_B), ...
```

```
[0 1000], y0_YF1, options);
```

% Take final values as steady state

```
steady_state_YF1_star(i) = y_temp(end, 1);
```

```
steady_state_YF1(i) = y_temp(end, 2);
```

% Calculate settling time

```
YF1_star_final = y_temp(end, 1);
```

```

YF1_final = y_temp(end, 2);
% Find YF1* settling time
for j = 1:length(t_temp)
if abs(y_temp(j, 1) - YF1_star_final) / (YF1_star_final + eps) < settling_threshold
settling_time_YF1_star(i) = t_temp(j);
break;
end
end
% Find YF1 settling time
for j = 1:length(t_temp)
if abs(y_temp(j, 2) - YF1_final) / (YF1_final + eps) < settling_threshold
settling_time_YF1(i) = t_temp(j);
break;
end
end
end

% Plot Function 2: Light intensity response analysis
fig2 = figure('Position', [100, 100, 1200, 500]);

% Figure 1: Effect of light intensity on YF1*/YF1 steady state concentrations
ax1 = subplot(1,2,1);
% Use plotyy to create dual y-axis plot (MATLAB 2015b compatible)
[ax_dual, h1, h2] = plotyy(IB_range, steady_state_YF1_star, IB_range, steady_state_YF1);

% Set line properties
set(h1, 'LineStyle', '-', 'Color', 'r', 'LineWidth', 3, 'Marker', 'o', 'MarkerSize', 4);
set(h2, 'LineStyle', '-', 'Color', 'b', 'LineWidth', 3, 'Marker', 's', 'MarkerSize', 4);

% Set axis properties
set(ax_dual(1), 'YColor', 'r', 'YLim', [0, max(steady_state_YF1_star)*1.1]);
set(ax_dual(2), 'YColor', 'b', 'YLim', [0, max(steady_state_YF1)*1.1]);

set(get(ax_dual(1), 'YLabel'), 'String', 'YF1* Concentration (μM)', 'Color', 'r', 'FontSize', 12);
set(get(ax_dual(2), 'YLabel'), 'String', 'YF1 Concentration (μM)', 'Color', 'b', 'FontSize', 12);
set(get(ax_dual(1), 'XLabel'), 'String', 'Blue Light Intensity (lx)', 'FontSize', 12);
set(get(ax_dual(2), 'XLabel'), 'String', 'Blue Light Intensity (lx)', 'FontSize', 12);

set(get(ax1, 'Title'), 'String', 'Effect of Light Intensity on YF1*/YF1 Steady State');
set(ax1, 'XGrid', 'on', 'YGrid', 'on');

% Add 50% activity point annotation
activity_ratio = steady_state_YF1_star ./ (steady_state_YF1_star + steady_state_YF1 + eps);
[~, idx_50] = min(abs(activity_ratio - 0.5));

```

```

% Correction: Plot text on correct axis
axes(ax_dual(1)); % Switch to first axis
hold on;
plot(IB_range(idx_50), steady_state_YF1_star(idx_50), 'ko', 'MarkerSize', 8, 'LineWidth', 2);

% Corrected text function call - using numerical parameters directly
text(IB_range(idx_50), steady_state_YF1_star(idx_50)*0.8, ...
sprintf('I_{B,50} = %d lx', round(IB_range(idx_50))), ...
'HorizontalAlignment', 'center', 'FontSize', 10, 'FontWeight', 'bold', 'Color', 'k');

legend([h1, h2], {'YF1* (Active)', 'YF1 (Inactive)'}, 'Location', 'best');

% Figure 2: Effect of light intensity on settling time
ax2 = subplot(1,2,2);
plot(IB_range, settling_time_YF1_star, 'r-o', 'LineWidth', 3, 'MarkerSize', 4);
hold on;
plot(IB_range, settling_time_YF1, 'b-s', 'LineWidth', 3, 'MarkerSize', 4);
set(get(ax2, 'XLabel'), 'String', 'Blue Light Intensity (lx)', 'FontSize', 12);
set(get(ax2, 'YLabel'), 'String', 'Settling Time (s)', 'FontSize', 12);
set(get(ax2, 'Title'), 'String', 'Effect of Light Intensity on Settling Time');
legend('YF1*', 'YF1', 'Location', 'best');
set(ax2, 'XGrid', 'on', 'YGrid', 'on');

% Annotate fastest and slowest response times
[min_time, min_idx] = min(settling_time_YF1_star);
[max_time, max_idx] = max(settling_time_YF1_star);
text(IB_range(min_idx), min_time*0.9, sprintf('Fastest: %.1fs', min_time), ...
'HorizontalAlignment', 'center', 'FontSize', 10, 'Color', 'r', 'FontWeight', 'bold');
text(IB_range(max_idx), max_time*1.05, sprintf('Slowest: %.1fs', max_time), ...
'HorizontalAlignment', 'center', 'FontSize', 10, 'Color', 'r', 'FontWeight', 'bold');

%% Function 3: Complete Parameter Sensitivity Analysis (positive and negative perturbations)
fprintf('\n=== Function 3: Complete Parameter Sensitivity Analysis ===\n');

% Key parameters for analysis
param_names = {'Vmax_B', 'IB_50', 'k_YF1', 'V1_F', 'V2_Y', 'Vmax_Fp_cI', 'Vmax_cI_Z'};
param_values = [Vmax_B, IB_50, k_YF1, V1_F, V2_Y, Vmax_Fp_cI, Vmax_cI_Z];
param_descriptions = {'Max blue light response rate', 'Half-max response intensity', 'YF1 recovery
rate', ...
'Max FixJ phosphorylation rate', 'Max CheY dephosphorylation rate', ...
'Max cI expression rate', 'Max CheZ expression rate'};

fprintf('Baseline CheY-P steady state concentration: %.4f μM\n', Yp_base);

```

```

% Sensitivity analysis - ±10% perturbation
perturbation = 0.10; % ±10% perturbation
sensitivity_results = zeros(length(param_names), 3); % Store results: [negative, baseline, positive]

fprintf('\nParameter Sensitivity Analysis (±%.0f%% perturbation):\n', perturbation*100);
fprintf('%-15s %-20s %-12s %-12s %-12s %-10s\n', ...
'Parameter', 'Description', '-10% Conc', 'Baseline Conc', '+10% Conc', 'Sensitivity');

for i = 1:length(param_names)
% Calculate perturbed parameter values
param_neg = param_values(i) * (1 - perturbation);
param_pos = param_values(i) * (1 + perturbation);
% Store baseline value
sensitivity_results(i, 2) = Yp_base;
% Test negative perturbation
[~, y_neg] = simulate_parameter_effect(param_names{i}, param_neg, IB_stage2, IB_50, Vmax_B,
k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cl, kdF, n_Fp_cl, beta_cl, Vmax_cl_Z, kdcI, n_cl_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star, tspan2, y1(end,:));
sensitivity_results(i, 1) = y_neg(end, 8);
% Test positive perturbation
[~, y_pos] = simulate_parameter_effect(param_names{i}, param_pos, IB_stage2, IB_50, Vmax_B,
k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cl, kdF, n_Fp_cl, beta_cl, Vmax_cl_Z, kdcI, n_cl_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star, tspan2, y1(end,:));
sensitivity_results(i, 3) = y_pos(end, 8);
% Calculate sensitivity metric (average change rate)
sensitivity_neg = (sensitivity_results(i, 1) - Yp_base) / Yp_base / perturbation * 100;
sensitivity_pos = (sensitivity_results(i, 3) - Yp_base) / Yp_base / perturbation * 100;
avg_sensitivity = (abs(sensitivity_neg) + abs(sensitivity_pos)) / 2;
fprintf('%-15s %-20s %-12.4f %-12.4f %-12.4f %-10.2f\n', ...
param_names{i}, param_descriptions{i}, ...
sensitivity_results(i, 1), sensitivity_results(i, 2), sensitivity_results(i, 3), avg_sensitivity);
end

% Plot sensitivity analysis visualization
fig3 = figure('Position', [100, 100, 1400, 800]);

% Subplot 1: Effect of parameter perturbation on CheY-P concentration (bar chart)
ax1 = subplot(2,2,1);
% Calculate percentage changes

```

```

changes_neg = (sensitivity_results(:,1) - sensitivity_results(:,2)) ./ sensitivity_results(:,2) * 100;
changes_pos = (sensitivity_results(:,3) - sensitivity_results(:,2)) ./ sensitivity_results(:,2) * 100;

```

```

h = bar([changes_neg, changes_pos], 'grouped');
set(h(1), 'FaceColor', [0.8 0.2 0.2], 'EdgeColor', 'k');
set(h(2), 'FaceColor', [0.2 0.6 0.2], 'EdgeColor', 'k');

```

```

set(get(ax1, 'XLabel'), 'String', 'Parameter', 'FontSize', 12, 'FontWeight', 'bold');
set(get(ax1, 'YLabel'), 'String', 'CheY-P Concentration Change (%)', 'FontSize', 12, 'FontWeight', 'bold');
set(get(ax1, 'Title'), 'String', 'Effect of Parameter Perturbation on CheY-P ( $\pm 10\%$  perturbation));
set(gca, 'XTickLabel', param_names, 'XTickLabelRotation', 45);
legend('-10% Perturbation', '+10% Perturbation', 'Location', 'best');
set(ax1, 'XGrid', 'on', 'YGrid', 'on');

```

```

% Add numerical labels

```

```

for i = 1:length(param_names)
    if changes_neg(i) < 0
        text(i-0.2, changes_neg(i)-1.5, sprintf('%1f%%', changes_neg(i)), ...
            'HorizontalAlignment', 'center', 'FontSize', 9, 'FontWeight', 'bold', 'Color', 'r');
    else
        text(i-0.2, changes_neg(i)+1.5, sprintf('%1f%%', changes_neg(i)), ...
            'HorizontalAlignment', 'center', 'FontSize', 9, 'FontWeight', 'bold', 'Color', 'r');
    end
    if changes_pos(i) < 0
        text(i+0.2, changes_pos(i)-1.5, sprintf('%1f%%', changes_pos(i)), ...
            'HorizontalAlignment', 'center', 'FontSize', 9, 'FontWeight', 'bold', 'Color', 'g');
    else
        text(i+0.2, changes_pos(i)+1.5, sprintf('%1f%%', changes_pos(i)), ...
            'HorizontalAlignment', 'center', 'FontSize', 9, 'FontWeight', 'bold', 'Color', 'g');
    end
end
end

```

```

% Subplot 2: Sensitivity ranking (alternative to radar chart, using bar chart)

```

```

ax2 = subplot(2,2,2);
% Calculate absolute sensitivity index
sensitivity_index = (abs(changes_neg) + abs(changes_pos)) / 2;

```

```

% Sort by sensitivity

```

```

[sorted_sensitivity, sort_idx] = sort(sensitivity_index, 'descend');
sorted_names = param_names(sort_idx);

```

```

barh(sorted_sensitivity, 'FaceColor', [0.3 0.5 0.9], 'EdgeColor', 'k');
set(gca, 'YTickLabel', sorted_names, 'YTick', 1:length(sorted_names));

```

```

set(get(ax2, 'XLabel'), 'String', 'Sensitivity Index', 'FontSize', 12, 'FontWeight', 'bold');
set(get(ax2, 'YLabel'), 'String', 'Parameter', 'FontSize', 12, 'FontWeight', 'bold');
set(get(ax2, 'Title'), 'String', 'Parameter Sensitivity Ranking');
set(ax2, 'XGrid', 'on', 'YGrid', 'on');

% Add numerical labels
for i = 1:length(sort_idx)
text(sorted_sensitivity(i) + max(sorted_sensitivity)*0.02, i, ...
sprintf('%.2f', sorted_sensitivity(i)), ...
'FontSize', 10, 'FontWeight', 'bold');
end

% Subplot 3: Time dynamics comparison of key parameters
ax3 = subplot(2,2,3);
% Select top 3 most sensitive parameters for comparison
top_params = sort_idx(1:min(3, length(param_names)));

% Simulate time course of key parameter perturbations
t_compare = t2 - 500; % Time relative to blue light on
plot(t_compare, y2(:,8), 'k-', 'LineWidth', 4, 'DisplayName', 'Baseline');
hold on;

colors = ['r', 'g', 'b'];
line_styles = {'--', '-.', ':'};
for j = 1:length(top_params)
param_idx = top_params(j);
param_pos = param_values(param_idx) * 1.1;
[t_temp, y_temp] = simulate_parameter_effect(param_names{param_idx}, param_pos, IB_stage2,
IB_50, Vmax_B, k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cI, kdF, n_Fp_cI, beta_cI, Vmax_cI_Z, kdI, n_cI_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star, tspan2, y1(end,:));
plot(t_temp-500, y_temp(:,8), line_styles{j}, 'LineWidth', 2.5, 'Color', colors(j), ...
'DisplayName', sprintf('%s (+10%%)', param_names{param_idx}));
end

set(get(ax3, 'XLabel'), 'String', 'Time After Blue Light On (s)', 'FontSize', 12, 'FontWeight', 'bold');
set(get(ax3, 'YLabel'), 'String', 'CheY-P Concentration ( $\mu$ M)', 'FontSize', 12, 'FontWeight', 'bold');
set(get(ax3, 'Title'), 'String', 'Effect of Key Parameter Perturbation on CheY-P Dynamics');
legend('show', 'Location', 'best');
set(ax3, 'XGrid', 'on', 'YGrid', 'on');
xlim([0 500]);

```

```

% Subplot 4: Parameter perturbation comparison (scatter plot)

```

```

ax4 = subplot(2,2,4);
% Plot relationship between baseline and perturbed values
param_indices = 1:length(param_names);
plot(param_indices, sensitivity_results(:,2), 'ko-', 'LineWidth', 3, ...
'MarkerSize', 8, 'MarkerFaceColor', 'k', 'DisplayName', 'Baseline');
hold on;
plot(param_indices, sensitivity_results(:,1), 'rv', 'MarkerSize', 8, ...
'MarkerFaceColor', 'r', 'DisplayName', '-10% Perturbation');
plot(param_indices, sensitivity_results(:,3), 'g^', 'MarkerSize', 8, ...
'MarkerFaceColor', 'g', 'DisplayName', '+10% Perturbation');

set(get(ax4, 'XLabel'), 'String', 'Parameter Index', 'FontSize', 12, 'FontWeight', 'bold');
set(get(ax4, 'YLabel'), 'String', 'CheY-P Concentration ( $\mu$ M)', 'FontSize', 12, 'FontWeight', 'bold');
set(get(ax4, 'Title'), 'String', 'Parameter Perturbation Comparison for CheY-P');
legend('show', 'Location', 'best');
set(ax4, 'XGrid', 'on', 'YGrid', 'on');

% Add parameter name labels
for i = 1:length(param_names)
text(i, sensitivity_results(i,2) + 0.05, param_names{i}, ...
'Rotation', 45, 'FontSize', 9, 'HorizontalAlignment', 'left');
end

%% Analysis of Light Intensity Effect on YF1 System Settling Time
fprintf('\n=== Analysis of Light Intensity Effect on YF1 System Settling Time ===\n');

% Select key light intensity points for analysis
key_IB = [0, 25, 50, 75, 100, 125, 150, 175, 200];
settling_times_YF1_star = zeros(size(key_IB));
settling_times_YF1 = zeros(size(key_IB));
steady_state_YF1_star = zeros(size(key_IB));
steady_state_YF1 = zeros(size(key_IB));

fprintf('Intensity(lx) | YF1* Settling(s) | YF1 Settling(s) | Steady YF1*( $\mu$ M) | Steady YF1( $\mu$ M)\n');
fprintf('-----|-----|-----|-----|-----\n');

for i = 1:length(key_IB)
IB_current = key_IB(i);
% Use simplified YF1 ODE system
[t_temp, y_temp] = ode45(@(t,y) simple_YF1_ode(t, y, IB_current, IB_50, Vmax_B, k_YF1, n_B), ...
[0 1000], y0_YF1, options);
% Calculate settling time
YF1_star_final = y_temp(end, 1);
YF1_final = y_temp(end, 2);

```

```

% YF1* settling time
t_settle_YF1_star = 1000; % Default maximum
for j = 1:length(t_temp)
if abs(y_temp(j, 1) - YF1_star_final) / (YF1_star_final + eps) < settling_threshold
t_settle_YF1_star = t_temp(j);
break;
end
end

% YF1 settling time
t_settle_YF1 = 1000; % Default maximum
for j = 1:length(t_temp)
if abs(y_temp(j, 2) - YF1_final) / (YF1_final + eps) < settling_threshold
t_settle_YF1 = t_temp(j);
break;
end
end

settling_times_YF1_star(i) = t_settle_YF1_star;
settling_times_YF1(i) = t_settle_YF1;
steady_state_YF1_star(i) = YF1_star_final;
steady_state_YF1(i) = YF1_final;
fprintf('%12d | %16.1f | %15.1f | %16.3f | %14.3fn', ...
key_IB(i), t_settle_YF1_star, t_settle_YF1, YF1_star_final, YF1_final);
end

% Plot effect of light intensity on settling time
fig4 = figure('Position', [100, 100, 1000, 600]);

subplot(1,2,1);
plot(key_IB, settling_times_YF1_star, 'r-o', 'LineWidth', 3, 'MarkerSize', 6, 'MarkerFaceColor', 'r');
hold on;
plot(key_IB, settling_times_YF1, 'b-s', 'LineWidth', 3, 'MarkerSize', 6, 'MarkerFaceColor', 'b');
set(gca, 'XLabel', 'String', 'Blue Light Intensity (lx)', 'FontSize', 12, 'FontWeight', 'bold');
set(gca, 'YLabel', 'String', 'Settling Time (s)', 'FontSize', 12, 'FontWeight', 'bold');
set(gca, 'Title', 'String', 'Effect of Light Intensity on YF1 System Settling Time');
legend('YF1* Settling Time', 'YF1 Settling Time', 'Location', 'best');
set(gca, 'XGrid', 'on', 'YGrid', 'on');

% Annotate fastest response intensity
[min_time_star, min_idx_star] = min(settling_times_YF1_star);
[min_time, min_idx] = min(settling_times_YF1);
text(key_IB(min_idx_star), min_time_star*1.1, sprintf('Fastest: %.1fs @ %dlx', min_time_star,
key_IB(min_idx_star)), ...
'HorizontalAlignment', 'center', 'FontSize', 10, 'Color', 'r', 'FontWeight', 'bold');

```

```

subplot(1,2,2);
% Plot steady state concentration vs light intensity
[ax_final, h1_final, h2_final] = plotyy(key_IB, steady_state_YF1_star, key_IB, steady_state_YF1);
set(h1_final, 'LineStyle', '-', 'Color', 'r', 'LineWidth', 3, 'Marker', 'o', 'MarkerSize', 6);
set(h2_final, 'LineStyle', '-', 'Color', 'b', 'LineWidth', 3, 'Marker', 's', 'MarkerSize', 6);
set(ax_final(1), 'YColor', 'r');
set(ax_final(2), 'YColor', 'b');
set(get(ax_final(1), 'XLabel'), 'String', 'Blue Light Intensity (lx)', 'FontSize', 12, 'FontWeight', 'bold');
set(get(ax_final(1), 'YLabel'), 'String', 'YF1* Steady State ( $\mu$ M)', 'Color', 'r', 'FontSize', 12);
set(get(ax_final(2), 'YLabel'), 'String', 'YF1 Steady State ( $\mu$ M)', 'Color', 'b', 'FontSize', 12);
set(get(gca, 'Title'), 'String', 'Effect of Light Intensity on YF1 System Steady State');
set(gca, 'XGrid', 'on', 'YGrid', 'on');

fprintf('\n=== All Simulations Completed ===\n');

```

```

% Define the ODE system function
function dydt = pd_ode_system(t, y, IB, IB_50, Vmax_B, k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cl, kdF, n_Fp_cl, beta_cl, Vmax_cl_Z, kdcI, n_cl_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star)
% Extract state variables
YF1_star = y(1); % Active YF1
YF1 = y(2); % Inactive YF1
F = y(3); % FixJ
Fp = y(4); % Phosphorylated FixJ
cl = y(5); % cI protein
Z = y(6); % CheZ
Y = y(7); % CheY
Yp = y(8); % Phosphorylated CheY
% Blue light response kinetics
blue_response = Vmax_B * (IB^n_B) / (IB^n_B + IB_50^n_B);
% YF1 activation/inactivation kinetics
dYF1_star_dt = blue_response * YF1 - k_YF1 * YF1_star;
dYF1_dt = -blue_response * YF1 + k_YF1 * YF1_star;
% FixJ phosphorylation kinetics
dF_dt = -V1_F * YF1_star * F / (KF1 + F) + ...
V1star_F * Fp / (KFP1_star + Fp) + ...
V2star_F * Fp / (KFP2_star + Fp) - ...
V2_F * F / (KF2 + F);
dFp_dt = V1_F * YF1_star * F / (KF1 + F) - ...
V1star_F * Fp / (KFP1_star + Fp) - ...
V2star_F * Fp / (KFP2_star + Fp) + ...
V2_F * F / (KF2 + F);
% cI expression dynamics

```

```

dcI_dt = Vmax_Fp_cI * (Fp^n_Fp_cI) / (kdF^n_Fp_cI + Fp^n_Fp_cI) - beta_cI * cI;
% CheZ expression dynamics
dZ_dt = Vmax_cI_Z * (cI^n_cI_Z) / (kdcI^n_cI_Z + cI^n_cI_Z) - beta_Z * Z;
% CheY phosphorylation kinetics
dY_dt = -V1_Y * Y / (KY1 + Y) + ...
V1star_Y * Yp / (KYP1_star + Yp) + ...
V2star_Y * Yp / (KYP2_star + Yp) - ...
V2_Y * Z * Y / (KY2 + Y);
dYp_dt = V1_Y * Y / (KY1 + Y) - ...
V1star_Y * Yp / (KYP1_star + Yp) - ...
V2star_Y * Yp / (KYP2_star + Yp) + ...
V2_Y * Z * Y / (KY2 + Y);
% Assemble derivative vector
dydt = [dYF1_star_dt; dYF1_dt; dF_dt; dFp_dt; dcI_dt; dZ_dt; dY_dt; dYp_dt];
end

```

```

% Simplified YF1 ODE system for light response analysis
function dydt = simple_YF1_ode(t, y, IB, IB_50, Vmax_B, k_YF1, n_B)
YF1_star = y(1);
YF1 = y(2);
% Blue light response kinetics
blue_response = Vmax_B * (IB^n_B) / (IB^n_B + IB_50^n_B);
dYF1_star_dt = blue_response * YF1 - k_YF1 * YF1_star;
dYF1_dt = -blue_response * YF1 + k_YF1 * YF1_star;
dydt = [dYF1_star_dt; dYF1_dt];
end

```

```

% Function to simulate parameter effect
function [t_out, y_out] = simulate_parameter_effect(param_name, param_value, IB, IB_50, Vmax_B,
k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cI, kdF, n_Fp_cI, beta_cI, Vmax_cI_Z, kdcI, n_cI_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star, tspan, y0)
% Update the specified parameter
switch param_name
case 'Vmax_B'
Vmax_B = param_value;
case 'IB_50'
IB_50 = param_value;
case 'k_YF1'
k_YF1 = param_value;
case 'V1_F'
V1_F = param_value;
case 'V2_Y'

```

```

V2_Y = param_value;
case 'Vmax_Fp_cI'
Vmax_Fp_cI = param_value;
case 'Vmax_cI_Z'
Vmax_cI_Z = param_value;
end
% Simulate with updated parameter
[t_out, y_out] = ode45(@(t,y) pd_ode_system(t, y, IB, IB_50, Vmax_B, k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cI, kdF, n_Fp_cI, beta_cI, Vmax_cI_Z, kdcI, n_cI_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star), tspan, y0);
end

%% Define the ODE system function
function dydt = pd_ode_system(t, y, IB, IB_50, Vmax_B, k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cI, kdF, n_Fp_cI, beta_cI, Vmax_cI_Z, kdcI, n_cI_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star)
% Extracted variables
YF1_star = y(1); % Active YF1*
YF1 = y(2); % non-active YF1
F = y(3); % FixJ
Fp = y(4); % FixJ-P
cI = y(5); % cI protein
Z = y(6); % CheZ
Y = y(7); % CheY
Yp = y(8); % CheY-P
% Blue light response item (Hill function) -Correction: Use element-level power operation. ^
light_response = Vmax_B * (IB.^n_B ./ (IB.^n_B + IB_50.^n_B));
% YF1 system dynamics
dYF1_star = -light_response * YF1_star + k_YF1 * YF1;
dYF1 = light_response * YF1_star - k_YF1 * YF1;
% FixJ phosphorylation/dephosphorylation
v_phospho_F = V1_F * YF1_star * F / (KF1 + F) - V1star_F * YF1 * Fp / (KFP1_star + Fp);
v_dephospho_F = V2_F * Fp / (KF2 + Fp) - V2star_F * F / (KFP2_star + F);
dF = -v_phospho_F + v_dephospho_F;
dFp = v_phospho_F - v_dephospho_F;
% cI protein expression (regulated by Fp) -Revised: Using element-level exponentiation. ^
dcI = Vmax_Fp_cI * (Fp.^n_Fp_cI ./ (Fp.^n_Fp_cI + kdF.^n_Fp_cI)) - beta_cI * cI;
% Chez expression (suppressed by cI) -Revised: Using element-level exponentiation. ^
dZ = beta_Z + Vmax_cI_Z * (1 - cI.^n_cI_Z ./ (cI.^n_cI_Z + kdcI.^n_cI_Z)) - beta_Z * Z;
% CheY phosphorylation/dephosphorylation
v_phospho_Y = V1_Y * YF1_star * Y / (KY1 + Y) - V1star_Y * YF1 * Yp / (KYP1_star + Yp);
v_dephospho_Y = V2_Y * Z * Yp / (KY2 + Yp) - V2star_Y * Y / (KYP2_star + Y);

```

```

dY = -v_phospho_Y + v_dephospho_Y;
dYp = v_phospho_Y - v_dephospho_Y;
% Return the derivative vector
dydt = [dYF1_star; dYF1; dF; dFp; dcl; dZ; dY; dYp];
end

function dydt = simple_YF1_ode(t, y, IB, IB_50, Vmax_B, k_YF1, n_B)
YF1_star = y(1);
YF1 = y(2);
% Ensure that all parameters are scalar
IB = IB(1); IB_50 = IB_50(1); Vmax_B = Vmax_B(1);
k_YF1 = k_YF1(1); n_B = n_B(1);
% Blue light response item-Use element level power operation
light_response = Vmax_B * (IB.^n_B ./ (IB.^n_B + IB_50.^n_B));
% YF1 system dynamics
dYF1_star = -light_response * YF1_star + k_YF1 * YF1;
dYF1 = light_response * YF1_star - k_YF1 * YF1;
dydt = [dYF1_star; dYF1];
end

%% Auxiliary function: simulate parameter effects
function [t_out, y_out] = simulate_parameter_effect(param_name, param_value, IB, IB_50, Vmax_B,
k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cl, kdF, n_Fp_cl, beta_cl, Vmax_cl_Z, kdcI, n_cl_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star, tspan, y0)

switch param_name
case 'Vmax_B'
[t_out, y_out] = ode45(@(t,y) pd_ode_system(t, y, IB, IB_50, param_value, k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cl, kdF, n_Fp_cl, beta_cl, Vmax_cl_Z, kdcI, n_cl_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star), tspan, y0);
case 'IB_50'
[t_out, y_out] = ode45(@(t,y) pd_ode_system(t, y, IB, param_value, Vmax_B, k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cl, kdF, n_Fp_cl, beta_cl, Vmax_cl_Z, kdcI, n_cl_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star), tspan, y0);
case 'k_YF1'
[t_out, y_out] = ode45(@(t,y) pd_ode_system(t, y, IB, IB_50, Vmax_B, param_value, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cl, kdF, n_Fp_cl, beta_cl, Vmax_cl_Z, kdcI, n_cl_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star), tspan, y0);
case 'V1_F'

```

```

[t_out, y_out] = ode45(@(t,y) pd_ode_system(t, y, IB, IB_50, Vmax_B, k_YF1, n_B, ...
param_value, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cI, kdF, n_Fp_cI, beta_cI, Vmax_cI_Z, kdcI, n_cI_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star), tspan, y0);
case 'V2_Y'
[t_out, y_out] = ode45(@(t,y) pd_ode_system(t, y, IB, IB_50, Vmax_B, k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cI, kdF, n_Fp_cI, beta_cI, Vmax_cI_Z, kdcI, n_cI_Z, beta_Z, ...
V1_Y, V1star_Y, param_value, V2star_Y, KY1, KY2, KYP1_star, KYP2_star), tspan, y0);
case 'Vmax_Fp_cI'
[t_out, y_out] = ode45(@(t,y) pd_ode_system(t, y, IB, IB_50, Vmax_B, k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
param_value, kdF, n_Fp_cI, beta_cI, Vmax_cI_Z, kdcI, n_cI_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star), tspan, y0);
case 'Vmax_cI_Z'
[t_out, y_out] = ode45(@(t,y) pd_ode_system(t, y, IB, IB_50, Vmax_B, k_YF1, n_B, ...
V1_F, V1star_F, V2_F, V2star_F, KF1, KF2, KFP1_star, KFP2_star, ...
Vmax_Fp_cI, kdF, n_Fp_cI, beta_cI, param_value, kdcI, n_cI_Z, beta_Z, ...
V1_Y, V1star_Y, V2_Y, V2star_Y, KY1, KY2, KYP1_star, KYP2_star), tspan, y0);
end
end

```