**Exercise 4.1**

I used for loops to calculate and plot poles once at a time, the matlab script as below:

clear

clc

num = {[1]; [-1,1]; [1]; [1]; [1]; [1] ; [1]; [-1] ;[1] ;[1] ;[1] }; % numerator matrix of all the transfer funtions in Exercise 4.2

den = {[1,1];[1,1]; [-1,1]; [1,0,-1]; [1,0]; [1,0,0,0]; [1,1]; [1,1] ;[1,1,1] ;[1,0,1] ;[1,1,0]}; % denominator matrix of all the transfer funtions in Exercise 4.2

[m,n] = size(num); % determine number of transfer funtions

Calculated\_Poles = {}; %empty cell to store poles of transfer functions

for i= 1:m

% determine transfer funtions of corresponding exercise

if i~=7

H = tf(num(i,:),den(i,:));

else

H = tf(num(i,:),den(i,:),'InputDelay',1); %exercise 7 has a time delay

end

Pole = pole(H);

Calculated\_Poles = [Calculated\_Poles; {Pole}];

%Calculated\_Poles = sprintf('The pole for exercise %d is %d.', i, Pole)

%plot poles

figure(); hold on

pzmap(H)

title(['Zero pole plt for exercise' num2str(i)]);

hold off

end

The poles calculated by matlab for transfer functions in exercise 4.2 are, it is consistent with manual calculations:

-1 Asymptotically stable

-1 Asymptotically stable

1 Unstable

[-1;1] Unstable

0 Marginally stable

[0;0;0] Unstable

-1 Asymptotically stable

-1 Asymptotically stable

[-0,5 + 0,9i;-0,5 - 0,9i] Asymptotically stable

[0,0 + 1,0i;0,0 - 1,0i] Marginally stable

[0;-1] Marginally stable

**Exercise 4.2**

%system matrix

A=[0,1;0,-2];

B=[0;1];

C=[0,0];

D=[0];

%creat state-space model

ss1=ss(A,B,C,D);

%calculate eigenvalues of matrix A, to analyze stability of state space model

disp('eigenvalues of matrix A')

e = eig(A)

%initial state response of the state space model

x0 = [45 ; 28]; %assume some non-zero initial state

initial(ss1,x0)

%Find the transfer function

disp('corresponding transfer function')

H = tf(ss1)

disp('poles of transfer function')

pole(H)

It is a Marginally stable system

**Exercise 4.3**

clear

clc

s = tf('s');

P = 1/(s^3+2\*s^2+s); %transfer function of the plant P

%define the Pi controller

Kp = 1;

Ti = 300;

Ki = Kp/Ti;

C = pid(Kp,Ki);

%use feedback function to find transfer function model of the control system

T = feedback(C\*P,1);

figure (1)

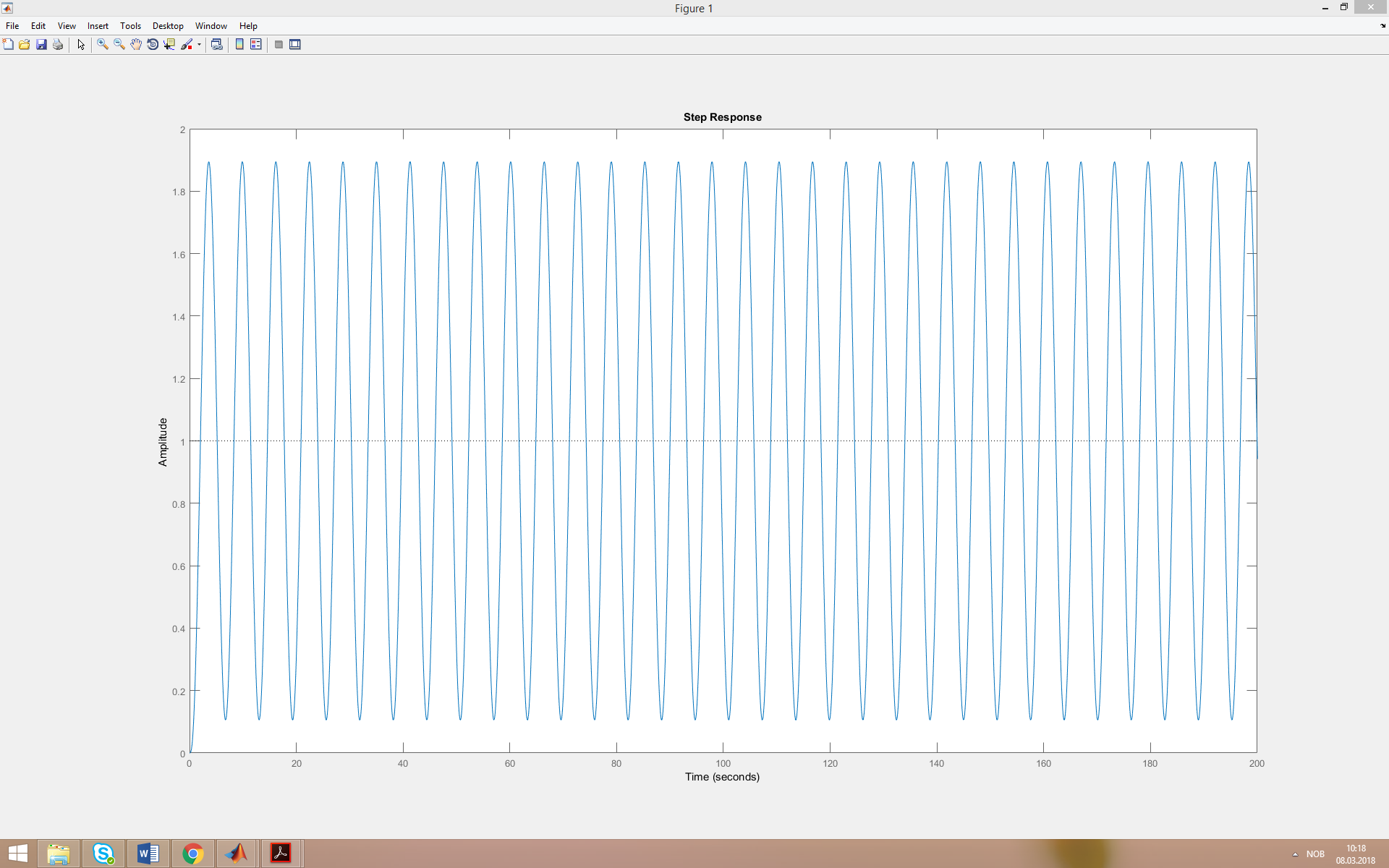
step(T)

figure (2)

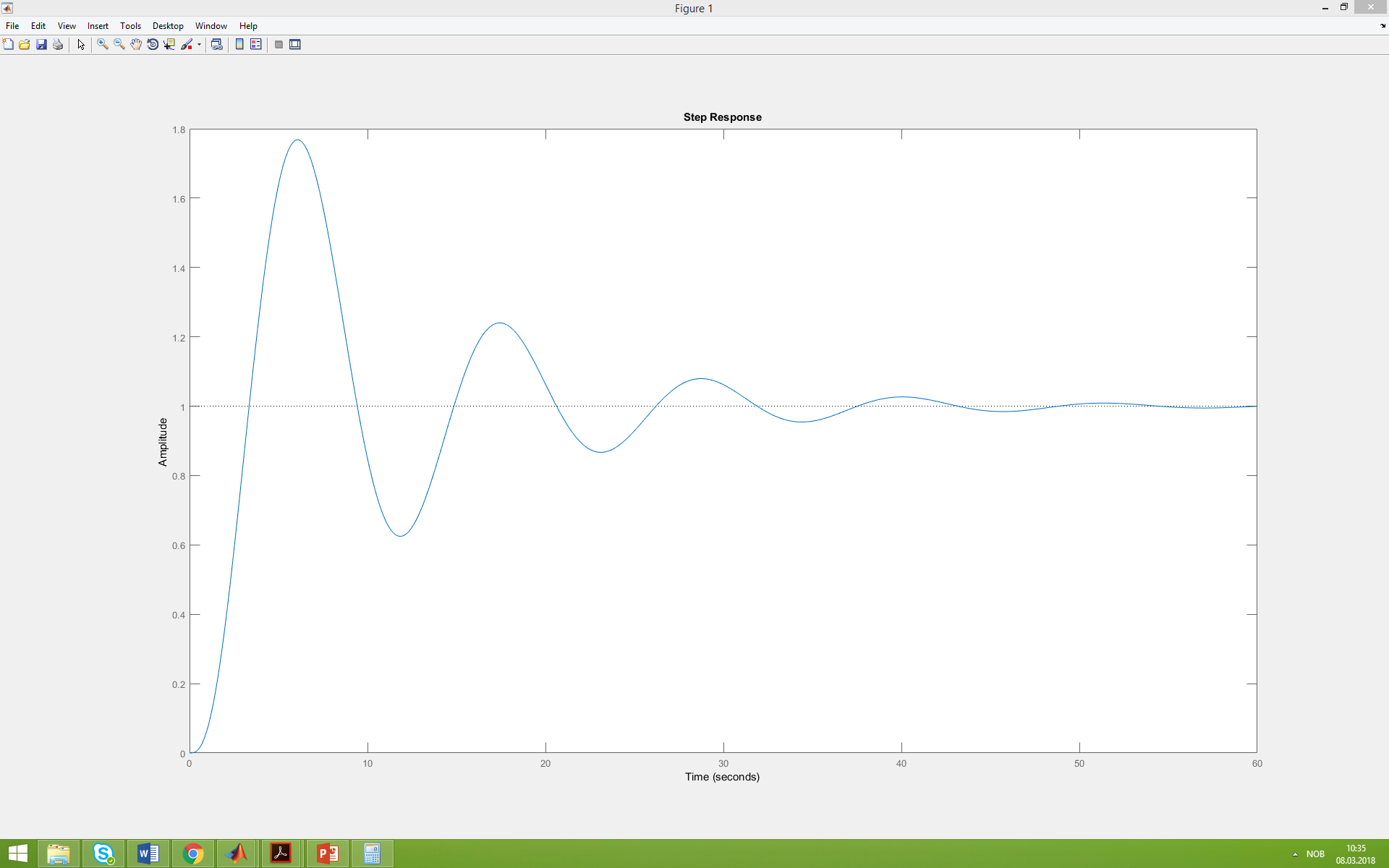
pzmap(T)

Tuning, relaxed ZN method

When P=2, we found the oscillation with period around Pu=6

So the tuned Kp = 0.32\*2 = 0.64

Ti = Pu = 6



We see the response fulfill ¼ decay rule

**Exercise 4.4**

clear

clc

%system matrix

A=[0,1;0,-2];

B=[0;1];

C=[0,0];

D=[0];

Ts=0.1;%Sampling time

model\_cont=ss(A,B,C,D);

model\_disc=c2d(model\_cont,Ts,'zoh');

A=model\_disc.a

%eigenvalues of matrix A

disp('eigenvalues of matrix A')

e = eig(A)

It is a Marginally stable system

**Exercise 4.5**

Used the following code

We need to calculate the following partial derivative：

Matrix A is [df1/dh1, df1/dh2; df2/dh1, df2/dh2]

Matrix B is [df1/du1, df1/du2; df2/du1, df2/du2]

Matrix B is [dy1/dh1, dy1/dh2; dy2/dh1, dy2/dh2]

Matrix B is [dy1/du1, dy1/du2; dy2/du1, dy2/du2]

clear all

clc

syms A1 Kp u1 Kv1 rho g h1 G A2 Kv2 u2 h2

f1 = (Kp/A1) \* u1 - ((Kv1/A1)\*((rho\*g)/G)^0.5)\*h1^0.5;

f2 = ((Kv1/A2)\*((rho\*g)/G)^0.5)\*h1^0.5 - ((Kv2/A2)\*((rho\*g)/G)^0.5)\*u2\*h2^0.5;

y1 = h1;

y2 = h2;

A = jacobian([f1;f2],[h1 h2])

B = jacobian([f1;f2],[u1 u2])

C = jacobian([y1;22],[h1 h2])

D = jacobian([y1;22],[u1 u2])

**Exercise 4.6, optimization 1 grid search**

I changed the following part of code to solve the present problem

In the initialization part, I changed the boundary of serach and initial guess

%Initialization:

x1\_min=-5;x1\_max=5;N\_x1=100;

x1\_array=linspace(x1\_min,x1\_max,N\_x1);

x2\_min=-5;x2\_max=5;N\_x2=100;

x2\_array=linspace(x2\_min,x2\_max,N\_x2);

f\_min=inf;

x1\_opt=-1.9; %initial guess

x2\_opt=2; %initial guess

Objective function also changed accordingly

%Objective function:

f=100\*(x2-x1^2)^2+(1-x1)^2;

**Exercise 4.7, optimization 1 fmincon**

Please check the matlab code

I didn’t use local function, I changed objective function to

fun = @(x)(100\*(x(2) - x(1)^2)^2 + (1 - x(1))^2);

**Exercise 4.8, optimization 1 Newton search**

Please check the matlab code

The gradient and Hessian should be calculated according to objective function

%Gradient:

G\_k=[400\*x1\_k^3-400\*x2\_k\*x1\_k+2\*x1\_k-2;

200\*(x2\_k - x1\_k^2)];

%Hessian:

H\_k=[1200\*x1\_k^2 - 400\*x2\_k + 2 , - 400\*x1\_k;

- 400\*x1\_k, 200];

This method also influenced a lot by initial location

**Exercise 4.9, optimization 4 grid search with constraints**

I changed the constraint in the following code

%Constraint:

if x2 >= x1+1,

f=inf;

end

**Exercise 4.10, optimization 5 fmincon, with constraints**

Please check the matlab code

In file ‘fun\_constraints.m’, I used the following code to define constraints

function [cineq,ceq]=fun\_constraints(x)

x1=x(1);

x2=x(2);

cineq = []; % Compute nonlinear inequalities.

x2 >= x1+2; %Inequality constraint

cineq = -x2+x1+2; % Compute nonlinear inequalities.

ceq = []; % Compute nonlinear equalities.

end