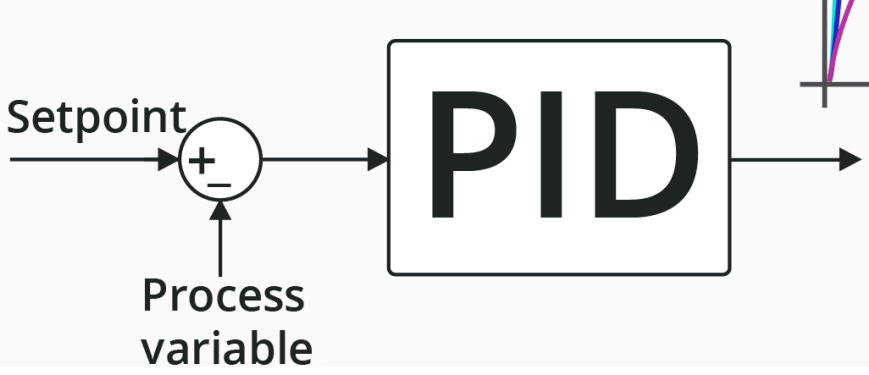
PROCESS CONTROL

REPORT

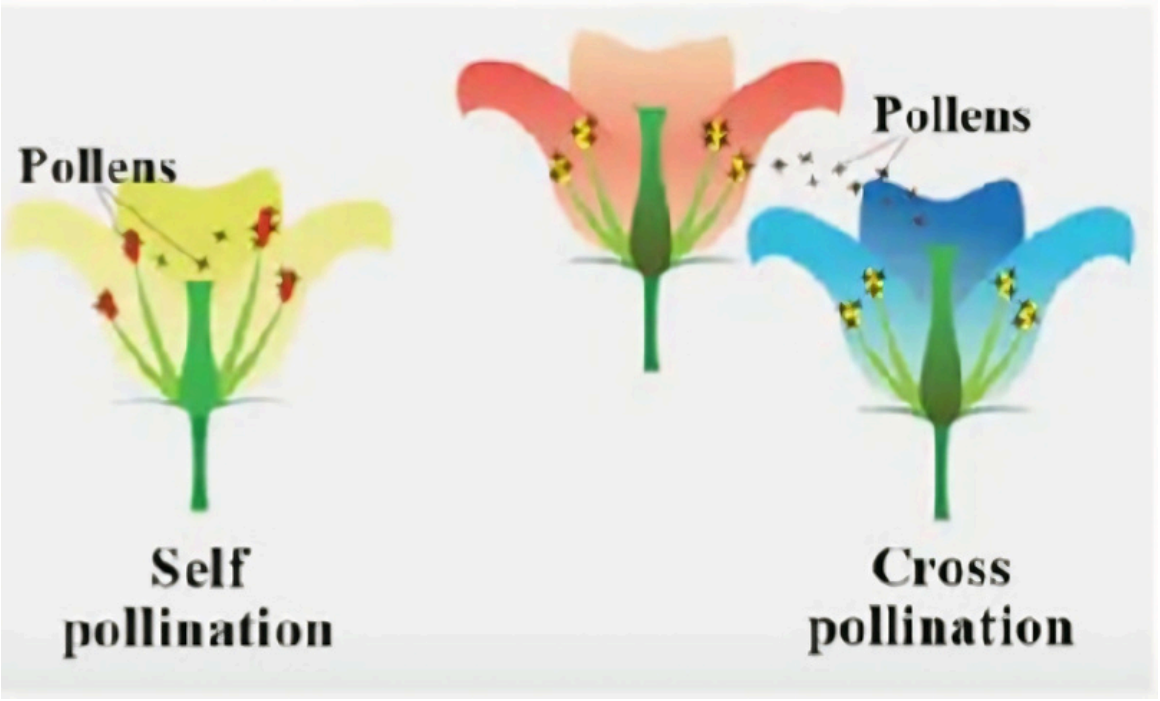
TUNE A PID CONTROLLER

PID
Controller



DR/ Ahmed Osman

Tune a PID Controller by Flower Pollination Algorithm



PREPARED BY: GROUB_16

sec	ID	Name
2	6368	Menna Allah Essam Ahmed Salem
1	6328	Sara Mohammed Abdelmeged

Introduction :

In control systems, tuning a PID (Proportional-Integral-Derivative) controller is crucial for achieving desired performance such as fast response, minimal overshoot, and stable steady-state behavior. This project applies the Flower Pollination Algorithm (FPA), a nature-inspired optimization method, to automatically tune PID parameters (Kp, Ki, Kd) for optimal system performance.

The Flower Pollination Algorithm (FPA) :

FPA is a nature-inspired metaheuristic optimization algorithm introduced by **(Xin-She Yang**) in **2012**. It mimics the natural process of flower pollination where pollen grains are transferred — often over long distances — to improve genetic diversity and reproduction success.

It's part of the swarm intelligence family (like Particle Swarm Optimization, Ant Colony Optimization), meaning it relies on a population of interacting solutions rather than a single search path.

1- Global Pollination (Cross-pollination):

- Pollen from one flower reaches a faraway flower (via pollinators like insects, birds, wind).
- In FPA, this represents long-range search — big steps exploring the search space.

2- Local Pollination (Self-pollination):

- Pollen is transferred within the same plant or nearby plants.
- This represents small refinements around the current solutions — local fine-tuning.

Optimization Problem

What is Optimization?

Optimization is the process of finding the best solution among all feasible solutions to a problem.

Search Space:

The defined lower and upper bounds [LB,UB] for each parameter.

Decision Variables:

Parameters the optimizer adjusts (Kp,Ki,Kd).

Objective Function:

A performance metric we aim to minimize (or maximize)

In Our Code we Using 2 objective Function

1.The Integral of Time-weighted Absolute Error (ITAE):

2.The Integral of Squared Error (ISE)

- ## FPA Procedure:

Step 1: Initialization

1. Randomly generate N initial solutions:

$$Xi = LB + rand \times (UB - LB)$$

2. Evaluate each solution's objective function

3. Identify the global best (Xg)

Step 2: Main Optimization Loop

1. For each generation (iteration):
2. For each solution:
   > Generate random number (rand).
   > **If rand<p** →
   Generate a Step size
   Generate a new solution(global pollination) :

$$Xi(t+1) = Xi(t) + L\cdot(Xi(t)-Xg)$$

> Else →
Generate a new solution(local pollination)

$$Xi(t+1) = Xi(t) + \epsilon(Xj-Xk)$$

where Xj, Xk are randomly selected solutions

4 Update Objective Function(ITAE OR ISE)

5 Perform Greedy selection and update Xg

Step 3: Termination

> The steps of FPA algorithm are iteratively repeated until the maximum number of generations is reached or a termination criterion is met.

> Convergence: is the case where the positions of all flowers converge to the same set of values, the method is assumed to have converged.

Pseudo code

1. Input
   - Objective function (fitness function), population size (N), switch probability (p), number of iteration (T)
2. Initialization
   - Initialize population N flowers
   - Evaluate objective function and assign xg
3. Loop:
   - For t = 1
   - For i = 1
   - If rand < p
     > Generate a step size (σ)
     > Generate a new solution based on global pollination
   - Else
     > Choose two solutions randomly among all solutions
     > Generate a new solution based on local pollination
   - Perform greedy selection and update xg
   - If there is no convergence of the current solution & if t > T go to Loop
4. Print the optimal solution

- ## PID Controller Tuning Using Different Objective Functions

To achieve optimal tuning for the PID controller, it was necessary to select appropriate objective functions. These objective functions serve as performance metrics that guide the optimization process. In this Project, we implemented the Flower Pollination Algorithm (FPA) with two different objective functions in separate implementations:

1. **The Integral of Time-weighted Absolute Error (ITAE):**

$$ITAE = \int_0^\infty t\,|\,e(t)\,|\,dt$$

2. **The Integral of Squared Error (ISE):**

$$ISE = \int_0^\infty e^2(t)dt$$

- **ISE (Integral Square Error)**

> ISE calculates the sum of squared errors, which gives higher weight to large errors regardless of when they occur

>It tends to produce faster response

- **ITAE (Integral Time Absolute Error)**

> ITAE weighs errors by time, which means it places greater emphasis on errors that persist longer in the system

>It produces better damped responses

- ## MATLAB Code Overview:

We implemented this procedure in MATLAB, using:

> Random initialization of populations.

> ITAE and ISE as the fitness (objective) function.

> A loop structure to perform global/local updates.

> Selection mechanisms to track the best PID parameters.

Process

We have been working on a third Order Process

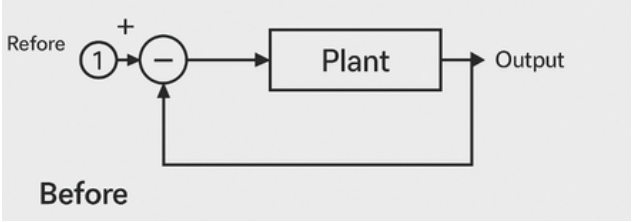
not stable

$$G(s) = \frac{s+2}{s^3+2s^2+3s+5}$$

- **Matlap code(ITAE)**

```
clc; clear all; close all;
% Simulation time
t_sim = 20;
tsim = 0:0.1:t_sim;
u = ones(size(tsim));
ns = [1 2];
ds = [1 2 3 5];
%ns = [1];
%ds = [1 0.6 1];
G = tf(ns, ds);
G_uncontrolled = feedback(G, 1);
y_uncontrolled = lsim(G_uncontrolled, u, tsim);
figure(1);
plot(tsim, y_uncontrolled, "r--", "LineWidth", 1.5);
xlabel("Time (s)");
ylabel("Amplitude");
title("Closed-Loop System Response (No Controller)");
grid on;
% FPA Parameters
N = 100;           % Population size
T = 30;            % Iterations
c = 0.7;           % Step size constant
p = 0.6;           % Probability for global pollination
var = 3;           % PID parameters (Kp, Ki, Kd)
lb = [0, 0, 0];    %minValue
```

Process Before Tuning



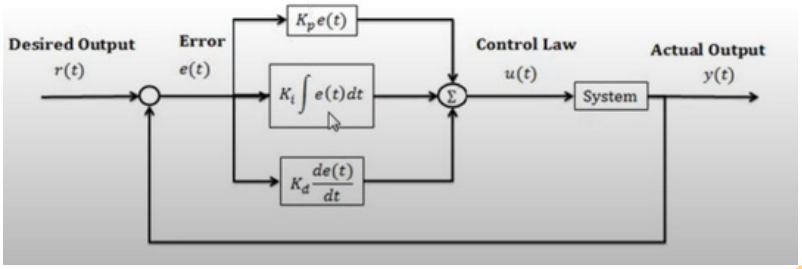

```
ub = [25, 25, 25]; %maxValue
% Desired response value(sp)
r_s = 1;
x = zeros(N, var);
for i = 1:N
    x(i, :) = lb + rand(1, var) .* (ub - lb);
end
% Evaluate initial objective function (ITAE + overshoot penalty)
ffb = zeros(1, N);
for i = 1:N
    [Kp, Ki, Kd] = deal(x(i, 1), x(i, 2), x(i, 3));
    Gc = pid(Kp, Ki, Kd);
    Gcf = feedback(Gc * G, 1);
    y = lsim(Gcf, u, tsim);
    y = y(:);
    overshoot = max(0, max(y) - r_s);
    time_weight = (1:length(y))';
    error = abs(y - r_s);
    itae = sum(error .* time_weight);
    overshoot_penalty = 50 * overshoot^2;
    ffb(i) = itae + overshoot_penalty;
end
% Get global best
[obj_Gbest, Gbest_location] = min(ffb);
xg = x(Gbest_location, :);
```

$$ITAE = \int_0^\infty t\,|e(t)|\,dt$$

```
xg = x(Gbest_location, :);
best_cf_fpa = zeros(1, T);
% Main FPA loop
for t = 1:T
    for i = 1:N
        r = rand;
        if r < p
            %global
            L = levy(var);
            for w = 1:var
                x(i, w) = xg(w) + L(w) * (x(i, w) - xg(w));
                x(i, w) = max(lb(w), min(ub(w), x(i, w)));
            end
        else
            %local
            jj = randi([1, N]);
            while jj == i, jj = randi([1, N]); end
            kk = randi([1, N]);
            while kk == i || kk == jj, kk = randi([1, N]); end
            for w = 1:var
                x(i, w) = x(i, w) + rand * (x(jj, w) - x(kk, w));
                x(i, w) = max(lb(w), min(ub(w), x(i, w)));
            end
        end
```

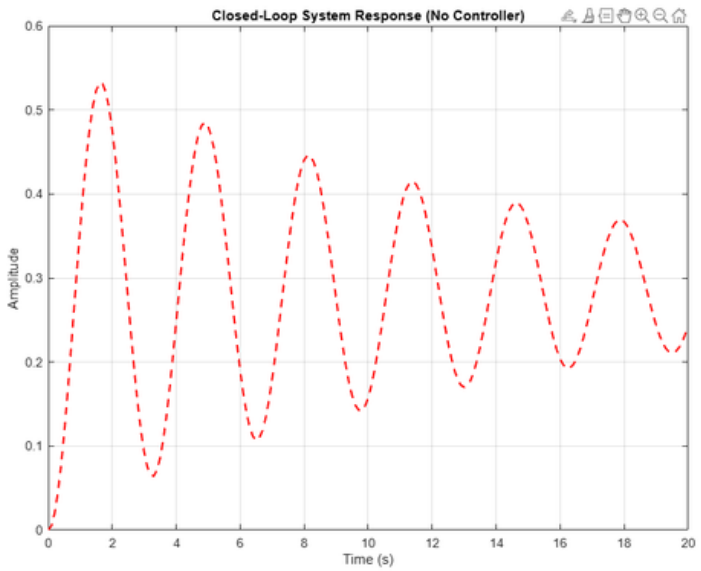
```
          [Kp, Ki, Kd] = deal(x(i, 1), x(i, 2), x(i, 3));
          Gc = pid(Kp, Ki, Kd);
          Gcf = feedback(Gc * G, 1);
          y = lsim(Gcf, u, tsim);
          y = y(:);
          overshoot = max(0, max(y) - r_s);
          error = abs(y - r_s);
          time_weight = (1:length(y))';
          itae = sum(error .* time_weight);
          overshoot_penalty = 50 * overshoot^2;
          ffb(i) = itae + overshoot_penalty;
          if ffb(i) < obj_Gbest
              obj_Gbest = ffb(i);
              xg = x(i, :);
          end
      end
      best_cf_fpa(t) = obj_Gbest;
      % Plot optimization progress
      figure(2);
      plot(1:t, best_cf_fpa(1:t), 'b', 'LineWidth', 2);
      xlabel('Iteration');
      ylabel('Cost Function (Modified ITAE)');
      title('Optimization Progress');
      grid on;
      drawnow;
```

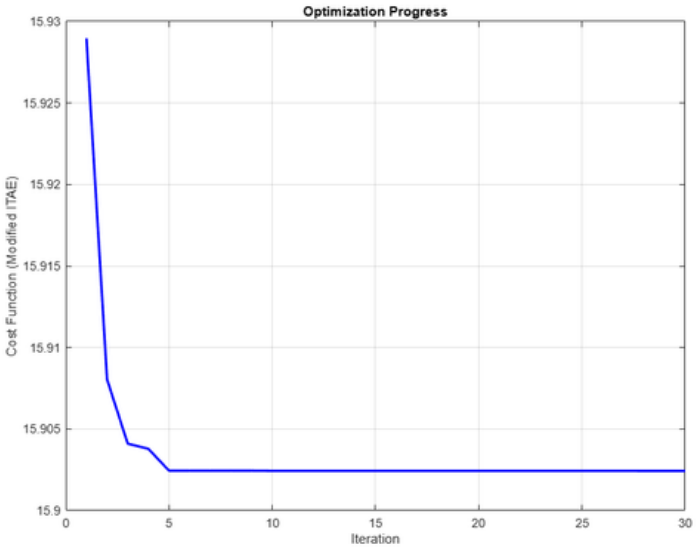
```
end
% Display final results
disp('Cost Function (Modified ITAE):');
disp(obj_Gbest);
disp('Optimal PID Parameters:');
disp(['Kp: ', num2str(xg(1))]);
disp(['Ki: ', num2str(xg(2))]);
disp(['Kd: ', num2str(xg(3))]);
% Final optimized response
Gc = pid(xg(1), xg(2), xg(3));
Gcf = feedback(Gc * G, 1);
y_optimized = lsim(Gcf, u, tsim);
% Performance metrics
rise_time = NaN;
for i = 1:length(y_optimized)
    if y_optimized(i) >= 0.9 * r_s
        rise_time = tsim(i); break;
    end
end
settling_time = NaN;
for i = length(y_optimized):-1:1
    if abs(y_optimized(i) - r_s) > 0.02 * r_s
        settling_time = tsim(i); break;
    end
end
```

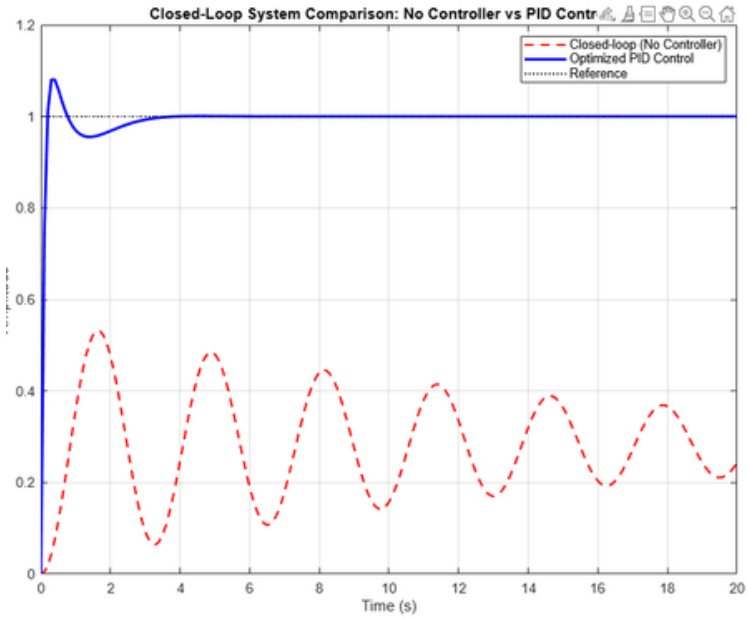


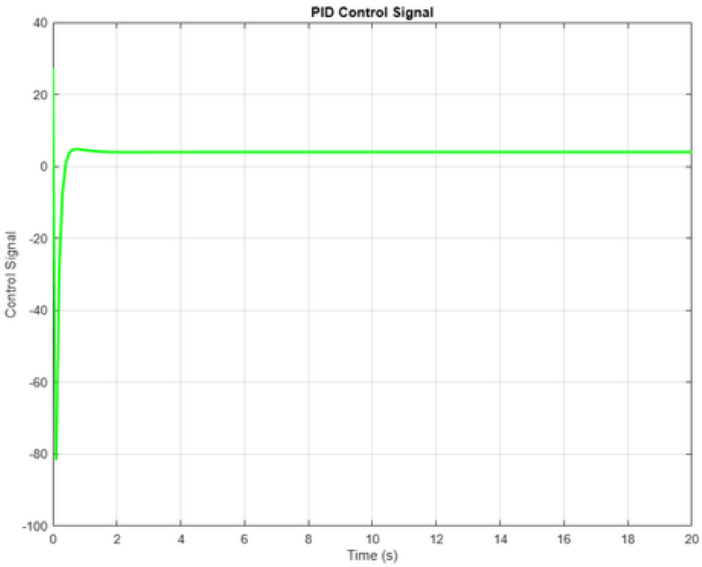

```
end
if isnan(settling_time)
    settling_time = tsim(end);
end
overshoot = max(0, 100 * (max(y_optimized) - r_s) / r_s);
disp(['Rise Time: ', num2str(rise_time), ' seconds']);
disp(['Settling Time: ', num2str(settling_time), ' seconds']);
disp(['Overshoot: ', num2str(overshoot), '%']);
% Comparison plot: no controller vs PID
figure(3);
plot(tsim, y_uncontrolled, 'r--', 'LineWidth', 1.5);
hold on;
plot(tsim, y_optimized, 'b', 'LineWidth', 2);
plot(tsim, ones(size(tsim)), 'k:', 'LineWidth', 1);
xlabel('Time (s)');
ylabel('Amplitude');
legend('Closed-loop (No Controller)', 'Optimized PID Control', 'Reference');
title('Closed-Loop System Comparison: No Controller vs PID Controller');
grid on;
% Control signal calculation
r = ones(size(tsim));
e = r(:) - y_optimized(:);
dt = tsim(2) - tsim(1);
P = xg(1) * e;
I = zeros(size(e)); integral_sum = 0;
for i = 1:length(e)
    integral_sum = integral_sum + e(i) * dt;
    I(i) = xg(2) * integral_sum;
end
D = zeros(size(e)); D(1) = 0;
for i = 2:length(e)
    D(i) = xg(3) * (e(i) - e(i-1)) / dt;
end
u_control = P + I + D;
% Plot control signal
figure(4);
plot(tsim, u_control, 'g', 'LineWidth', 2);
xlabel('Time (s)');
ylabel('Control Signal');
title('PID Control Signal');
grid on;
% Lévy flight function
function L = levy(d)
    beta = 3/2;
    sigma = (gamma(1+beta)*sin(pi*beta/2)/(gamma((1+beta)/2)*beta*2^((beta-1)/2)))^(1/beta);
    u = randn(1,d) * sigma;
    v = randn(1,d);
    L = u./abs(v).^(1/beta);
end
```

- **output(ITAE)**









Command Window

```
Cost Function (Modified ITAE):
   15.9024

Optimal PID Parameters:
Kp: 24.8385
Ki: 25
Kd: 12.0037
Rise Time: 0.2 seconds
Settling Time: 2.3 seconds
Overshoot: 8.0301%
>>
```

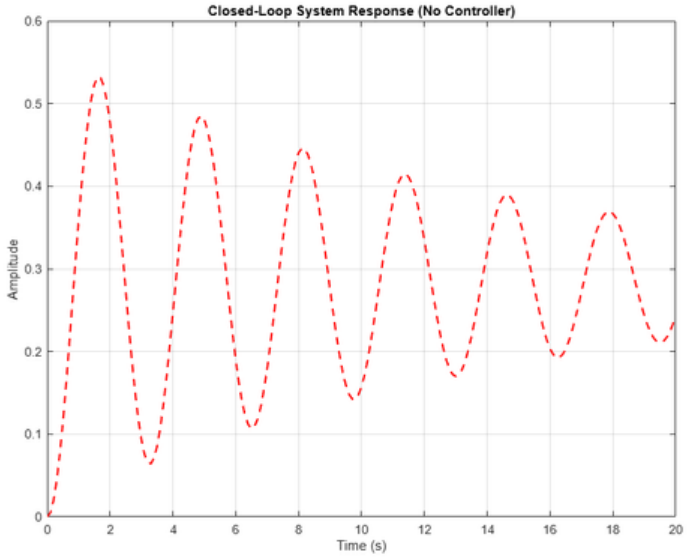
- **Matlap code(ISE)**

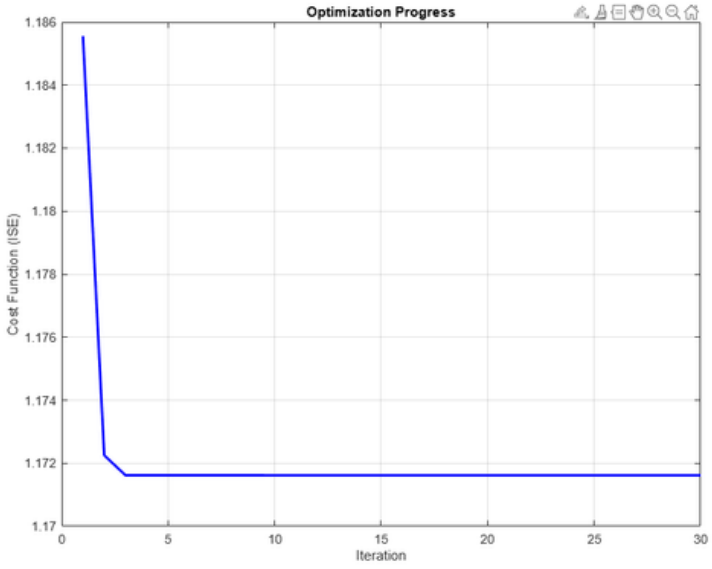
The code is the same as The (ITAE) different only in:

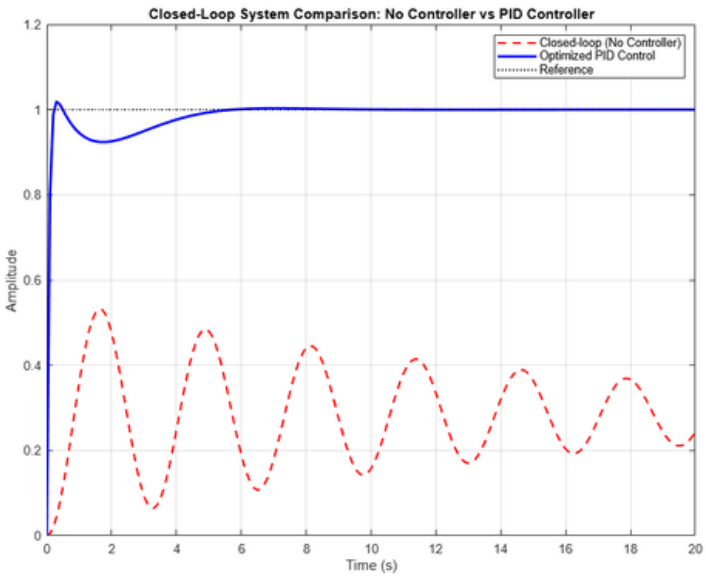
```
overshoot = max(0, max(y) - r_s);
error = y - r_s;
ise = sum(error.^2);
overshoot_penalty = 50 * overshoot^2;
ffb(i) = ise + overshoot_penalty;
```

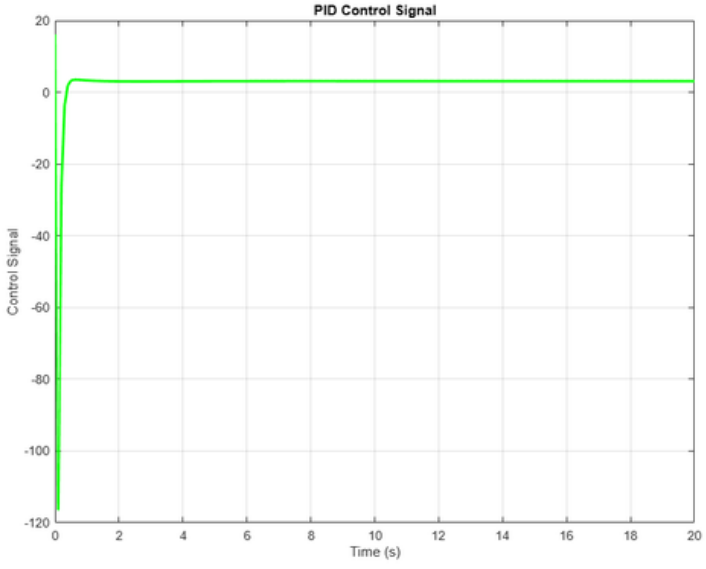
$$ISE = \int_0^\infty e^2(t)dt$$

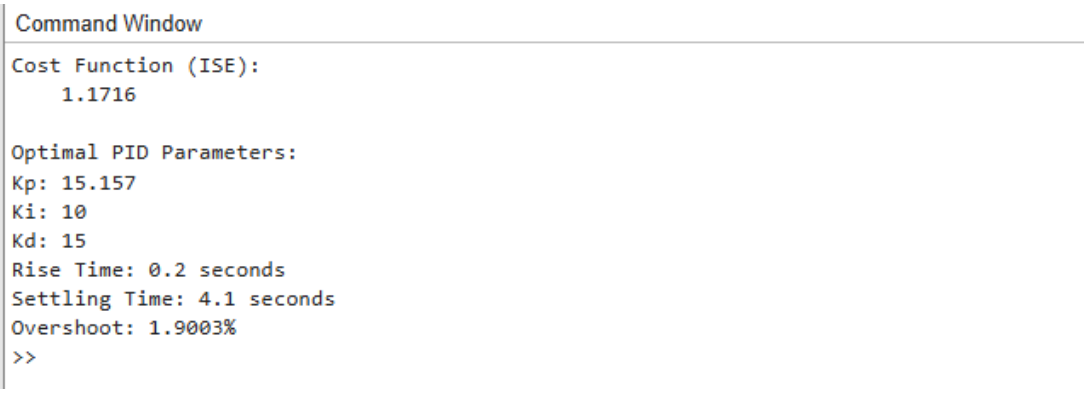
- **output(ISE)**











Why does ITAE reduce the settling time?

1. Because it penalizes errors that persist over time

2. As a result, the system tends to reach the desired value quickly in order to avoid a high "time-weighted penalty".

3. Even if some overshoot occurs, it is not a major issue as long as the system settles quickly

Tuning result using ITAE:

>Shorter settling time

>Overshoot may be slightly higher

Why does ISE reduce the overshoot?

1.Because it strongly penalizes large errors (as the error is squared).

2.This encourages the system to avoid aggressive responses and reduce overshoot so that the initial large errors don’t contribute significantly to the cost.

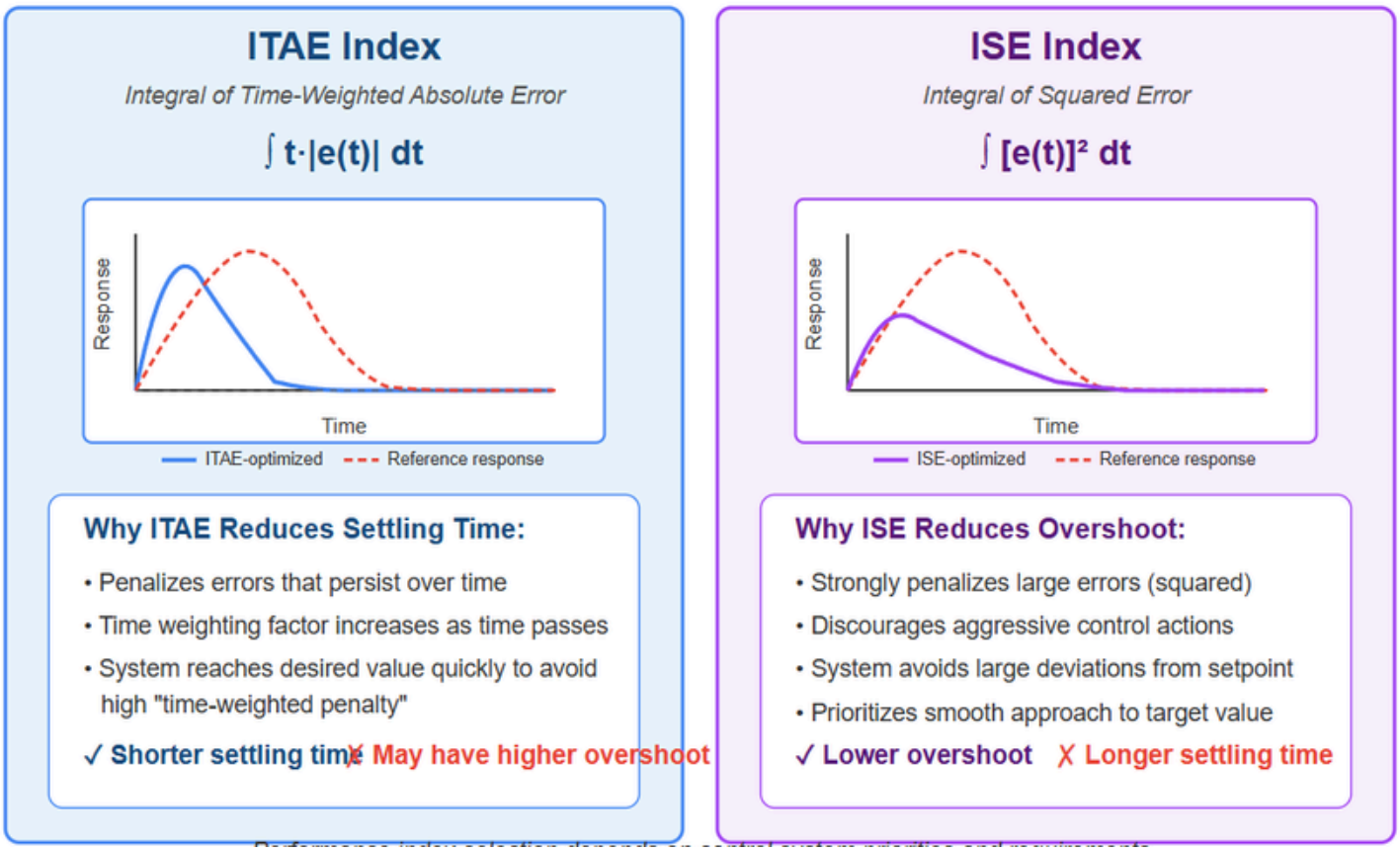
Tuning result using ISE:

>Lower overshoot

>Longer settling time

- **Comparison between ISE and ITAE :**

Criteria	ITAE	ISE
Objective	Penalizes errors that persist over time	Penalizes large errors more heavily
Cost Function Value	15.9024	1.1716
Optimal Kp	24.8385	15.157
Optimal Ki	25	10
Optimal Kd	12.0037	15
Rise Time	0.2 seconds	0.2 seconds
Settling Time	2.3 seconds	4.1 seconds
Overshoot	8.0301 %	1.9003 %
Tuning Foucus	Faster response, better settling	Smoother response, less overshoot
Common Use	When quick settling is prioritized	When stability and minimal overshoot are key



In the end, we have implemented two methods for tuning the PID controller.

Thanks...