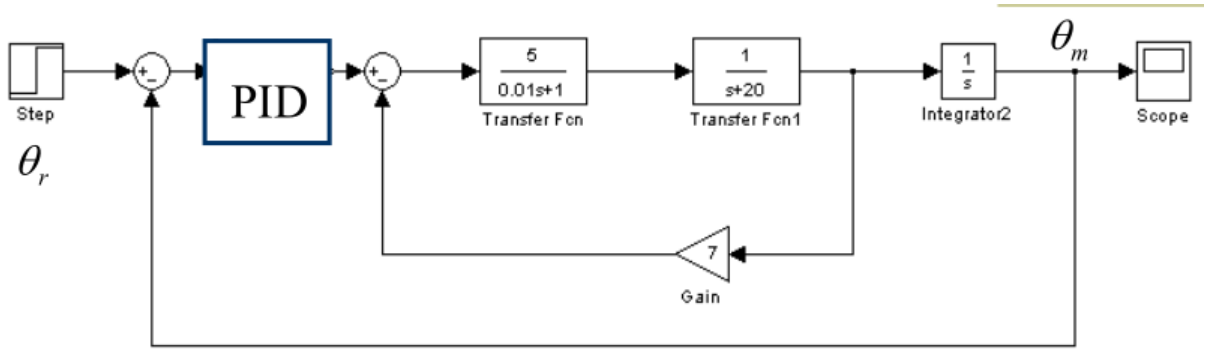
HOMEWORK ASSIGNMENT #1

Advisor:	**Prof. Shaw Jinsiang**
Student:	**NGUYEN CONG MINH**
ID'Student:	**103618040**

❖ Problem

Plant: DC Motor model

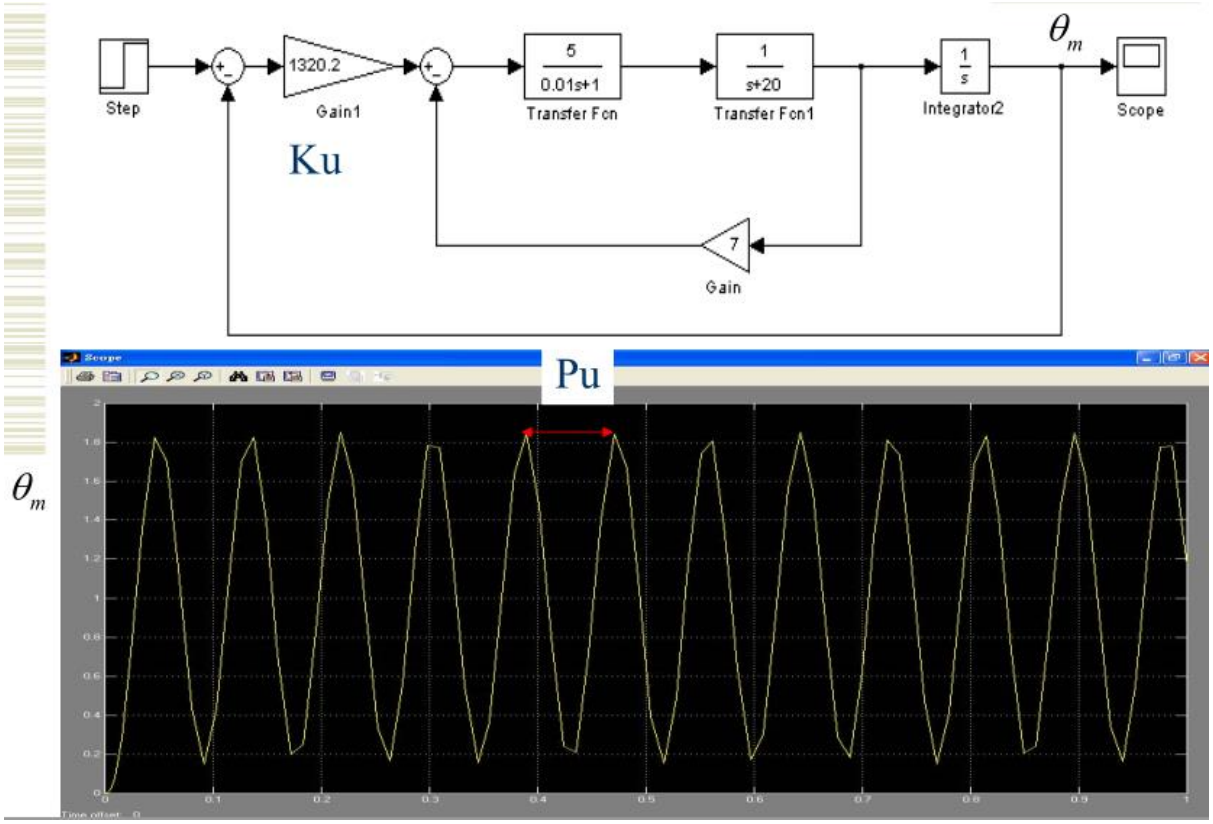


Search domain: $P \in [1 \quad 5000]$; $I \in [1000 \quad 80000]$; $D \in [1 \quad 1000]$

❖ Solution

1. PID tuning by Ziegler-Nichols method

PID tuning using Ziegler-Nichols method is based on the frequency response of the closed-loop system by determining the point of marginal stability under pure proportional gain (Ki and Kd equal 0) is creased until the system becomes marginally stable. At this point, the value of proportional gain known as the ultimate gain, Ku, together with its period of oscillation frequency so called the ultimate period, Pu, are recorded. Based on these values Ziegler-Nichols calculated the tuning parameters as the following:



From the chart above, we can determine Ku = 1320.2 and Pu = 0.085

$$K_p = 0.6K_u = 792.12$$

$$K_I = \frac{0.6K_u}{0.5P_u} = 18638$$

$$K_D = 0.6K_u \times 0.125P_u = 8.42$$

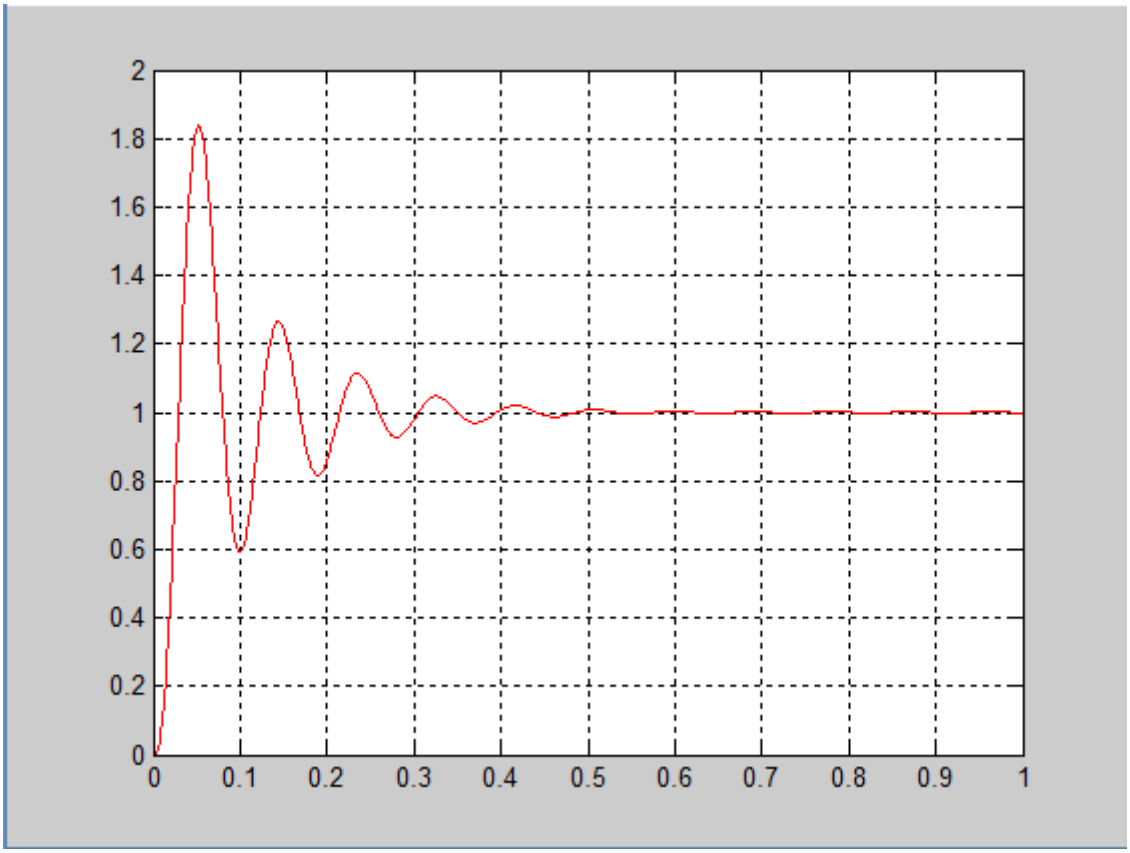


Fig 1. Step response of PID based on Ziegler-Nichols controller

2. GA tuning of PID Controller

The most important step in applying GA is to choose the objective functions that are used to evaluate fitness of each chromosome.

The selected fitness function:

$$fitness\ value = \frac{1}{error\ \ function}$$

Where *error function* is "Sum of absolute error" (SAE) and "Sum of square error" (SSE).

$$SAE = \sum_{k=0}^{N} |e(k)|$$

$$SSE = \sum_{k=0}^{N} e^2(k)$$

The SSE function has some advantages of producing smaller overshoot and oscillations than the SAE function.

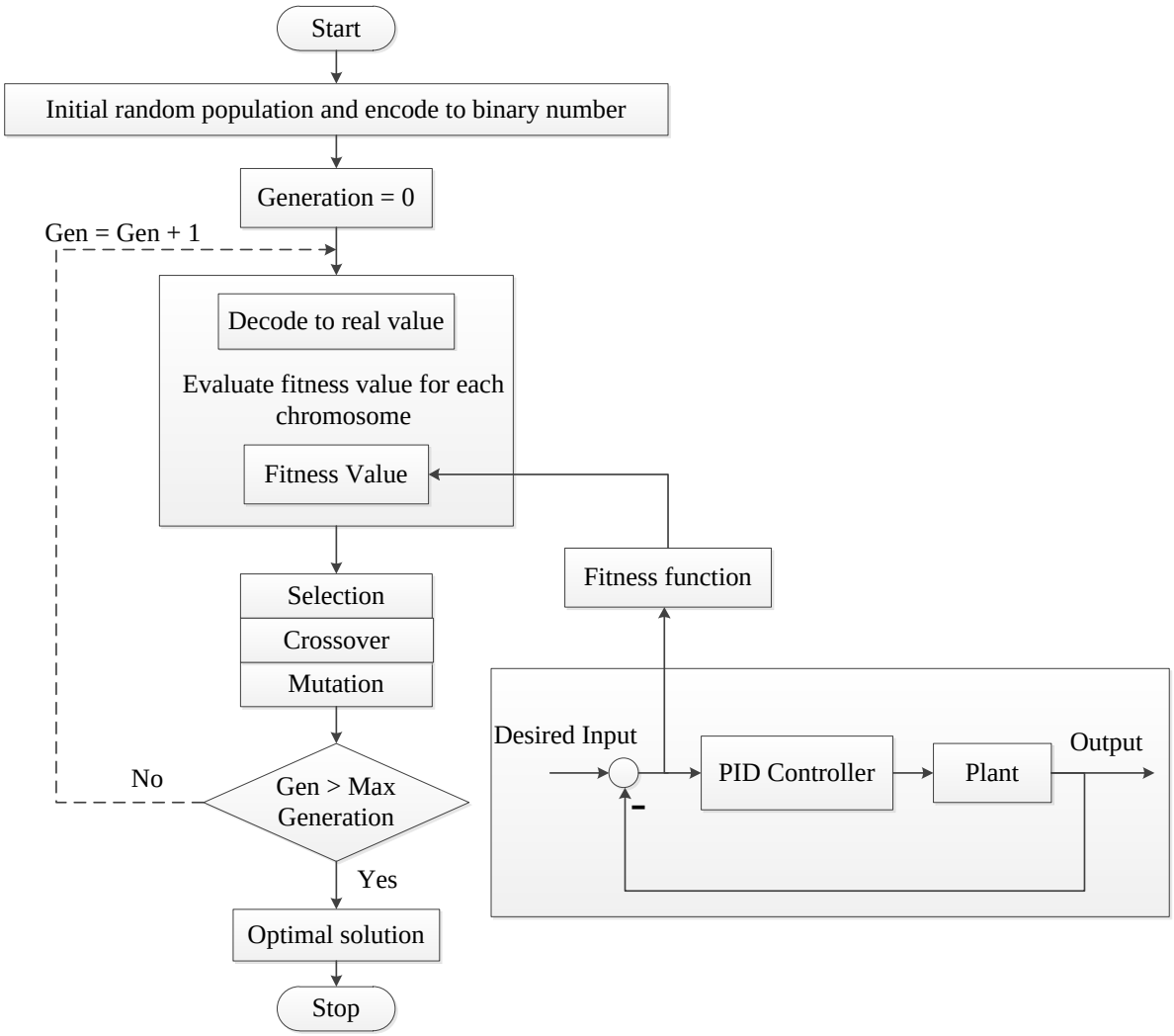


Fig 1. Flowchart of genetic algorithm for PID tuning

The figure 2 and 3 show the progress of the GA algorithm.

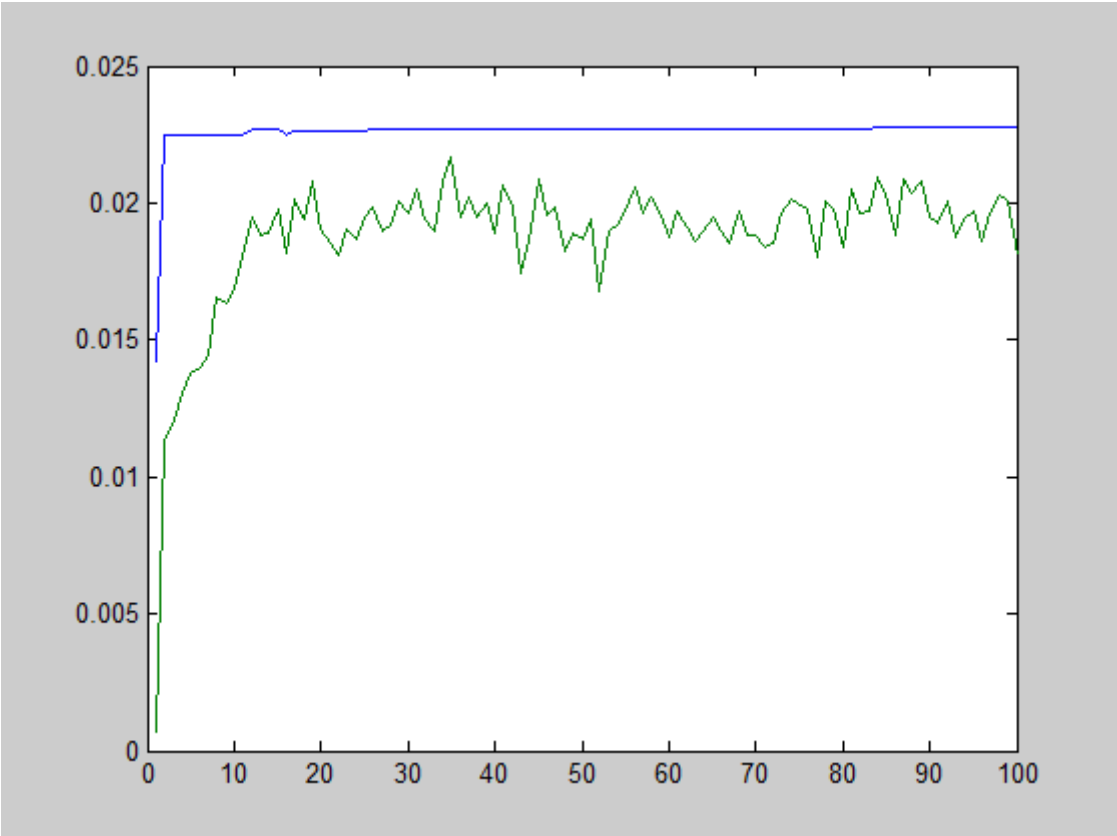


Fig 2. The performance curve when fitness function is SAE

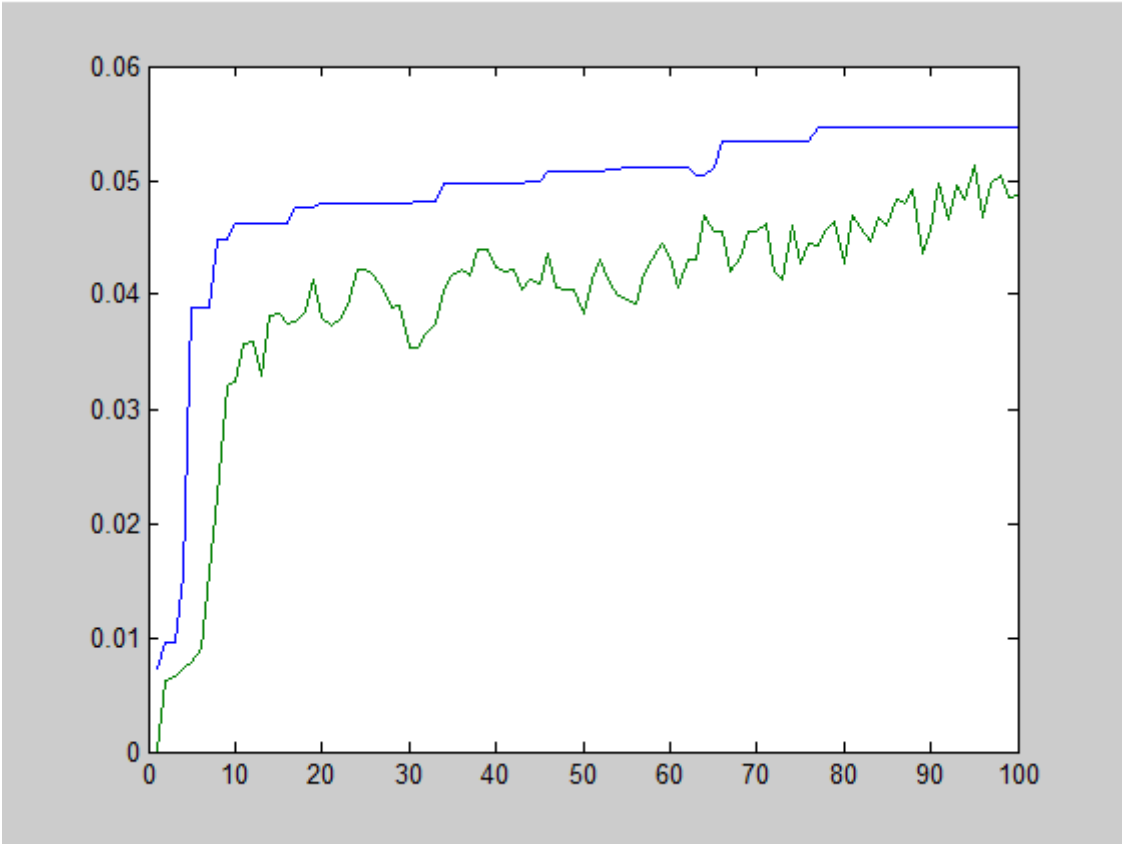


Fig 3. The performance curve when fitness function is SSE

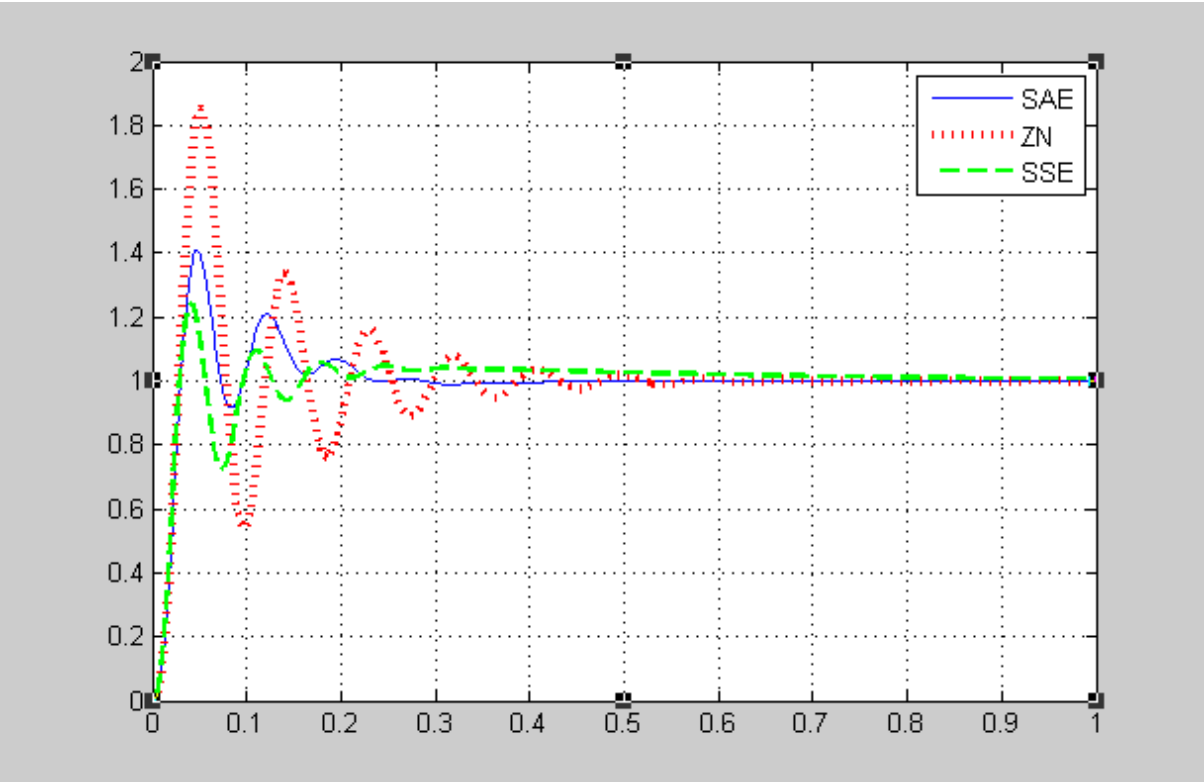


Fig 4. Responses of controllers (Z-N, GA-SAE and GA-SSE)

	Kp	Ki	Kd
Z-N (Ziegler-Nichols):	792.12	18638	8.42
SAE	547.80	7319	13.47
SSE	425.54	10193	19.11

3. GA Program (Matlab code)

popsize	gen	pcross	pmutation	chrom
50	100	0.9	0.01	20/17/17

This programme, I describe each chromosome by 3 genes, each gene has a different number of nucleotides(length). Then when do crossover, I cross the chromosome at multi-points. This method give better result than single cross point.

And, each gene has a different length. Why? Because the range of P(1st gene) is 1-5000, and the accuracy is 0.01 then I will have 500000 value in this range to check. So, I select resolution is 1048576. Therefore, length of this gene is 20bit.
Similar to other genes, I(2nd gene) has 80000 value, D(3rd gene) has 100000 value so they will need 17bit to describe.

I **choose the fitness function** is: $\frac{1}{SAE}$ or $\frac{1}{SSE}$ to evaluate generations.

More over, in the program, I still use a variable to contain the best value through the generations to save in to the end of process. This is the best result. Why? To avoid the situation that the next generation is worse than previous generation. In this situation, the best result will not be surely to save until the end of process. That is why we have to save in a storehouse than give it out at the end of process.

```
%%=========================================================================
MAIN PROGRAM
%%=========================================================================
% close all; clear all; clc;
startTime = clock;

%% Initialize parameters
InitializeParameters;

%% Initialize population
InitializePopulation;
maximumFitness = -100;
chromosome(chromosomeProperties.populationNumber+1).fitness = -100;

%% Generate generations
for generationNo = 1:chromosomeProperties.generation
% for i = 1:10
    %% decode chromosomes then caculate fitness function
    for individualNo = 1:chromosomeProperties.populationNumber
        % decode chromosome
        for geneNo = 1:chromosomeProperties.geneNumber
            % decode [0 1 ...] array into a value in range [0 2^resolution-
1] (relative value)
            relativeValue =
decodeChromosome(chromosome(individualNo).gene(geneNo).code,
chromosomeProperties.gene(geneNo).decodeCode);
            % decode ralative value into real value
            chromosome(individualNo).gene(geneNo).value =
decodeResolution(relativeValue,
chromosomeProperties.gene(geneNo).resolution,
chromosomeProperties.gene(geneNo).startRange,
chromosomeProperties.gene(geneNo).endRange);
        end
        % caculate fitness function
        gainP = chromosome(individualNo).gene(1).value;
        gainI = chromosome(individualNo).gene(2).value;
        gainD = chromosome(individualNo).gene(3).value;
        P = gainP;
        I = gainI;
        D = gainD;
        sim('motor1');
        Error = Error1.signals.values;

        T = 1:1:1000;
        SSE = sum(Error.*Error);
        STSE = sum(T'.*SSE);
        SAE = sum(abs(Error));
        STAE = sum(T'.*SAE);
```

```
        fitness = 1/SSE;
%         fitness = 1/SAE;

        chromosome(individualNo).fitness = fitness;
        if chromosome(individualNo).fitness >
chromosome(chromosomeProperties.populationNumber+1).fitness
            chromosome(chromosomeProperties.populationNumber+1) =
chromosome(individualNo);
        end
    end % for individualNo...
    % caculate sum of fitness and mean of fitness; find the chromosome
having max fitness
    [sumFitness, meanFitness, maxValue, maxPosition] =
measureFitness(chromosomeProperties, chromosome);

    %% reproduction (selection)
    for individualNo = 1:chromosomeProperties.populationNumber
        % random to wheel roulette to select a parent chromosome
        k = wheelRoulette(chromosome, sumFitness);
        % select chromosome
        selectedChromosome(individualNo) = chromosome(k);
    end % for individualNo...
    selectedChromosome(1) = chromosome(maxPosition); % elite preservation

    %% crossover
    % browse each chromosome pair
    for individualNo = 1:2:chromosomeProperties.populationNumber
        % browse each gene
        for geneNo = 1:chromosomeProperties.geneNumber
            if rand <= crossProperties.percentCross % if crossover occurs
on this gene of this chromosome
                % random a cross position
                lengthGene = chromosomeProperties.gene(geneNo).resolution;
                crossPosition = ceil(rand*(lengthGene - 1));
                if crossPosition == 0
                    crossPosition = 1;
                end
                % process crossover 2 gene beggining from crossPoint
                [selectedChromosome(individualNo).gene(geneNo),
selectedChromosome(individualNo+1).gene(geneNo)] =
crossGene(selectedChromosome(individualNo).gene(geneNo),
selectedChromosome(individualNo+1).gene(geneNo), crossPosition,
chromosomeProperties.gene(geneNo).resolution);
            end % if rand...
        end % for geneNo...
    end % for individualNo...

    %% mutation
    % browse each chromosome to mutate
    for individualNo = 1:chromosomeProperties.populationNumber
        % browse each gene to mutate
        for geneNo = 1:chromosomeProperties.geneNumber
            % browse each nucleotide to mutate
            for nucleotideNo =
1:chromosomeProperties.gene(geneNo).resolution
                if rand <= crossProperties.percentMutation
                    % reverse bit to mutate

selectedChromosome(individualNo).gene(geneNo).code(nucleotideNo) = 1 -
selectedChromosome(individualNo).gene(geneNo).code(nucleotideNo);
```

```
                end
            end % for nucleotideNo...
        end % for geneNo...
    end % for individualNo...
    % end of mutation

    %% select population for next generation
    chromosome(1:chromosomeProperties.populationNumber) =
selectedChromosome;

    %% save tracking curves
    result = [result; generationNo, maxValue, meanFitness];
end
figure(1);
plot([result(2:chromosomeProperties.generation+1, 2),
result(2:chromosomeProperties.generation+1, 3)]);

P = chromosome(chromosomeProperties.populationNumber+1).gene(1).value;
I = chromosome(chromosomeProperties.populationNumber+1).gene(2).value;
D = chromosome(chromosomeProperties.populationNumber+1).gene(3).value;

elapsedTime = etime(clock, startTime)


%%=========================================================================
%% InitializeParameters;
%% Initialize parameters
crossProperties.percentCross = 0.9;
crossProperties.percentMutation = 0.01;

% define number of genes each chromosome
chromosomeProperties.generation = 100;
% define number of genes each chromosome
chromosomeProperties.geneNumber = 3;
% define number of population
chromosomeProperties.populationNumber = 50;
%[524288 262144 131072 65536 32768 16384 8192 4096 2048 1024 512 256 128 64
32  16   8   4   2   1]
% 1st gene properties
% [1 5000]
% [600 1000]
chromosomeProperties.gene(1).startRange = 1;
chromosomeProperties.gene(1).endRange = 5000;
chromosomeProperties.gene(1).resolution = 20;
chromosomeProperties.gene(1).decodeCode = [524288 262144 131072 65536 32768
16384 8192 4096 2048 1024 512 256 128 64 32  16   8   4   2   1];
% 2nd gene properties
% [1000 80000]
chromosomeProperties.gene(2).startRange = 1000;
chromosomeProperties.gene(2).endRange = 80000;
chromosomeProperties.gene(2).resolution = 17;
chromosomeProperties.gene(2).decodeCode = [65536 32768 16384 8192 4096 2048
1024 512 256 128 64 32  16   8   4   2   1];
% 3rd gene properties
% [1 1000]
% [1 50]
chromosomeProperties.gene(3).startRange = 1;
chromosomeProperties.gene(3).endRange = 200;
chromosomeProperties.gene(3).resolution = 17;
```

```
chromosomeProperties.gene(3).decodeCode = [65536 32768 16384 8192 4096 2048
1024 512 256 128 64 32  16   8   4   2   1];

result = [0 0 0];
%%=========================================================================

%%=========================================================================
%% InitializeParameters;
%%=========================================================================

%%=========================================================================
%% InitializePopulation;
%% Initialize population
% browse sequently the chromosomes
for i = 1:chromosomeProperties.populationNumber
    % browse sequently the genes
    for j = 1:chromosomeProperties.geneNumber
        % initialize the gene (random then ceil)
        chromosome(i).gene(j).code = ceil(rands(1,
chromosomeProperties.gene(j).resolution));
    end
end
%%=========================================================================

%%=========================================================================
function value = decodeChromosome(chromosome, decodeCode)
    value = chromosome * decodeCode';
return
%%=========================================================================

%%=========================================================================
%%=========================================================================
function value = decodeResolution(relativeValue, resolution, startRange,
endRange)
    value = relativeValue/(2^resolution - 1)*(endRange - startRange) +
startRange;
return
%%=========================================================================

%%=========================================================================
function [sumFitness, meanFitness, maxValue, maxPosition] =
measureFitness(chromosomeProperties, chromosome)
% caculate sum of fitness and mean of fitness; find the chromosome having
max fitness
    sumFitness = 0;
    maxValue = chromosome(1).fitness;
    maxPosition = 1;
    for j = 1:chromosomeProperties.populationNumber
        sumFitness = sumFitness + chromosome(j).fitness;
        if maxValue < chromosome(j).fitness
            maxValue = chromosome(j).fitness;
            maxPosition = j;
        end
    end
    meanFitness = sumFitness/chromosomeProperties.populationNumber;
return;
%%=========================================================================

%%=========================================================================
function k = wheelRoulette(chromosome, sumFitness)
```

```
    % random a roulette value
    randValue = rand*sumFitness;
    % wheel the roulette to select chromosome
    partSum = chromosome(1).fitness;
    k = 1;
    while partSum < randValue
        k = k + 1;
        partSum = partSum + chromosome(k).fitness;
    end % while
return;
%%=========================================================================

%%=========================================================================
function [childGene1, childGene2] = crossGene(fatherGene, motherGene,
crossPosition, geneLength)
    childGene1 = fatherGene;
    childGene2 = motherGene;
    for nucleotideNo = crossPosition+1:geneLength
        childGene1.code(nucleotideNo) = motherGene.code(nucleotideNo);
        childGene1.code(nucleotideNo) = fatherGene.code(nucleotideNo);
    end % for m...
return;
%%=========================================================================
```

4. Comments and Suggestions

- The average values of GA performance indices, as expected, are always smaller than its corresponding Ziegler-Nichols (Percent Overshoot, Rise Time, and Settling Time). We see that, with SAE algorithm, the system is absolutely stable very fast, but the SSE algorithm make the system has smaller overshoot.

- According results, We see that, sometime the next generation can be worse than previous generation. So I suggest that, we should keep some best parents from previous generation to the next generation. By this way, we will be sure that next generation will never be worse than previous generation. Because, some best individuals may be immortal.