2025

PID Tuning Report

Presented for :

Dr: Ahmed Osman

Computer and Systems Engineering Department

Table of Contents

Team members

Hossam Salah Ibrahim Mahmoud	**20812021100594**
Muhammad Tarek Abdel-Hafez Abdel-Wahed	**20812021100973**

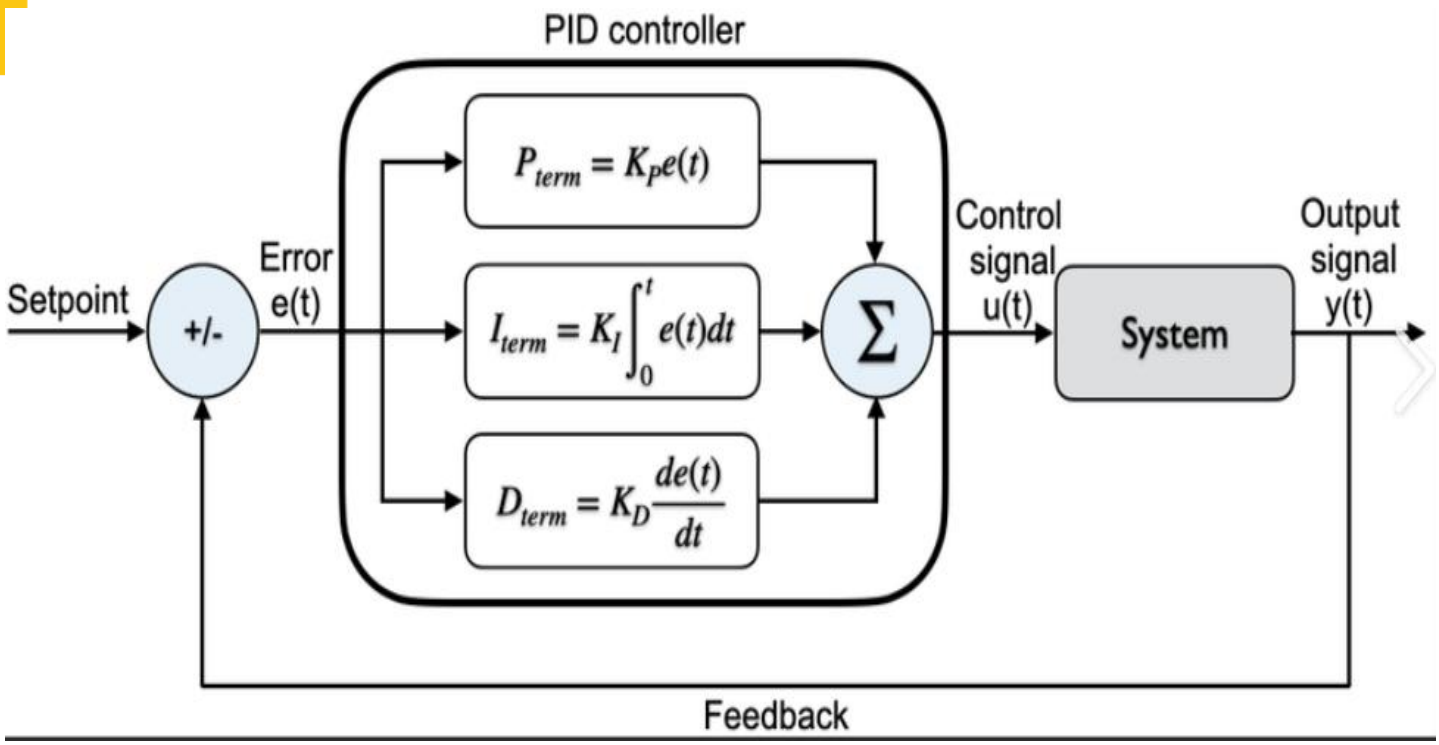
Introduction

This project implements an adaptive PID (Proportional-Integral-Derivative) controller tuning system using MATLAB. The main goal is to optimize the PID parameters for a user-defined system, improving its performance in terms of overshoot, settling time, and steady-state error. The system features a graphical input dialog, step response comparison, and quantitative metrics display.

What is PID Control

PID stands for **Proportional–Integral–Derivative**. It is a feedback control strategy used to continuously calculate an error value as the difference between a desired setpoint and a measured process variable, and apply a correction based on proportional, integral, and derivative terms.

The controller attempts to minimize the error over time by adjusting the control inputs. PID controllers are found in everything from temperature control systems and autopilot systems to manufacturing machinery and even camera gimbals.



Project Objectives

- Allow users to input any linear transfer function, including those with time delay.
- Perform initial PID tuning using the Skogestad method.
- Apply adaptive gradient-based optimization to refine the PID parameters.
- Visually compare system performance before and after tuning.
- Display final performance metrics: overshoot, settling time, and steady-state error.

Methodology

- **User Input via GUI:** A prompt requests the user to enter a transfer function as a string (e.g., 1/((s+1)*(s+2))).
- **Transfer Function Parsing:** The string is parsed and converted into a MATLAB transfer function object, including support for delays using symbolic processing.
- **Initial Tuning:** The Skogestad method estimates initial values for Kp, Ki, and Kd using the step response characteristics of the system.
- **Closed-Loop Simulation (Before Tuning):** The step response of the closed-loop system with initial PID values is generated.
- **Adaptive PID Optimization:** A gradient descent-based algorithm iteratively adjusts Kp, Ki, and Kd to minimize the cost function related to the error integral.
- **Closed-Loop Simulation (After Tuning):** The optimized PID values are used to simulate the final closed-loop response.
- **Results Visualization:** A plot compares the system response before and after tuning.
- **Performance Analysis:** The script calculates overshoot, settling time, and steady-state error of the optimized system and displays them in a message box.

Skogestad method

The Skogestad method offers a simple and effective starting point for tuning PID parameters (Kp, Ki, Kd). Instead of relying on random values, this analytical method provides a structured estimation based on step response characteristics.

How It Works:

1. A **step input** is applied to the system to observe how it reacts over time.
2. From the system's response, two important parameters are extracted:
   - **θ (Theta)**: The delay time, which is the time the system takes to start responding from rest.
   - **τ (Tau)**: The time required for the system to reach approximately 63% of its final value after the delay.
3. These parameters are used to calculate the initial PID values using the following Skogestad formulas:
   - `Kp = (1/K) * (0.5 * τ) / θ` `Ti = min(τ, 4θ)` `Ki = Kp / Ti` `Kd = θ`

Purpose:

This method ensures that the controller starts from a reasonable and functional state, providing a good foundation for further optimization.

Adaptive PID Optimization

Concept Overview:

Initial tuning may not guarantee optimal performance. Therefore, an **adaptive optimization algorithm** is applied to refine the PID parameters using a gradient-descent-based learning approach.

How It Works:

1. A **cost function** is defined to evaluate how good or bad the current PID configuration is. The function penalizes long response times and high errors using:

$$J = \int t * |1 - y(t)| \, dt$$

This favors faster and more accurate responses.

Gradients of the cost function are estimated for each parameter (Kp, Ki, Kd) using finite differences. This shows how sensitive the cost is to changes in each parameter.

Gradient descent is applied:

- Each parameter is adjusted in the opposite direction of its gradient to reduce the cost.
- For example:

```
Kp = Kp - alpha * dJ/dKp
```

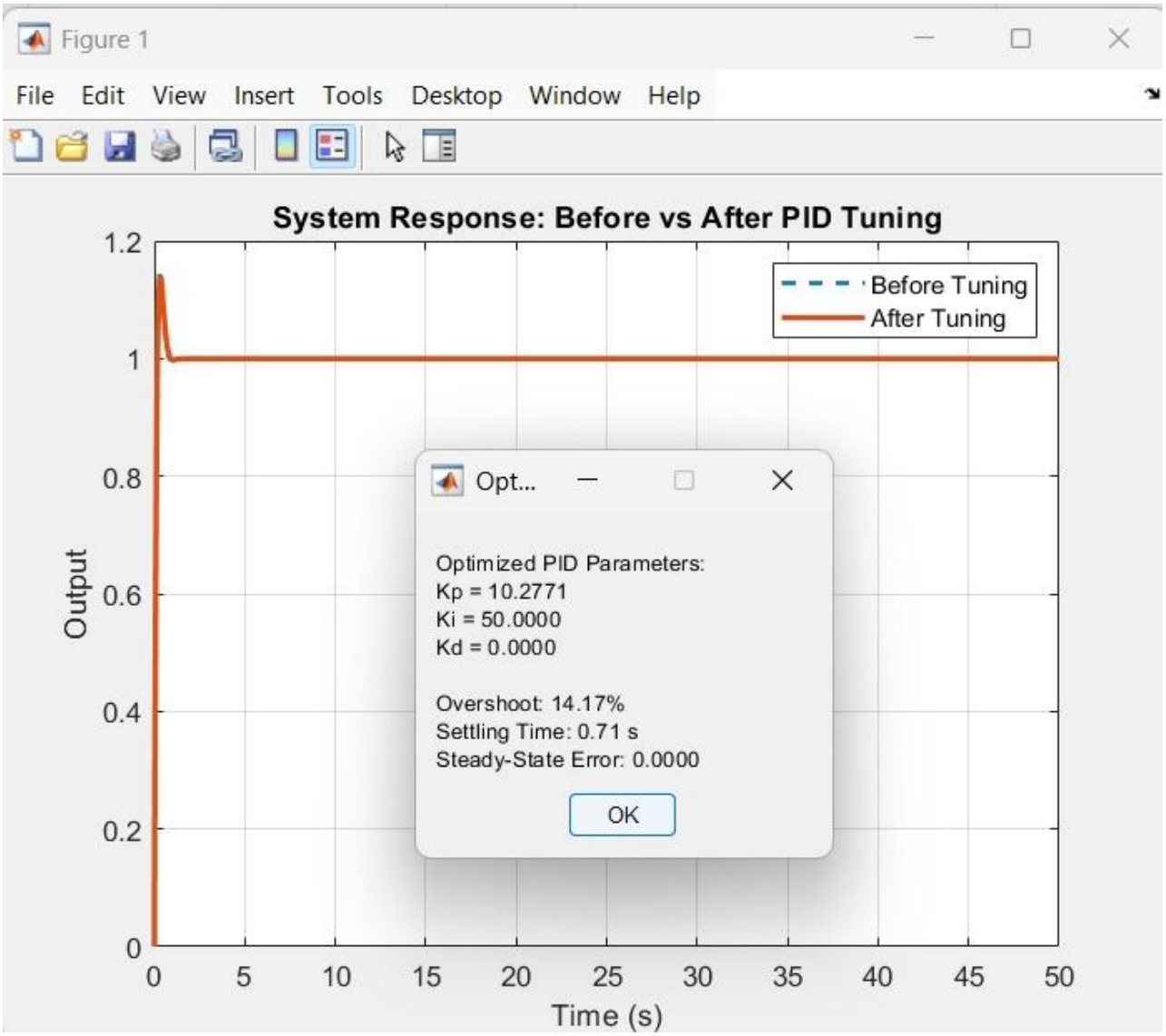
The process is repeated over multiple **iterations**, and a learning rate (α) that decreases over time is used to ensure convergence and stability.

Results

Upon running the program:

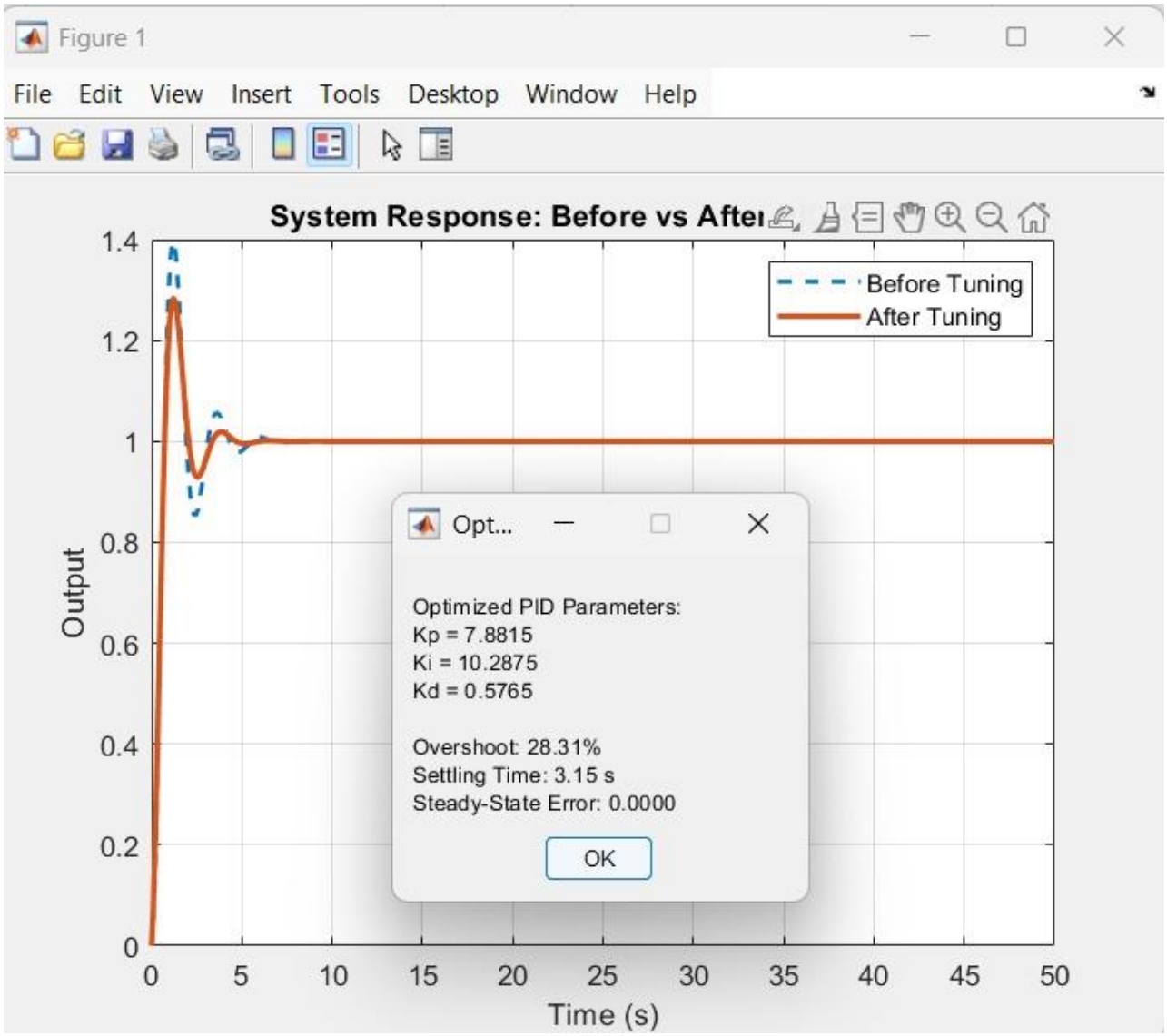
- The system requests a transfer function from the user.
- It visualizes the improvement in system response after adaptive tuning.
- It displays optimized PID values and performance metrics, showing clear improvement in system behavior.

Testing and Results



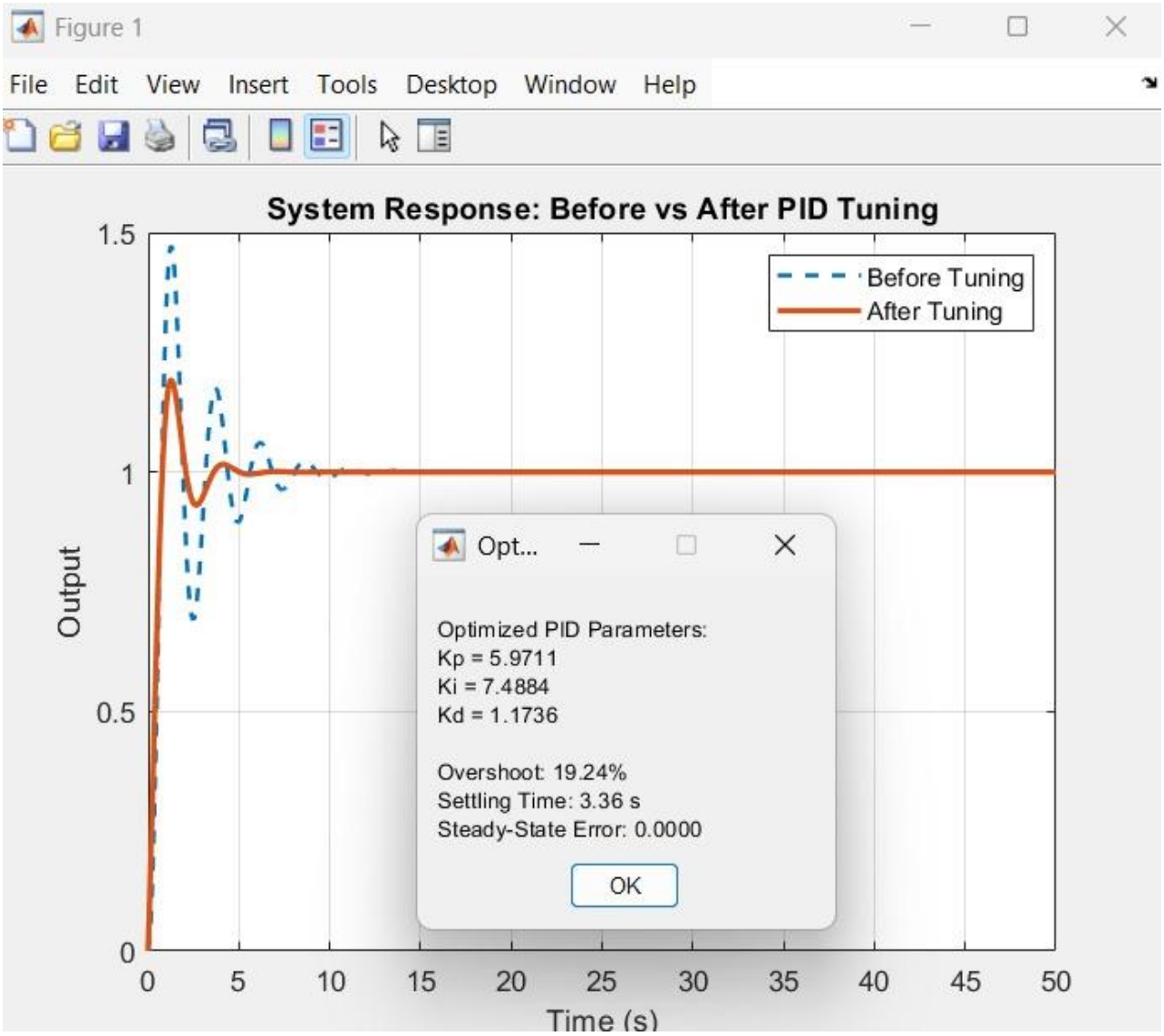
1/(s + 1)

Simple 1st Order System



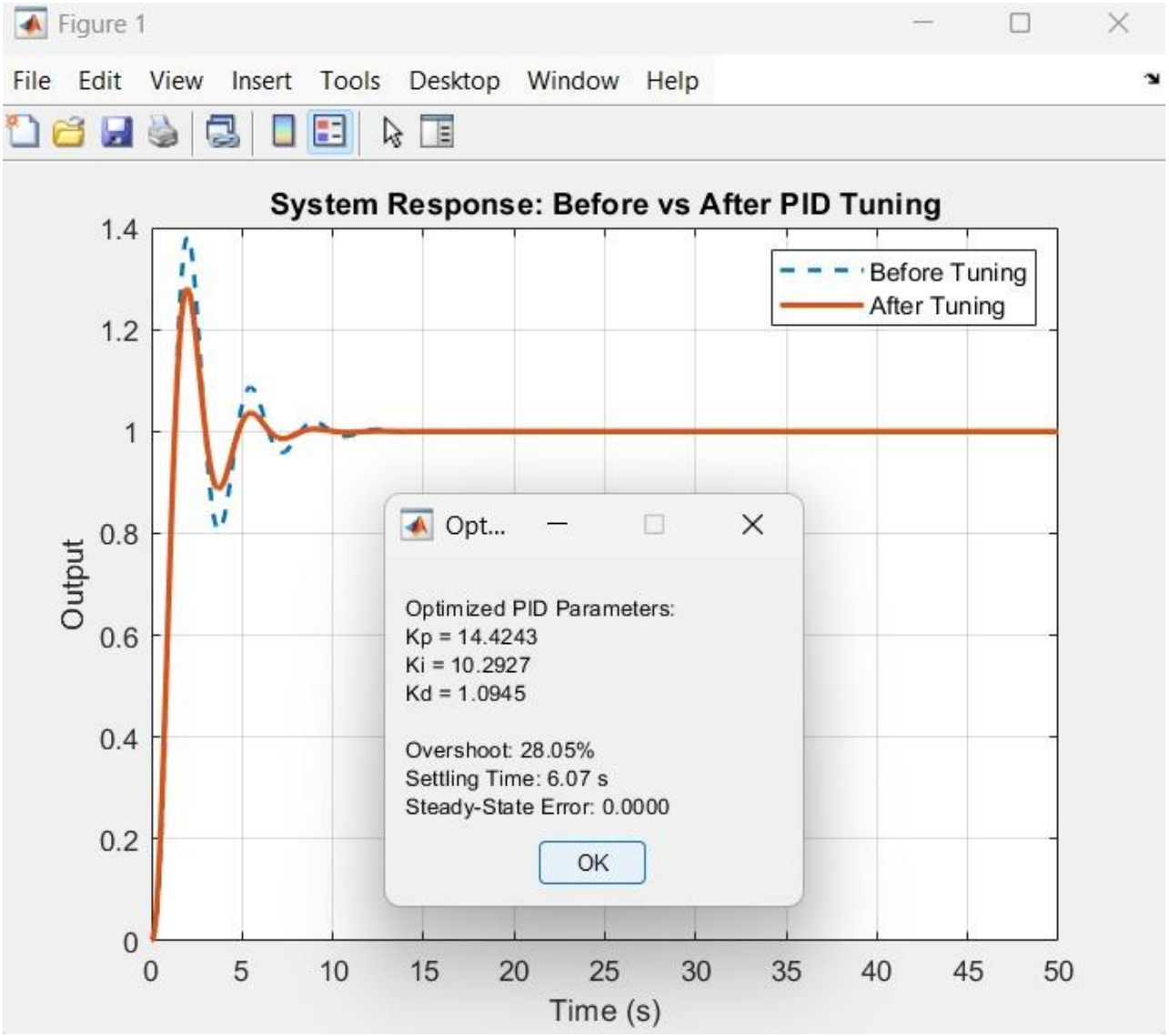
1/((s + 1)*(s + 2))

Stable 2nd Order System



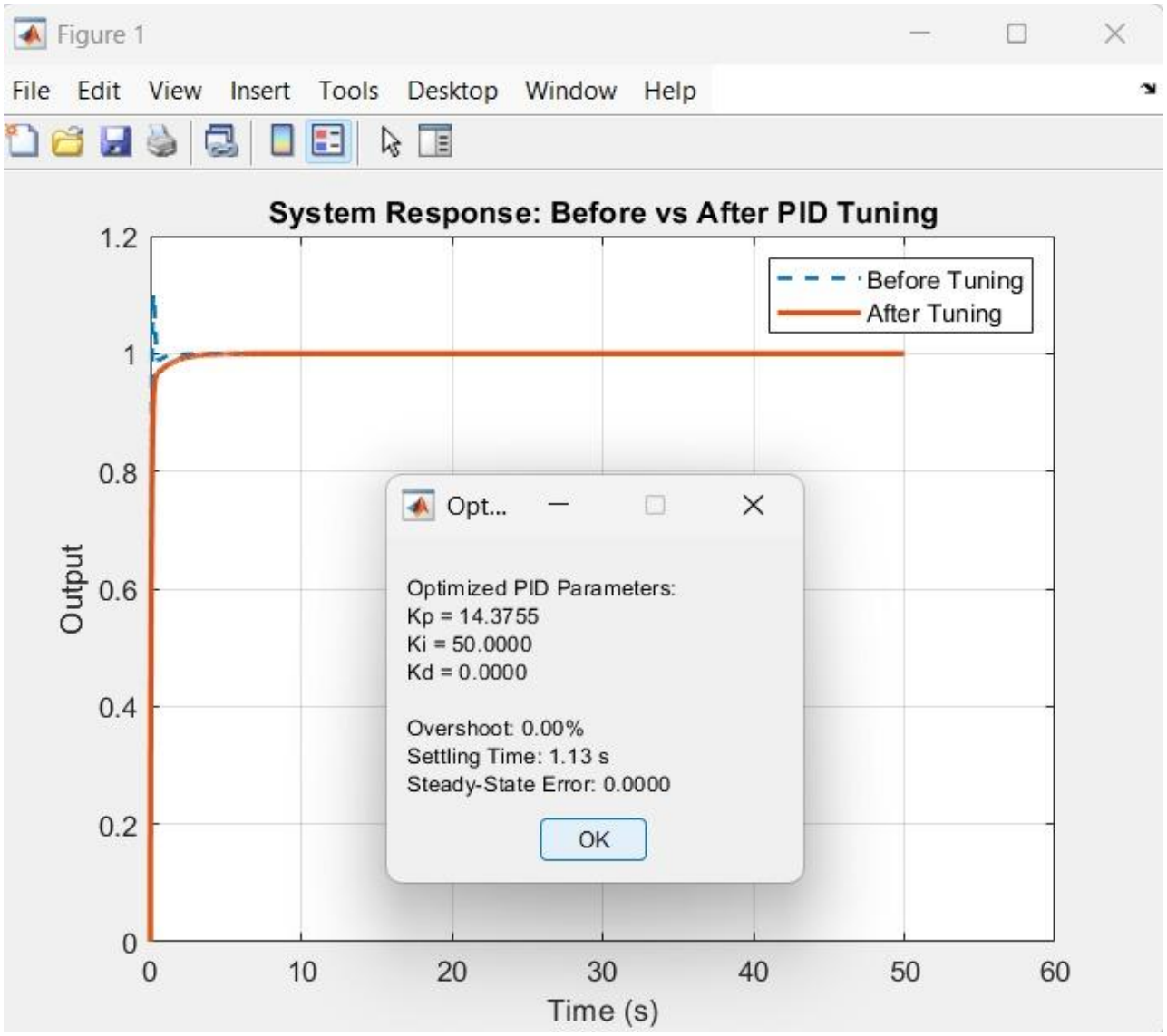
1/(s^2 + 2*s + 2)

2nd Order System (underdamped)



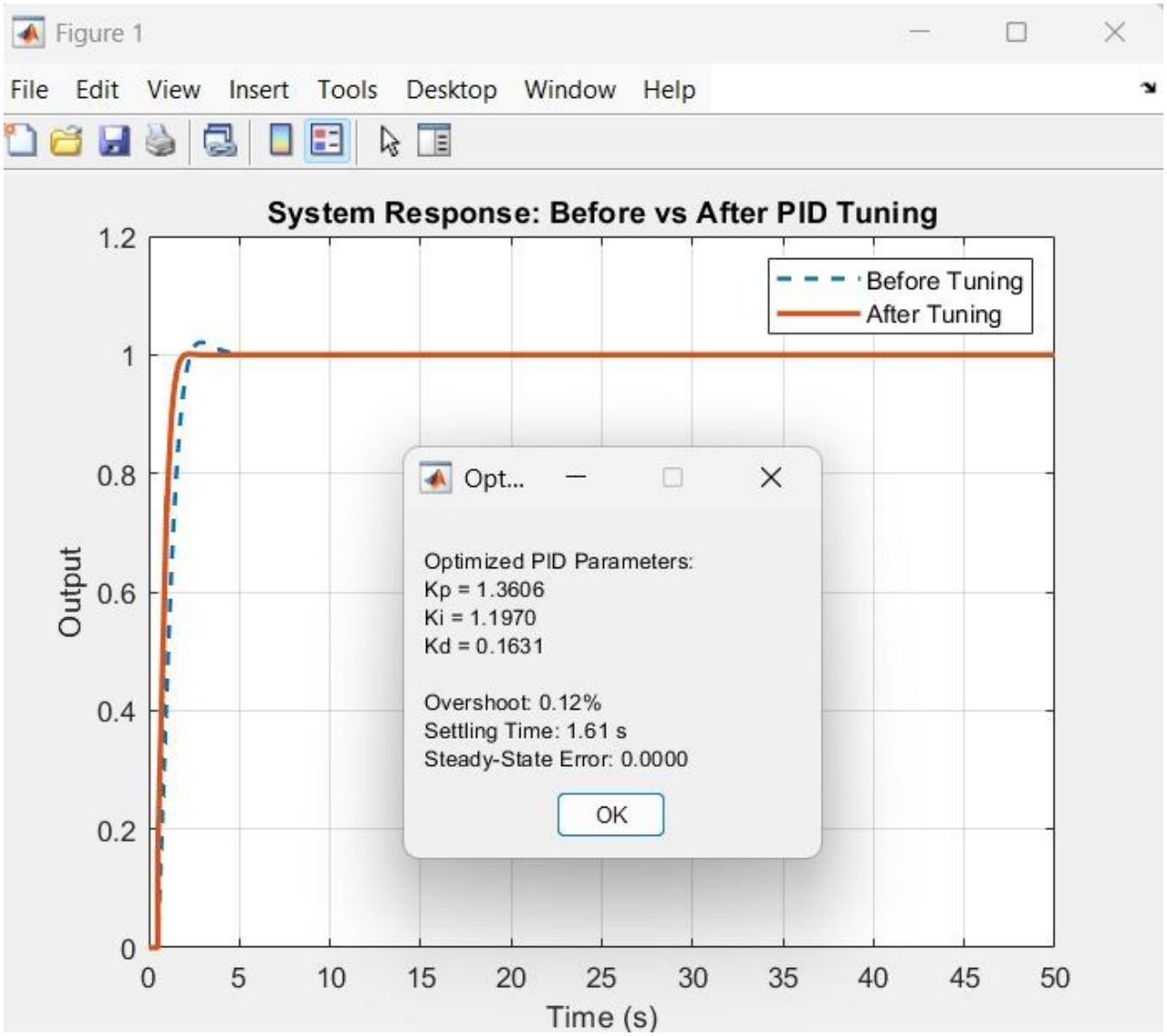
1/((s + 1)(s + 2)(s + 3))

3rd Order System



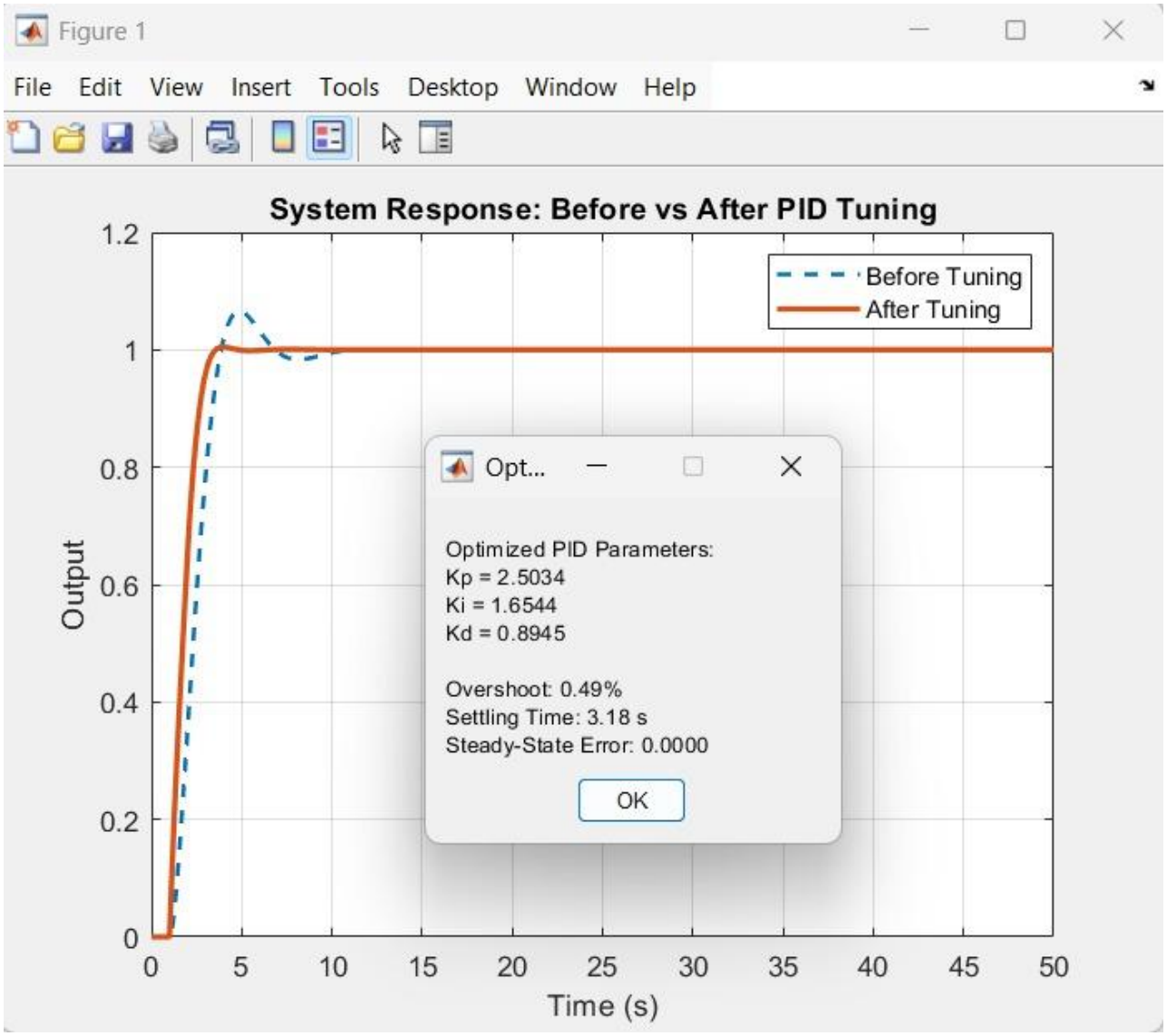
(s + 1)/((s + 2)*(s + 3))

Contains Zero , Positive Pole and Common Pole



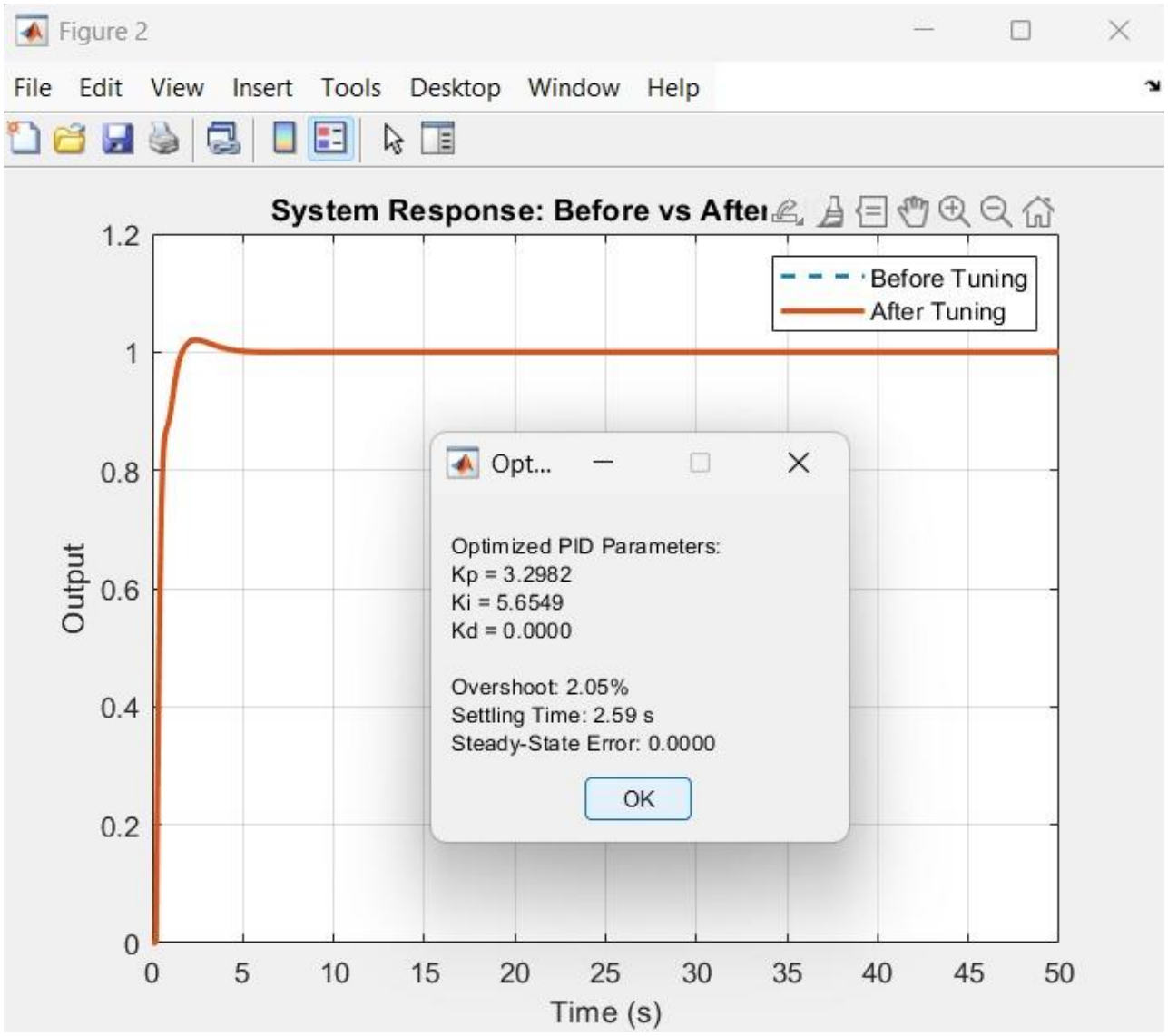
1/(s + 1)*exp(-0.5*s)

Simple 1st System with Small Time Delay = 0.5 Second



1/((s + 1)*(s + 3))*exp(-1*s)

2nd Order System with Time Delay = 1 Second



(s + 2)/((s + 5)*(s + 1))*exp(-0.2*s)

System Contains Zero & Time Delay Together