Project:

System Identification and Intelligent Controller Design on a Mass Spring Damper System

Banaan Kiamanesh

Winter, 2023

**** This project is meant for the Industrial Control Systems Course Final Project.***

Project Flow:

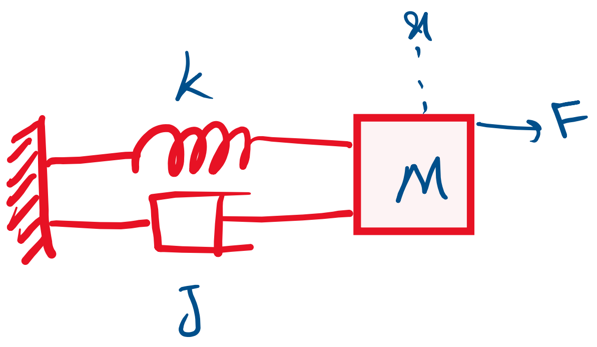
The goal of this project is to do a simple and secure system identification process for a Mass-Spring-Damper System using Minimum Variance Unbiased Estimator Linear Method. Then 3 ways of designing a digital PID controller are implemented for the plant:

1. Using MATLAB's PID tuner
2. Manual Tuning
3. Intelligent Methods.

Although the reasonably better way of tuning a PID controller will be to make some assumptions on the controller gains based on the simulation results using a trial and error procedure and using them as initial conditions for an Intelligent method, therefore fine-tuning the gains in the process.

Part 0: Mass-Spring-Damper Plant Description

$$\boldsymbol{G}_{(s)} = \frac{\frac{1}{M}}{s^2 + {}^{J}\!/_{M}\, s + \frac{K}{M}}$$

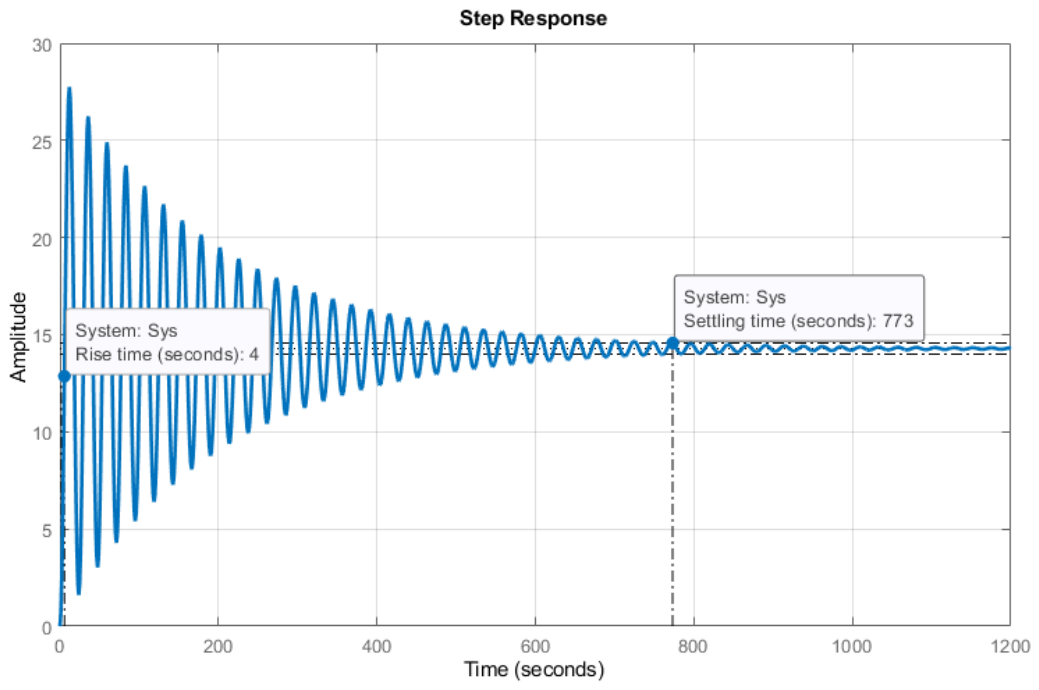


The system is assumed to have parameter of:

$$J = 0.01$$
$$M = 1$$
$$K = 0.07$$

Therefore, the system step response is simulated as follows:

```
clear;
close all;
clc;
format compact;
rng(1, 'twister');

%% System Parameter Declaration / Sys Continous Time Representation

% Mass Spring Damper System Parameters
J = 0.01;                   % Damper Const
M = 1;                      % Mass Const
K = 0.07;                   % Spring Const

s = tf('s');
Sys = (1/M) / (s^2 + J/M * s + K/M);

% Check System and Specify the Sampling Time
step(Sys); grid on

tmp = findobj(gcf, 'type', 'figure');
set(tmp, 'Name', "Continuous System Step Responce");

tmp = findobj(gcf, 'type', 'line');
set(tmp, 'linewidth', 2);
```



Part 1: System Discretization

The system shown in the modelling section, the system is a typical second order system. Bringing it into the z domain will map the poles of the system and again the system will have a couple poles and it may or may not have a zero.

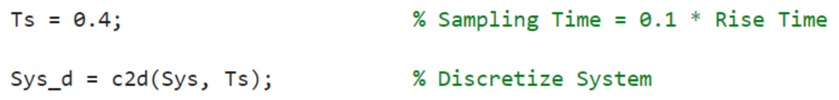
$$z = e^{st}$$

$$\frac{Y_{(z)}}{X_{(z)}} = \frac{\theta_3 z + \theta_4}{z^2 + \theta_1 z + \theta_2}$$

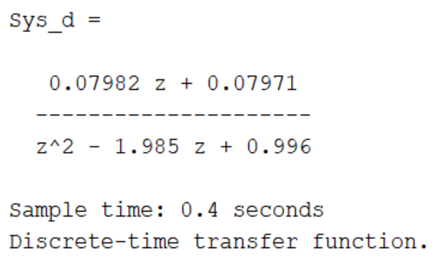
*** There are 4 unknown parameters as shown.**

Part 1*: What do we Expect to get?

The system has to be discretized. Then the difference equation is extracted.



```
Ts = 0.4;                       % Sampling Time = 0.1 * Rise Time

Sys_d = c2d(Sys, Ts);           % Discretize System
```



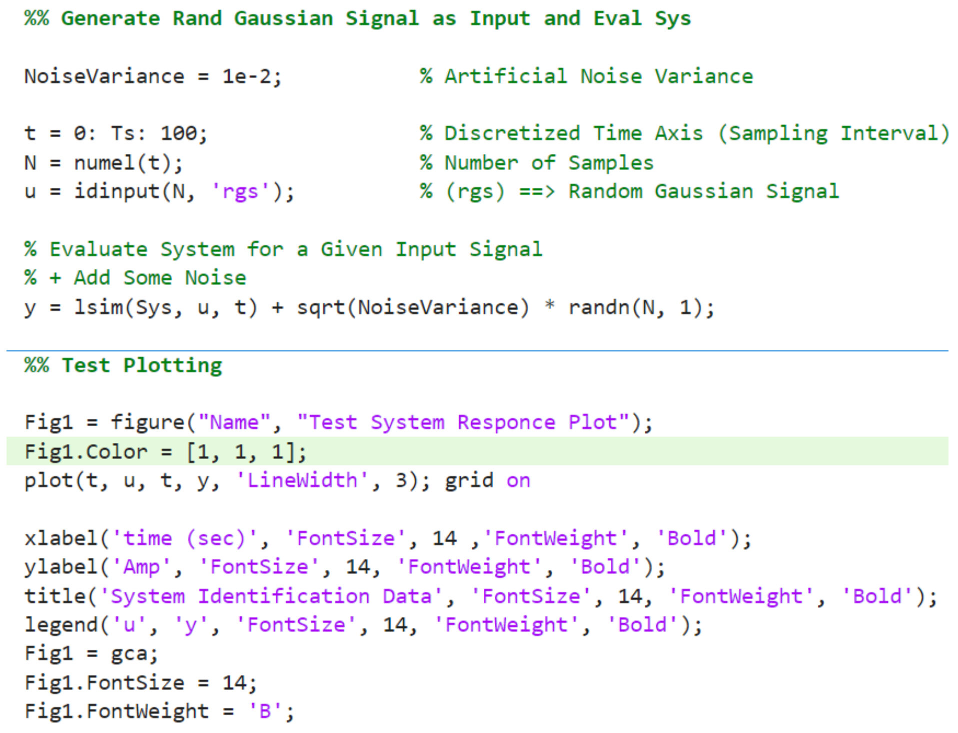
```
Sys_d =

   0.07982 z + 0.07971
  ---------------------
  z^2 - 1.985 z + 0.996

Sample time: 0.4 seconds
Discrete-time transfer function.
```

$$\boldsymbol{y(n+2) = 1.985y(n+1) - 0.996y(n) + 0.7982u(n+1) + 0.07971u(n)}$$

The coefficients of this equation are the desired values that we tend to get.

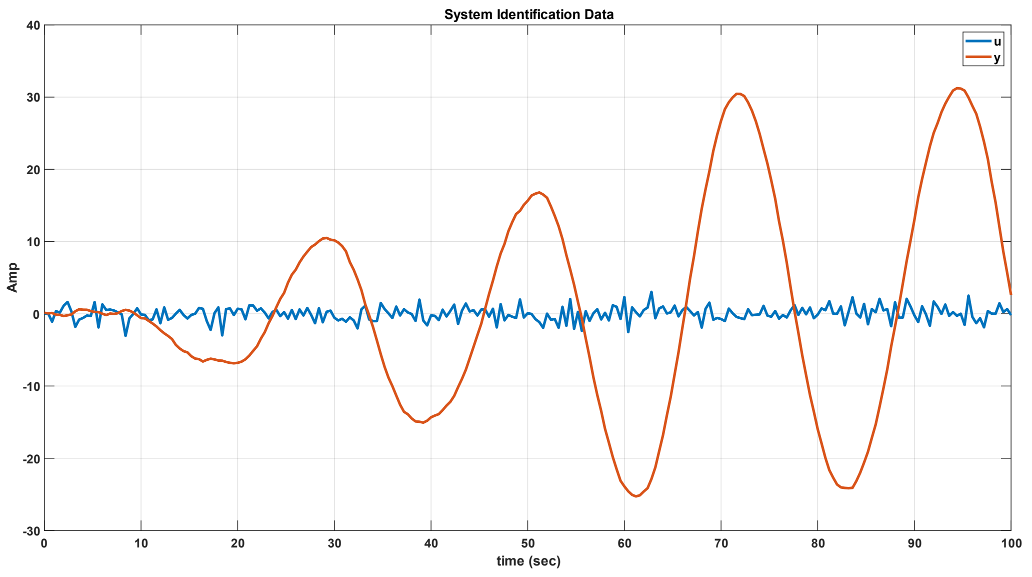
Part 2: Simulation of a Single Realization

To make simulate the process in a way that is near the real conditions a random gaussian noise is also added to the system response results. Also, from the system identification theory we know that



```
%% Generate Rand Gaussian Signal as Input and Eval Sys

NoiseVariance = 1e-2;           % Artificial Noise Variance

t = 0: Ts: 100;                 % Discretized Time Axis (Sampling Interval)
N = numel(t);                   % Number of Samples
u = idinput(N, 'rgs');          % (rgs) ==> Random Gaussian Signal

% Evaluate System for a Given Input Signal
% + Add Some Noise
y = lsim(Sys, u, t) + sqrt(NoiseVariance) * randn(N, 1);

%% Test Plotting

Fig1 = figure("Name", "Test System Responce Plot");
Fig1.Color = [1, 1, 1];
plot(t, u, t, y, 'LineWidth', 3); grid on

xlabel('time (sec)', 'FontSize', 14 ,'FontWeight', 'Bold');
ylabel('Amp', 'FontSize', 14, 'FontWeight', 'Bold');
title('System Identification Data', 'FontSize', 14, 'FontWeight', 'Bold');
legend('u', 'y', 'FontSize', 14, 'FontWeight', 'Bold');
Fig1 = gca;
Fig1.FontSize = 14;
Fig1.FontWeight = 'B';
```

the perfect signal for identifying the system is a Random Gaussing Input. The system step response is simulated for 10 seconds as shown below.



Part 3: Parameter Estimation

Turning the system response into the matrix format, we have:

in each time step:

$$y(n+2) = \underbrace{[y(n+1)\ \ y(n)\ \ u(n+1)\ \ u(n)]}_{\text{a single observation}} \begin{bmatrix} \theta_1 \\ \theta_2 \\ -\theta_3 \\ -\theta_4 \end{bmatrix}$$

overall realization:

$$\underbrace{\begin{bmatrix} y_0(n+1) \\ y_1(n+1) \\ . \\ . \end{bmatrix}}_{\text{output matrix}} = \underbrace{\begin{bmatrix} y_0(n+1) & y_0(n) & u_1(n+1) & u_1(n) \\ y_1(n+1) & y_1(n) & u_1(n+1) & u_1(n) \\ & & . & \\ & & . & \end{bmatrix}}_{\text{observation matrix}} \underbrace{\begin{bmatrix} \theta_1 \\ \theta_2 \\ -\theta_3 \\ -\theta_4 \end{bmatrix}}_{\text{parameter matrix}}$$

$$x = H\theta$$
$$\theta = H^{-1}x$$

```
%% Test System Identification x = H * theta

% Make the X(Measurement) and H(Observation) Matrices
x = zeros(N - 2, 1);
H = zeros(N - 2, 4);

for n = 1: N - 2
    x(n) = y(n + 2);
    H(n, :) = [y(n + 1), y(n), u(n + 1), u(n)];
end

Theta = H \ x;
EstimatedSys = tf(Theta(3 : 4)', [1 -Theta(1 : 2)'], Ts)
```

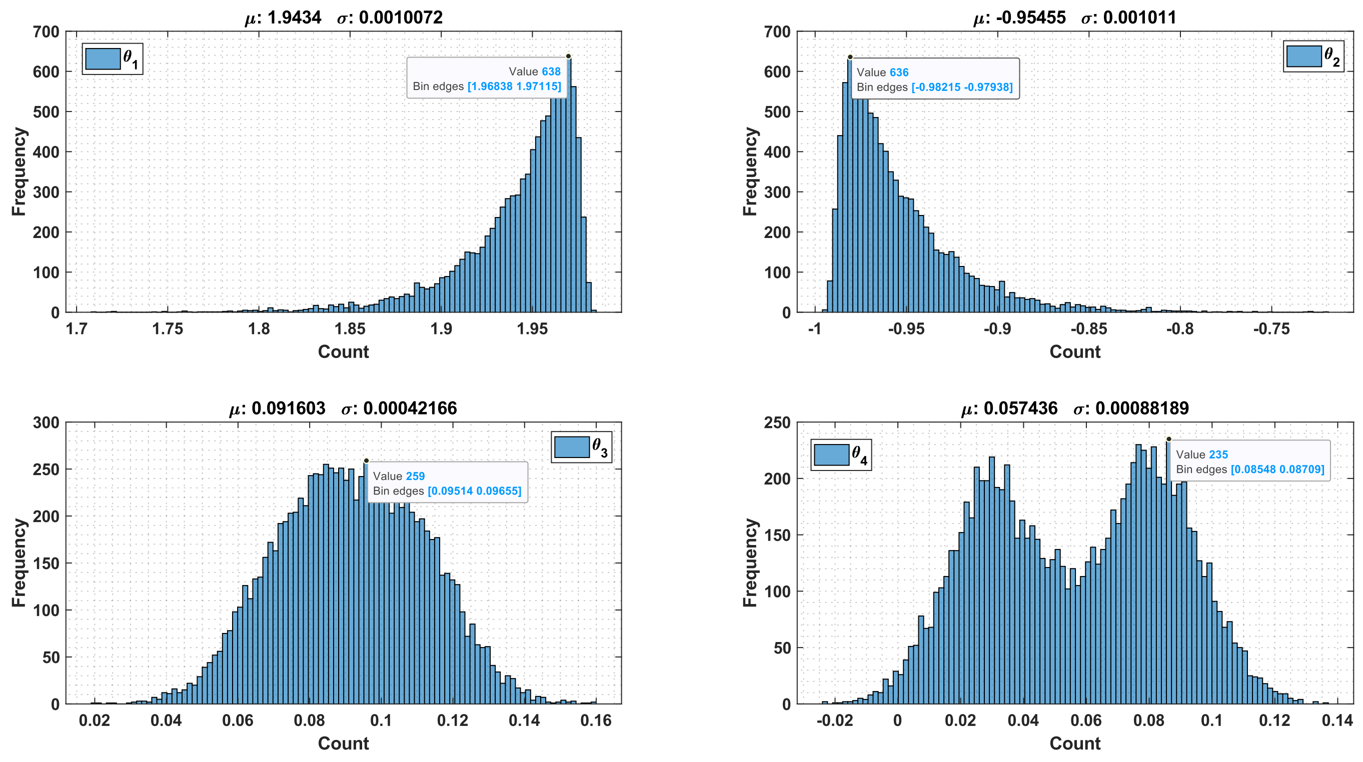
```
EstimatedSys =

   0.08894 z + 0.08611
  ----------------------
  z^2 - 1.973 z + 0.9837

Sample time: 0.4 seconds
Discrete-time transfer function.
```

Part 4: Statistical Validation of the Estimated Parameters

For the purpose, 10000 realizations of the process is simulated and parameter are collected.

```
%% Statistical Analysis of the Process

M = 10000;               % Number of Realizations
Theta = zeros(4, M);     % Parameter Matrix

for k = 1 : M

    % Generate Random Gaussian Input
    u = idinput(N , 'rgs');

    % Evaluate System for Given Input Signal
    y = lsim(Sys , u  , t) + sqrt(NoiseVariance) * randn(N , 1);

    % Create Observation and Measurement Matrices
    x = zeros(N - 2, 1);
    H = zeros(N - 2, 4);

    for n = 1 : N - 2
        x(n) = y(n+2);
        H(n, :) = [y(n+1), y(n), u(n+1), u(n)];
    end

    % Evaluate Theta_i
    Theta(:, k) = H\x;
end
```

To check the statistical properties of the estimated parameters, the histogram of each parameter is plotted.



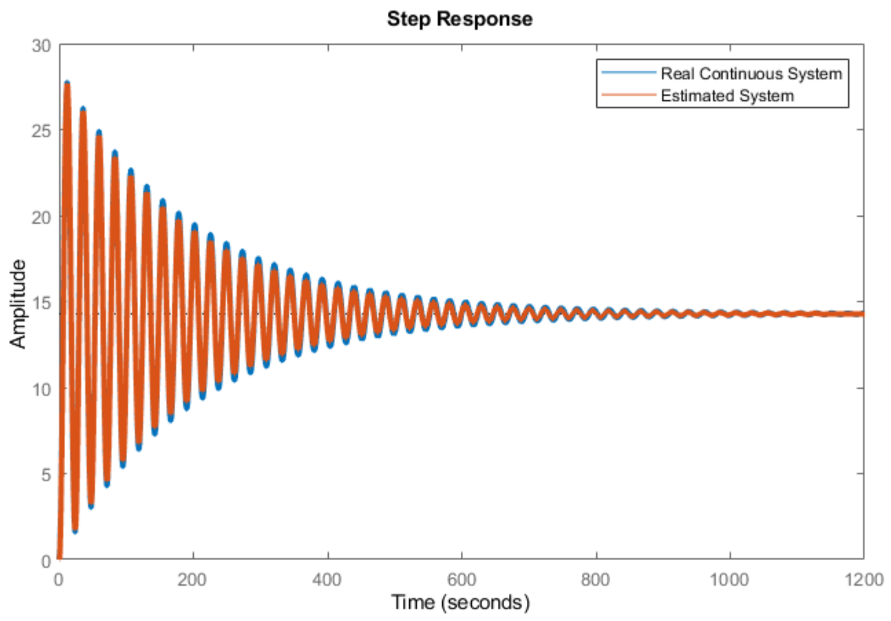
```matlab
%% Statistical Evaluation: Histogram Plotting

Fig2 = figure("Name", "Estimation Properties");
Fig2.Color = [1, 1, 1];

for i = 1:size(Theta, 1)

    subplot(2, 2, i);
    histogram(Theta(i, :), 100, "Normalization", "count");
    grid minor;

    xlabel('Count' , 'FontSize' , 14 , 'FontWeight' , 'Bold') ;
    ylabel('Frequency' , 'FontSize' , 14 , 'FontWeight' , 'Bold') ;

    title(['\mu: ', num2str(mean(Theta(i, :))), ...
        '    \sigma: ', num2str(var(Theta(i, :)))] , ...
        'FontSize' , 12 , 'FontWeight' , 'Bold') ;

    legend(['\theta_', num2str(i)], 'FontSize' , 14 , 'FontWeight' , 'Bold') ;

    Fig2 = gca ;
    Fig2.FontSize = 14 ;
    Fig2.FontWeight = 'B' ;
end
```

Part 5: Final Parameter Selection and Model Validation

Here the Mean or Weighted Mean of all Histograms bins can be used for the Final values of Thetas. Or as a better alternative, mid-point of the bin with the highest frequency is used. As shown below the parameter are pretty close to the real parameters of the system.

```
%% Final Estimated System

% for Noise Variance of: 1e-4
ThetaHat = [1.9845
            -0.9957
            0.0797
            0.0803];

EstimatedSysFinal = tf(ThetaHat(3 : 4)', ...
                        [1 -ThetaHat(1 : 2)'], ...
                        Ts);
```

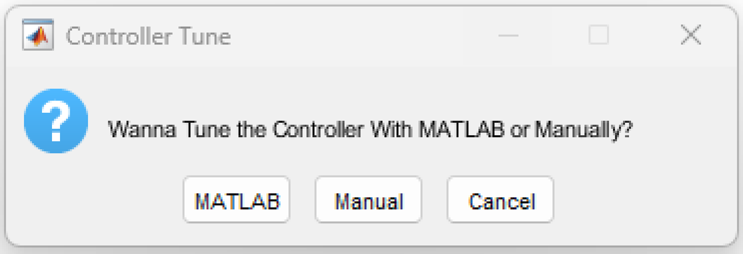
```
EstimatedSysFinal =

    0.0797 z + 0.0803
  ---------------------
  z^2 - 1.984 z + 0.9957

Sample time: 0.4 seconds
Discrete-time transfer function.
```



Part 6: Controller Design

For the controller a Digital PID Controller is used. And, in each section the step response of the system is checked for the performance of the controller.
This Section is separated into 3 main parts:

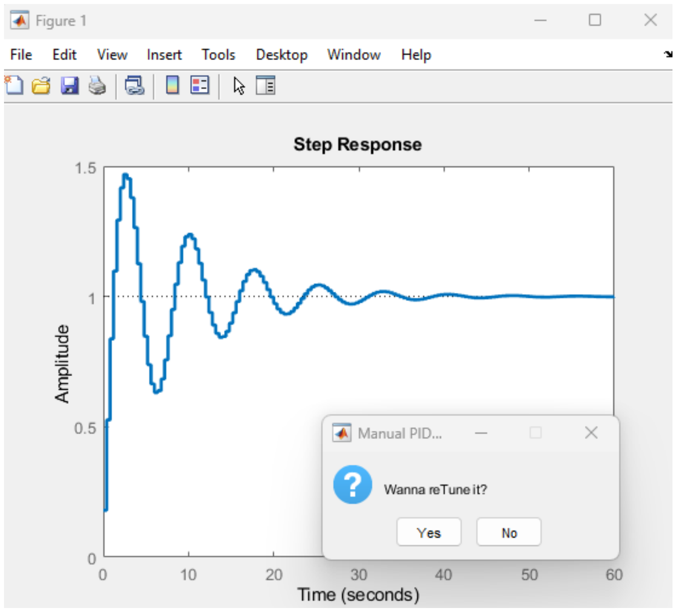
1. Manual Tuning
2. MATLAB PID-Tuner
3. Genetic Algorithm


Controller Tune

Wanna Tune the Controller With MATLAB or Manually?

6.1 Manual PID Tuning:

A simple yet handy tool is made to iteratively help the user evaluate to the performance of the controller as it first asks for the controller gains graphically, and then plots the step response of the controlled system.

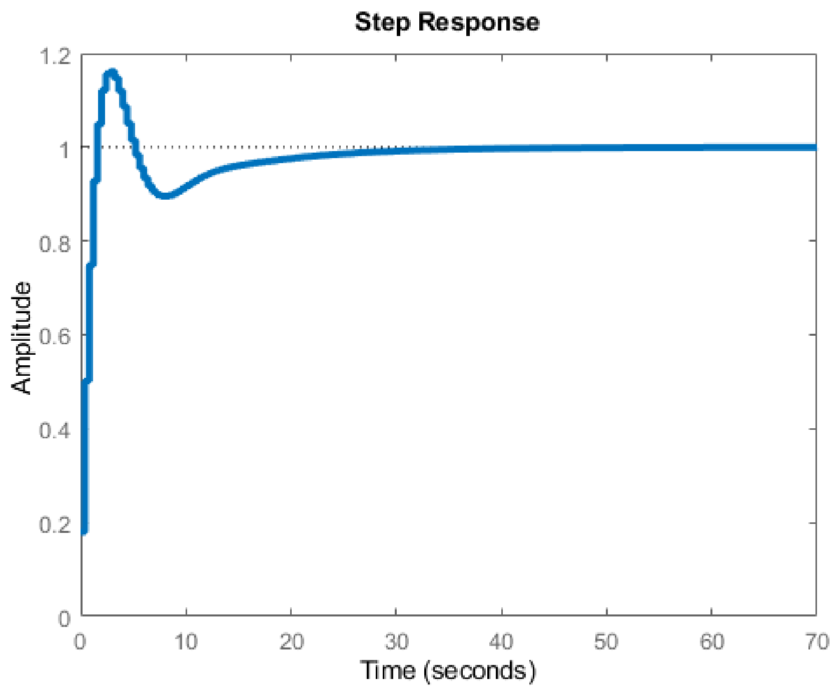


Wanna reTune it?

And after the “No” button is pressed, the controller gains along with the system get saved into a file.

6.2 MATLAB PID-Tuner:

If the “MATLAB” button in the first dialog box is clicked, then the system gets passed to the MATLAB’s pidtune function, and the digital PID controller will be designed as demonstrated below.



6.2 Intelligent PID Tuning:

A smart way of controller design is to use an intelligent method to demonstrate the system and decide for the controller gains. Here a genetics algorithm is used by the “ga” function offered by MATLAB optimization toolbox.

6.2.1 Cost Function:

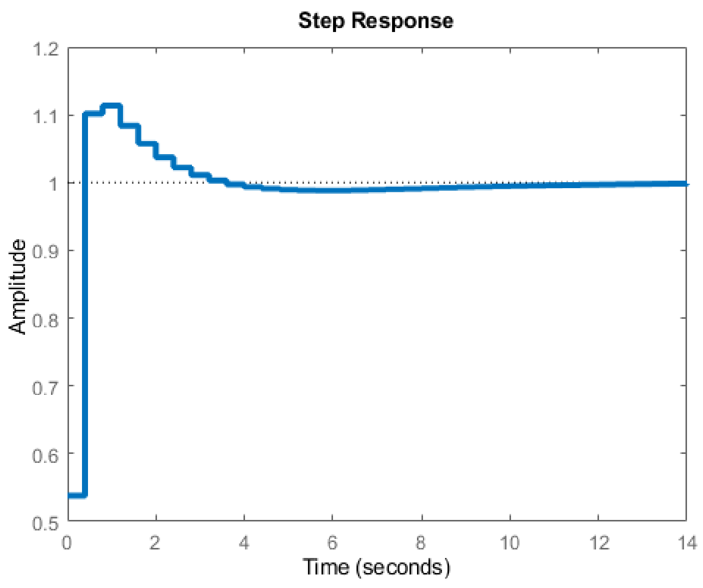
For the cost of the controller a weighted sum of the settling time and Overshoot percentage is calculated using MATLABs control toolbox and used as the cost of Tuned PID. The lower the cost, the better the control policy.

6.2.2 Genetic Algorithm Parameters:

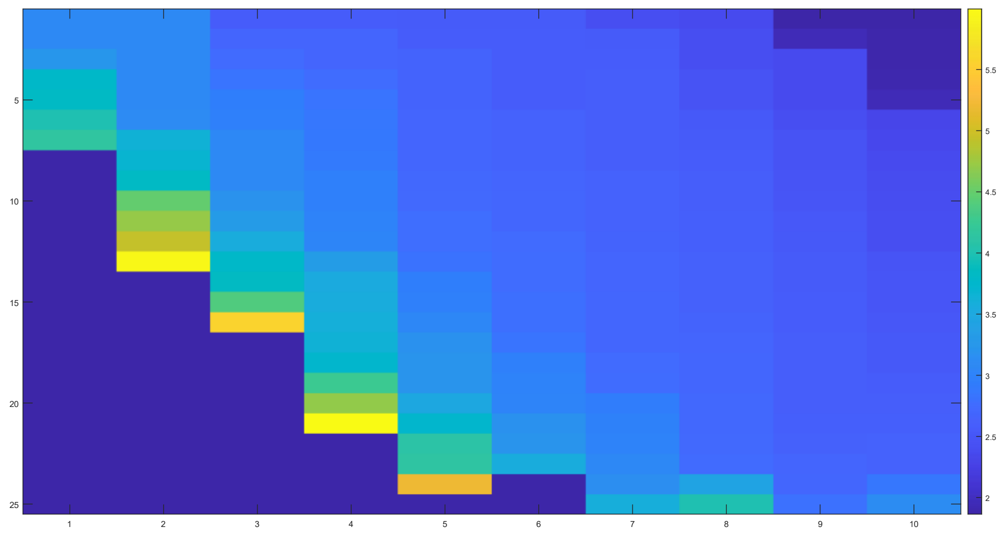
A modestly small population number of 25 is created and randomly initialized. The algorithm ran for 10 iterations.

6.2.3 Results:

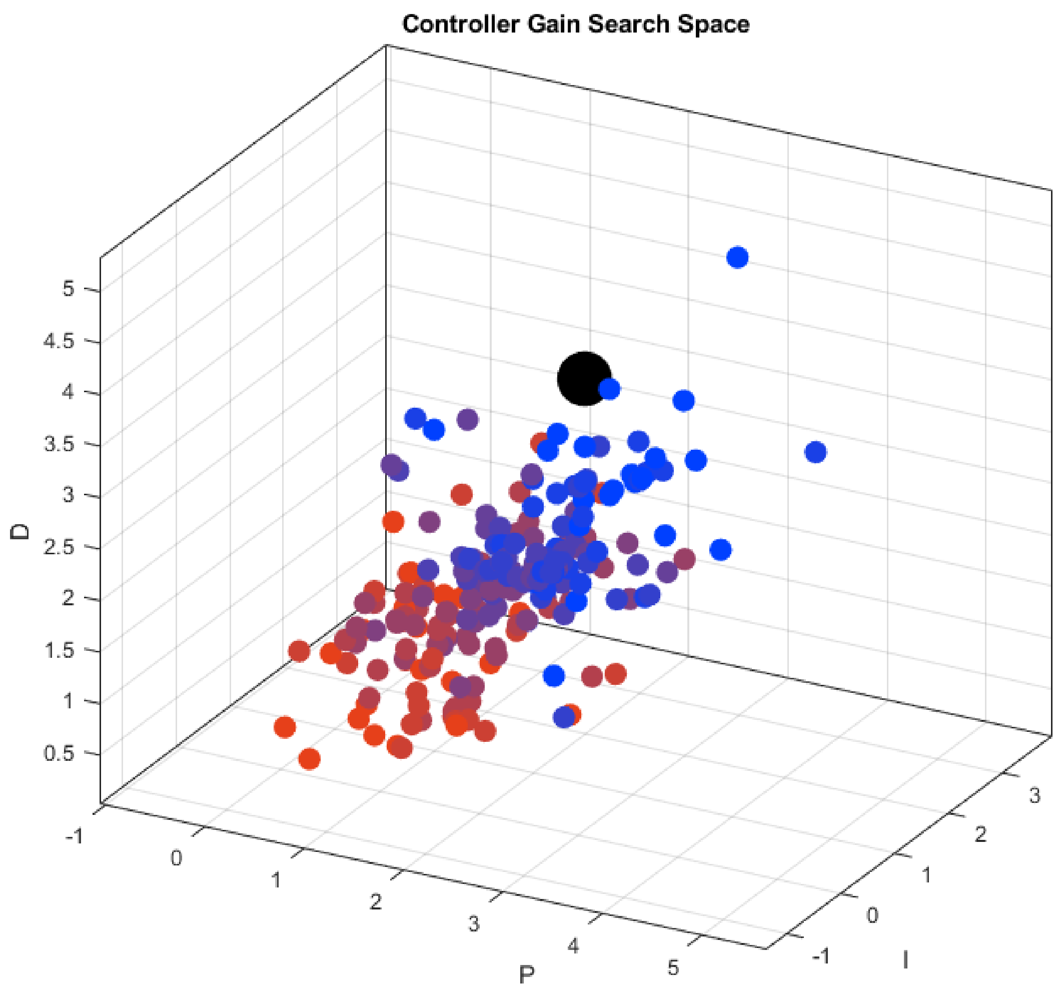
1. First, the best tuned controller is illustrated in the following plot with overshoot of **around 10 percent** and a **zero** steady state error.



2. The performance of the GA algorithm is shown in the heatmap sketching the individual costs by iteration. It can be clearly seen that the cost is getting significantly lower as the algorithm proceeds.



3. Finally the 3D plot of the individuals is sketched by cost. Note the colder the individual color gets the better the cost is. The black individual is the best one with the lowest cost.



Part 7: Conclusion

In the whole process, a simple mass spring damper system is analyzed. First for the Identification and then for the controller design using the 3 discussed methods. As shown, there are several methods in designing controllers along with the analytical methods which weren't discussed at all.

But, among all the possible ways, intelligent methods have the advantage of actively monitoring the plant and compensating for the controller parameters, which makes it more and more robust during the run-time.