EE 362K
Homework 2 Solutions

Q1

5 points
Credit for a signed statement as per the instructions in the homework

Q2

7 points
First graph: 2 points for modification in code, 1 point for graph
Second graph: 2 points for modifying code, 2 points for graph

Code (with modifications included):

```
%% Q2

close all
clear all

%% Modification to plot population that is alive

z(1,1) = 10000; %initial uninfected pop
z(2,1) = 10; %initial infected pop
z(3,1) = 0; %initial immunized/cured
z(4,1) = 0; %dead

a=1; b=10; c=1; %#/100/day die, infected, immunized
d=50; e=25; %#/100/day of infected who die or are cured
f=1; %#/100/day of immune who die
u(1) = 10; u(2) = 10; %#/uninfected and infected added per day.

A = [ -a-b-c 0 0 0; b -d-e 0 0; c e -f 0; a d f 0; ]./100;
B = [ 1 0; 0 1; 0 0; 0 0 ];
C = [ eye(3) zeros(3,1) ];

day =1:200;
for t=1:length(day)-1
    z(:,t+1)= z(:,t) + A*z(:,t)+B*u';

    for j=1:4
        if z(j,t+1) < 0
            z(j,t+1) = 0;
        end
    end
end

plot(day,[C*z; sum(z(1:3,:),1)]) % alive population = z1+z2+z3 (could also modify C)
legend({'uninfected','infected','immune' 'alive'}, 'Fontsize',12);
xlabel('Days')
ylabel('Number')
```

```
%% Graphs of total population for different immunization rates
% The immunization rate is absolute, not a percentage here.

figure;
hold on;
c_arr = [1,10,100,1000];
alive = [];
for i = 1:length(c_arr)

    c = c_arr(i);
    z(1,1) = 10000; %initial uninfected pop
    z(2,1) = 10; %initial infected pop
    z(3,1) = 0; %initial immunized/cured
    z(4,1) = 0; %dead

    A = [ -a-b 0 0 0; b -d-e 0 0; 0 e -f 0; a d f 0; ]./100;

    day =1:200;
    for t=1:length(day)-1
        z(:,t+1)= z(:,t) + A*z(:,t)+B*u'+min(c_arr(i),z(1,t))*[-1 0 1 0]';  % last term for
            immunization

%           if i==4
%               disp(min(c_arr(i),z(1,t)));
%               disp z
        for j=1:4
            if z(j,t+1) < 0
                z(j,t+1) = 0;
            end
        end
    end

    alive = [alive; sum(z(1:3,:),1)]; % alive population = z1+z2+z3
end

plot(day,alive)
legend({'1','10','100','1000'}, 'Fontsize',12);
xlabel('Days')
ylabel('Number (log scale)')
title('Alive population for different immunization rates')
```

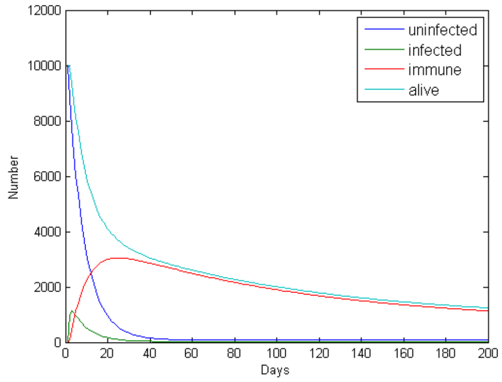

Figure 1: Graph for total population that is alive

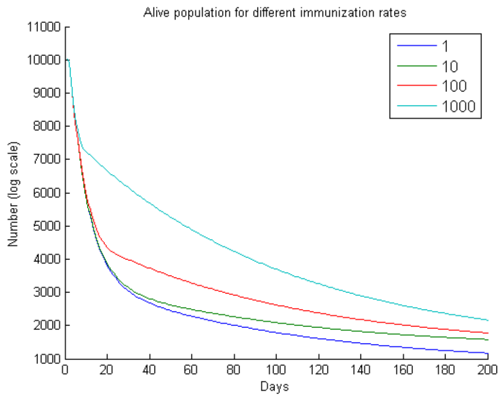

Figure 2: Population that is alive for different absolute immunization rates

Q3

20 points 5 points each for (a), (b) and (c), 1 point for plot of each numerical solution, 1 point for labels

Code for generating numerical solutions and plot:

```
%% Q3

clear all
close all

%% Euler's method, step = 0.5
h_e = 0.5;
t_e = 0:h_e:2;
x_e = zeros(length(t_e),1);
x_e(1)=2;
for i = 2:length(t_e)
    f = q3slope(t_e(i-1),x_e(i-1));
    x_e(i) = x_e(i-1)+f*h_e;
end

%% Heun's method, step = 1.0
h_h = 1;
t_h = 0:h_h:2;
x_h = zeros(length(t_h),1);
x_h(1)=2;
for i = 2:length(t_h)
    f_begin = q3slope(t_h(i-1),x_h(i-1));
    x_tmp = x_h(i-1)+h_h*f_begin;
    f_end = q3slope(t_h(i),x_tmp);
    f_av = 0.5*(f_begin+f_end);
    x_h(i) = x_h(i-1)+h_h*f_av;
end

%% 4th order Ranga-Kutta method, step = 2.0
h_rk = 2;
t_rk = 0:h_rk:2;
x_rk = zeros(length(t_rk),1);
x_rk(1)=2;
for i = 2:length(t_rk)
    k1 = q3slope(t_rk(i-1),x_rk(i-1));
    x_tmp = x_rk(i-1)+0.5*h_rk*k1;
    k2 = q3slope(t_rk(i-1)+0.5*h_rk,x_tmp);
    x_tmp = x_rk(i-1)+0.5*h_rk*k2;
    k3 = q3slope(t_rk(i-1)+0.5*h_rk,x_tmp);
    x_tmp = x_rk(i-1)+h_rk*k3;
    k4 = q3slope(t_rk(i-1)+h_rk,x_tmp);
    x_rk(i) = x_rk(i-1)+h_rk*1/6*(k1+2*k2+2*k3+k4);
end

%% Plotting analytical solution
t_a = 0:0.1:2;
x_a = 4/1.3*(exp(0.8*t_a)-exp(-0.5*t_a))+2*exp(-0.5*t_a);

plot(t_e,x_e,t_h,x_h,t_rk,x_rk,t_a,x_a);
legend({'Euler','Heun','Ranga-Kutta','analytical'},'Location','southeast','Fontsize',12);
xlabel('t (in s)')
ylabel('x')
```

Function to compute derivative:

```
%% function to compute dx/dt for given x and t for Q3

function f = q3slope(t,x)
    f = 4*exp(0.8*t)-0.5*x;
end
```

(a) Euler's method

t	0	0.5	1	1.5	2
x	2	3.5	5.6086	8.6576	13.1334

(b) Heun's method

t	0	1	2
x	2	6.7011	16.3198

(a) 4th order Ranga-Kutta method

t	0	2
x	2	15.1058

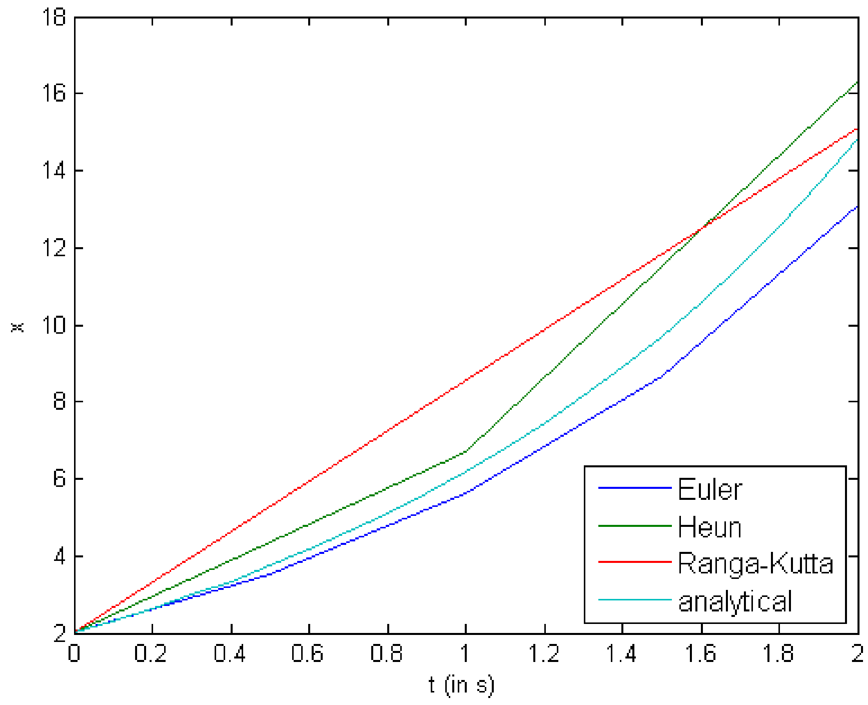

Figure 3: Plots of numerical solutions and analytical solution

Q4

12 points 5 points for solution using Ranga-Kutta method, 5 points for analytical solution, 2 points for absolute error

4th Ranga-Kutta method:

$$
\begin{aligned}
k_1 &= 1 \\
k_2 &= 3.375 \\
k_3 &= 5.14 \\
k_4 &= 13.0255 \\
\implies y(2.5) &= 1 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\
&= 13.940
\end{aligned}
$$

Analytical solution:

$$\frac{dt}{dt} = (1+t)\sqrt{y}$$
$$\int_1^y \frac{dy}{\sqrt{y}} = \int_0^t (1+t)dt$$
$$2\sqrt{y} - 2 = t + \frac{t^2}{2}$$
$$y = (1 + \frac{t}{2} + \frac{t^2}{4})^2 \implies y(2.5) = 14.535$$

Absolute error $= 14.535 - 13.9398 = 0.595$

Q5

14 points 3 points for state space representation, 4 points for 1-step Ranga-Kutta, 7 points for 4-step Ranga Kutta

The states are taken to be $z_1 = x, z_2 = \dot{x}, z_3 = \ddot{x}$. Then, the state space representation is as follows

$$\frac{d\mathbf{z}}{dt}\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -4 & -\frac{1}{2} & -3 \end{bmatrix}\mathbf{z} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u$$

Code to compute states at different times:

```
%% Q5

clear all
close all

A = [0 1 0; 0 0 1; -4 -1/2 -3];
step_num = [1 4];    % Number of steps

for s = 1:length(step_num)
    h = 4/step_num(s);  % Step size
    t_rk = 0:h:4;
    z_rk = zeros(3,length(t_rk));
    z_rk(:,1) = ones(3,1);  % Initialization of states

    % Fourth order Ranga-Kutta method
    for i = 2:length(t_rk)
        k1 = q5slope(A,z_rk(:,i-1));
        z_tmp = z_rk(:,i-1)+0.5*h*k1;
        k2 = q5slope(A,z_tmp);
        z_tmp = z_rk(:,i-1)+0.5*h*k2;
        k3 = q5slope(A,z_tmp);
        z_tmp = z_rk(:,i-1)+h*k3;
        k4 = q5slope(A,z_tmp);
        z_rk(:,i) = z_rk(:,i-1)+h*1/6*(k1+2*k2+2*k3+k4);
    end

    fprintf('For %d steps, the states at t=4 is:',step_num(s))
    disp(z_rk(:,end))
    fprintf('\n The value of x at t=4 is %.3f\n', z_rk(1,end))
end
```

Code to find state derivative:

```
%% function to compute dz/dt for given z, A, B for Q5
% z is a state (column vector)
% dz/dt = A*z + B*u

function f = q5slope(A,z,B,u)
    switch nargin
        case 2
            f = A*z;
        case 4
            f = A*z + B*u;
        otherwise
            error('Give either 2 or 4 arguments to q5slope')
    end
end
```

For 1 step RK, the state values are:

t	0	4
z_1	1	125
z_2	1	-441.7
z_3	1	1496.3

For 4 step RK, the state values are:

t	0	1	2	3	4
z_1	1	2	1.2405	-0.3079	-0.0939
z_2	1	-1.0104	-3.7751	5.4528	-6.3844
z_3	1	1.1146	4.8725	12.9685	24.9724

The result from the 4-step Ranga-Kutta can be seen to be drastically different from the 1-step Ranga-Kutta solution. Clearly, a single iteration of the Ranga-Kutta method does not give an accurate numerical solution for this ODE.

Q6

19 points 2 points for using ODE45 correctly, 1 point for each plot (8 points), 1 point for labels, 4 points for estimating natural frequency, 2 points for discussion on effect of modifying other system parameters

The code for using ODE45 and plotting the required graphs is shown below.

```
%% Q6

clear all
close all

global A;

%% Plots of position and velocity
k_arr = [2 4];  % values that k can take
t_arr = [1 5 30 60];    %Times for plots

for i = 1:length(k_arr)
    k = k_arr(i); c = 0.25; m = 4;     % b in question = c in slides
    A = [0 1; -k/m -c/m];

    figure(i);
    s = suptitle(['k = ' num2str(k_arr(i))]);
    set(s,'FontSize',10)
    for j = 1:length(t_arr)
        [t,z] = ode45(@q6slope,[0,t_arr(j)],[0,4]);
        subplot(2,2,j)
        plot(t,z);
        legend({'Position','Velocity'},'Fontsize',10, 'Location','northeast')
        xlabel('Time (in s)')
        ylabel('Position/Velocity')
        title(['Time period = ' num2str(t_arr(j)) 's'])
    end


end

%% Plot of position after 30s
%% Varing friction factor

k = 4; m = 4; c = 0.5;
A = [0 1; -k/m -c/m];
[t,z] = ode45(@q6slope,[0,30],[0,4]);


figure;
plot(t,z(:,1));
xlabel('Time (in s)')
ylabel('Position')
title('b = 0.5')

%% Varying mass

k = 4; m = 8; c = 0.25;
A = [0 1; -k/m -c/m];
[t,z] = ode45(@q6slope,[0,30],[0,4]);


figure;
plot(t,z(:,1));
xlabel('Time (in s)')
ylabel('Position')
title('m = 8')
```

It uses the following function to be called by ODE45.

```
%% function to compute dz/dt for given z, A for Q6
% z is a state (column vector)
% dz/dt = A*z


function f = q6slope(t,z)
    global A
    f = A*z;
end
```

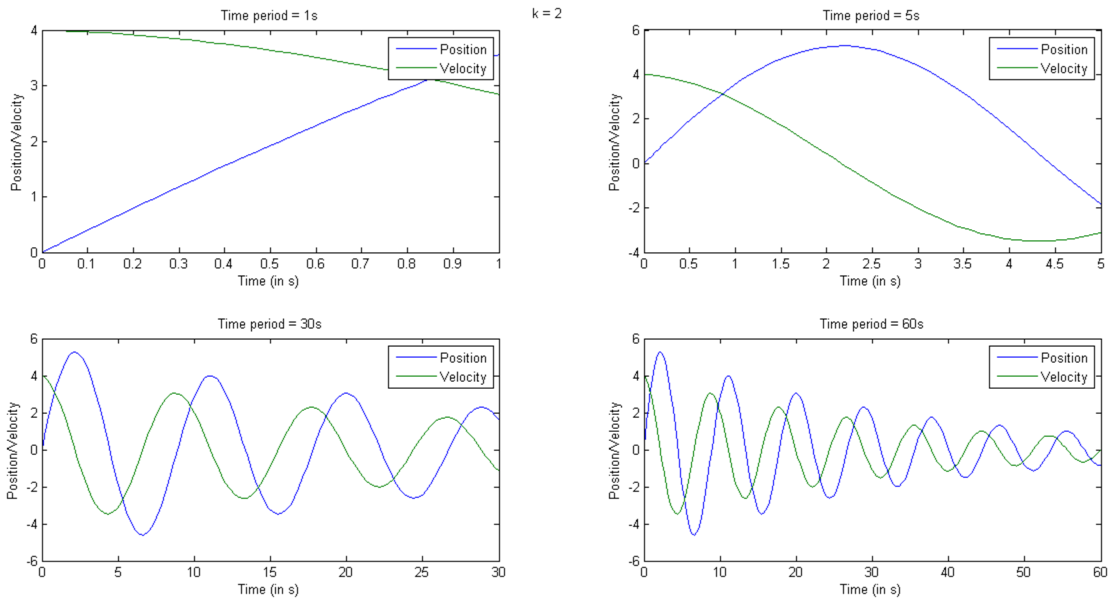

Figure 4: Plots for $k = 2$

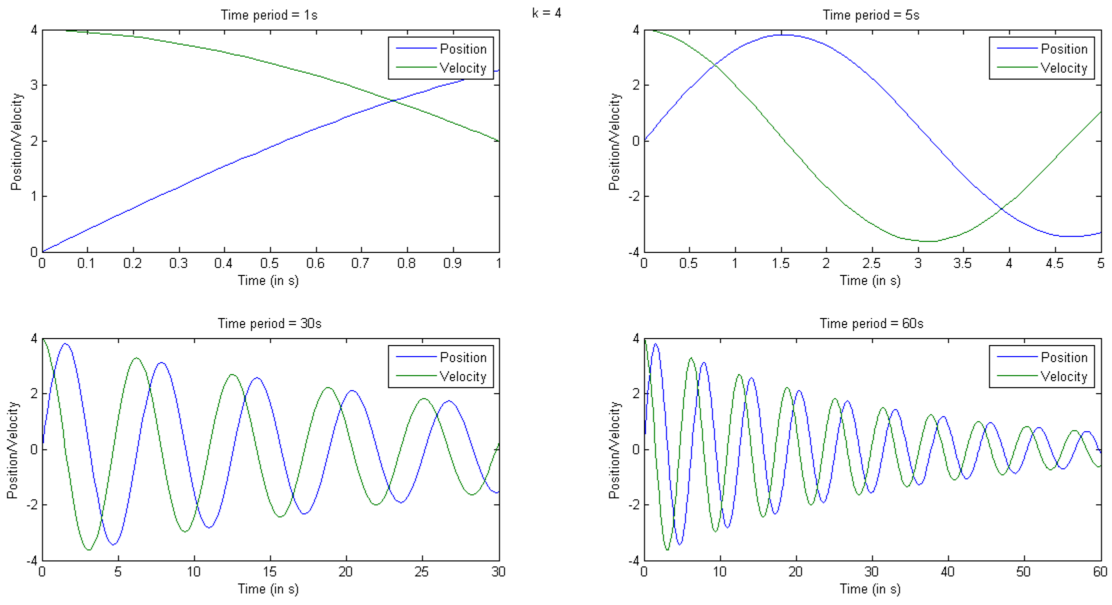

Figure 5: Plots for $k = 4$

For $k = 2$, we can observe that the damped frequency is $\omega_d = \frac{2\pi}{11.22 - 2.053} = 0.6854$ rad/s. The decay rate gives us $\zeta\omega_n = ln(\frac{5.272}{3.977}) = 0.2819$ rad/s.
So, the natural frequency is $\omega_n = sqrt\omega_d^2 + (\zeta\omega_n)^2 = 0.7411$ rad/s. This is around 0.118 Hz. Theoretical calculations gives us that $\omega_n = \sqrt{\frac{k}{m}} = 0.707$ rad/s.

Similarly, for $k = 4$, we get $\omega_d = \frac{2\pi}{7.828 - 1.574} = 1.0047$ rad/s. $\zeta\omega_n = ln(\frac{3.81}{3.131}) = 0.1963$ rad/s. So, $\omega_n = sqrt\omega_d^2 + (\zeta\omega_n)^2 = 1.024$ rad/s. This is about 0.163 Hz. Theory gives us $\omega_n = 1$ rad/s.

Increasing the friction factor (b) increases the damping ratio (ζ) but leaves ω_n unchanged. Hence, the frequency of oscillation, ω_d, will decrease. This can be seen from the following plot with $\omega_d = \frac{2\pi}{7.913-1.578} = 0.9918$ rad/s.

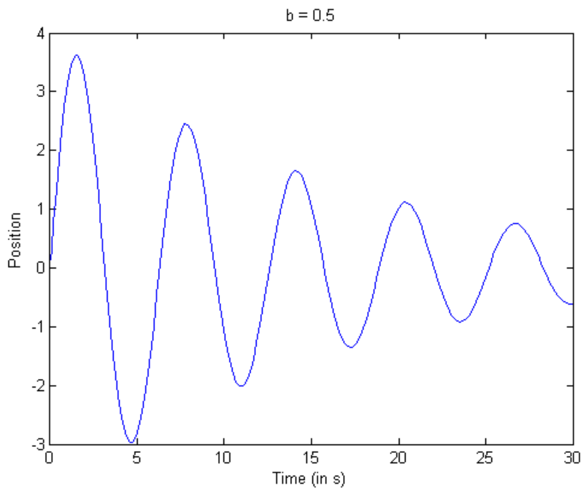

Figure 6: Position after 30s for $b = 0.5$

Increasing the mass (m) affects ω_n, ω_d and ζ. The clearest observable difference is the decreased decay rate with $\zeta\omega_n = ln(\frac{5.441}{4.753}) = 0.135$ rad/s.

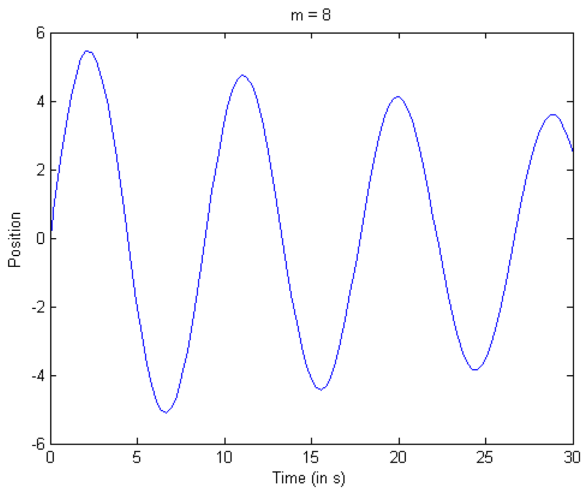

Figure 7: Position after 30s for $m = 8$

Q7

8 points 3 points for Taylor series, 1 point for grouping terms to get exponential form, 4 points for explaining stability

The Taylor series are:

$$\begin{aligned} e^{i\alpha t} &= 1 + i\alpha t - \frac{\alpha^2 t^2}{2!} - i\frac{\alpha^3 t^3}{3!} + ... \\ sin(\alpha t) &= \alpha t - \frac{\alpha^3 t^3}{3!} + \frac{\alpha^5 t^5}{5!} - ... \\ cos(\alpha t) &= 1 - \frac{\alpha^2 t^2}{2!} + \frac{\alpha^4 t^4}{4!} - ... \end{aligned}$$

We can see that

$$\begin{aligned} e^{i\alpha t} &= [1 - \frac{\alpha^2 t^2}{2!} + \frac{\alpha^4 t^4}{4!} - ...] + i[\alpha t - \frac{\alpha^3 t^3}{3!} + \frac{\alpha^5 t^5}{5!} - ...] \\ &= cos(\alpha t) + isin(\alpha t) \end{aligned}$$

Now, for any solution

$$\begin{aligned} x(t) &= x_0 e^{\alpha t} \\ &= x_0 e^{Re(\alpha)t + iIm(\alpha)t} \\ &= x_0 e^{Im(\alpha)t} e^{Re(\alpha)t} \end{aligned}$$

We can see that $x_0 e^{Im(\alpha)t} = x_0[cos(\alpha t) + isin(\alpha t)]$ is a value that will always lie between $-x_0$ and x_0. On the other hand $e^{Re(\alpha)t}$ can either decay to zero or "blow up" to infinity as time goes on depending on $Re(\alpha)$. Hence, the stability of the system is dependent only on $Re(\alpha)$.

Q8

15 points 2 points for description of each file (10 points), 2 points for modified parameter for overshoot, 3 points for other performance optimization
The purpose of the files are listed below.

- **amsetup:** This file sets the values for plotting parameters to ensure consistency across plots.
- **aminit:** This is an initialization file. It checks for the presence of amsetup.m in the directory, as per some pre-defined directory structure, and runs it if it has not already been run.
- **amprint:** This file contains the script to print figures to the appropriate file based on the value of the AMPRINT_FLAG.
- **cruisedyn:** This function essentially contains the equations of motion governing the motion of the vehicle. It returns the acceleration of the vehicle, given information on current velocity, throttle setting, gear ratio, road slope, user input (fuel injection rate) and mass (if not default). There are certain default parameter values, which are used based on the number of input arguments to the function.
- **cruise_response:** This file is the main file in that it models the dynamical system and derives the feedback model based on that. It employs a PI controller. The output of this file are plots of the step response of the system with the step input given at $t = 2.5s$.

With $K_p = 0.8, K_i = 0.02$, we can get an overshooot just 0.84m/s.

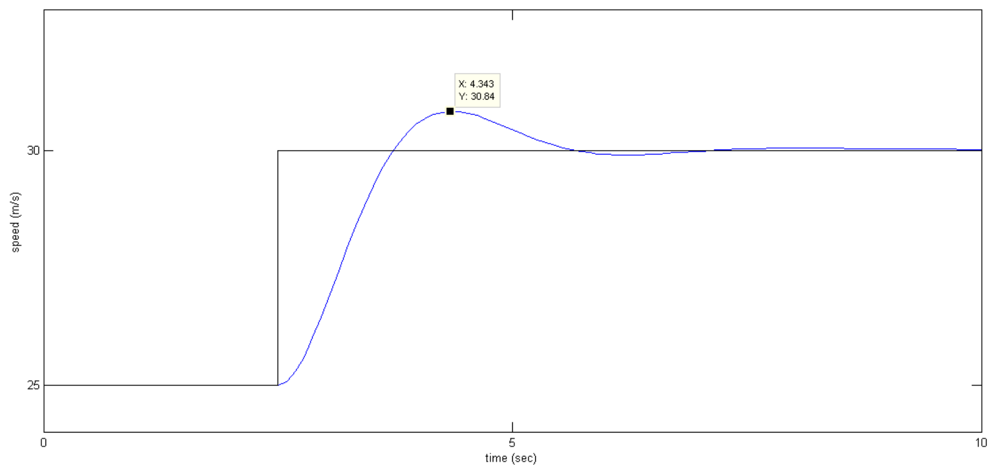

Figure 8: Step response for $m = 2500$ kg, $K_p = 0.8, K_i = 0.02$

If we look to minimize the rise time, it can be observed that varying K_i does not significantly alter rise time but increasing K_p reduces the rise time. The step response for $K_p = 3, K_i = 0.02$ is shown below.

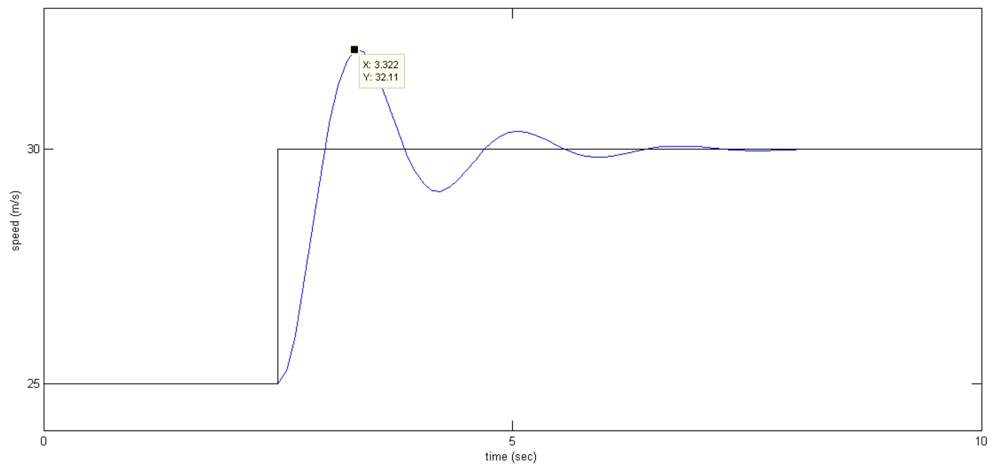

Figure 9: Step response for $m = 2500$ kg, $K_p = 3, K_i = 0.02$

It is notable that increasing K_p increases overshoot as well. So, there is a trade-off between minimizing overshoot and rise time. Further increasing K_p will reduce rise time further.