The Rockin' Roller's velocity and displacement options

John Carey

Introduction-

This project is an effort to engineer a fun but safe carnival ride, The Rockin' Roller. The Rockin' Roller will operate as an oscillating seat ride based on a spring-dashpot system. Data is available for the spring co-efficient and 3 different damper co-efficients. Co-efficient of the spring will be determined using Force and Displacement data, and co-efficients of the dampers will be determined using Force and Velocity data. My job is to find these co-efficients, find out the mass of the seat plus a rider, and then graph the velocity and displacement of the ride based on the mass and these co-efficients. Velocity and Displacement data will be found using the standard equation of motion for a damped mass-spring system. This data will be plotted, and one of the 3 dampers will be chosen based on predetermined safety and excitement criteria.

Program Description.

The program begins by loading the four data files with the spring and damper information. We create data variables to plot

```
% Step 1: Load the data files
KData = load('C:\Users\johnm\OneDrive\Desktop\KData.txt');
C1Data = load('C:\Users\johnm\OneDrive\Desktop\C1Data.txt');
C2Data = load('C:\Users\johnm\OneDrive\Desktop\C2Data.txt');
C3Data = load('C:\Users\johnm\OneDrive\Desktop\C3Data.txt');
```

Next, we use linear regression to find the best fit for the data. The co-efficients will be the slope of these lines. Polyfit, the built in matlab feature, will find the line of best fit for us. Since the data returns a slope and an intercept, I set new variable that are just the slope (the co-efficient.)

```
% Step 2: Use linear regression to determine the spring constant and damping coefficients
k = polyfit(KData(:,1), KData(:,2), 1);
c1 = polyfit(C1Data(:,1), C1Data(:,2), 1);
c2 = polyfit(C2Data(:,1), C2Data(:,2), 1);
c3 = polyfit(C3Data(:,1), C3Data(:,2), 1);

l=k(1);

cc1 = c1(1);
cc2 = c2(1);
cc3 = c3(1);
```

From here we plot the data, label the axis and put in a legend. This will ensure that our line of best fit is closely grouped with the data.

```
% Step 3: Generate plots of the testing data for the spring and dampers
figure;
plot(KData(:,1), KData(:,2), 'o');
hold on;
plot(KData(:,1), polyval(k, KData(:,1)), '-');
xlabel('Displacement (m)');
```

```
ylabel('Force (N)');
legend('Spring Data', 'Spring Fit');

figure;
plot(C1Data(:,1), C1Data(:,2), 'o');
hold on;
plot(C1Data(:,1), polyval(c1, C1Data(:,1)), '-');
xlabel('Velocity (m/s)');
ylabel('Force (N)');
legend('Damper 1 Data', 'Damper 1 Fit');

figure;
plot(C2Data(:,1), C2Data(:,2), 'o');
hold on;
plot(C2Data(:,1), polyval(c2, C2Data(:,1)), '-');
xlabel('Velocity (m/s)');
ylabel('Force (N)');
legend('Damper 2 Data', 'Damper 2 Fit');

figure;
plot(C3Data(:,1), C3Data(:,2), 'o');
hold on;
plot(C3Data(:,1), polyval(c3, C3Data(:,1)), '-');
xlabel('Velocity (m/s)');
ylabel('Force (N)');
legend('Damper 3 Data', 'Damper 3 Fit');
```

Now we will use numerical integration to compute the mass of a Rockin' Roller seat. I set values for density and width and then created the function to be integrated. I first used the quad function to integrate the equation over the given bounds. This was just to ensure I get the right number. Next, I did a gauss quadrature to get my manual integration. I set my bounds as A and B. From these bounds I extrapolated my co-efficients and the remainder for the quadrature equations. X1 and x2 are the the x values using the quadrature equations. Then I ran them thru the functions with eq1 and eq2. The variable ms was the mass of the seat over the density and width, the sum of the functions at x1 and x2 times the coefficient. I multiplied by density and width to get the final mass. Then I added 75 to get the mass of the rider and the seat.

```
% Step 4: Use numerical integration to compute the mass of a Rockin' Roller seat
w= .5;
p = 100;

F = @(x)(3.*(x.^2).*cosd(1.2./x.^(1/2))+.7);

m1 = quad(F,.05,.75);

a=.05; b= .75

co = (b-a)/2
rem = (a+b)/2

x1 = co*(-1/sqrt(3))+rem
x2 = co*(1/sqrt(3))+rem

eq1 = (3.*(x1.^2).*cosd(1.2./x1.^(1/2))+.7);
eq2 = (3.*(x2.^2).*cosd(1.2./x2.^(1/2))+.7);
ms = co*(eq1)+co*(eq2)

mseat = ms*.5*100
m = mseat + 75
```

-1

Using cosd may be what caused error in m.

Use regular cos instead (radians version)

Next, I use the Runga Kutta method to determine velocity and displacement of each of the 3 dampers. Each block of the code is one damper. I call the state space function (at the end of the code), followed by the time interval, the initial conditions, the step size, and then pass three variables into the function call, m is the mass of the seat and rider, cc1-3 are the damper coefficients and l is the spring constant. I then plot each of the velocities and displacements on a singular graph.

```
[tp1,yp1] = rk4sys(@dxdt_c11,[0,5],[-1,0],.1,m,cc1,l)

figure;
plot(tp1,yp1)
hold on
xlabel('time')
ylabel('Velocity and Displacement')
legend('C1 Displacement', 'C1 Velocity');


[tp2,yp2] = rk4sys(@dxdt_cc2,[0,5],[-1,0],.1,m,cc2,l)



figure;
plot(tp2,yp2)
hold on
xlabel('time')
ylabel('Velocity and Displacement')
legend('C2 Displacement', 'C2 Velocity');

[tp3,yp3] = rk4sys(@dxdt_cc3,[0,5],[-1,0],.1,m,cc3,l)
%[tp,yp] = rk4sys(@dxdt_c3,[0,5],[-1,0],.1,m,c)


figure;
plot(tp3,yp3)
hold on
xlabel('time')
ylabel('Velocity and Displacement')
legend('C3 Displacement', 'C3 Velocity');
```

Now, I separate out the velocities and displacements and plot them overlaying each other, one graph for displacements and one graph for velocities.

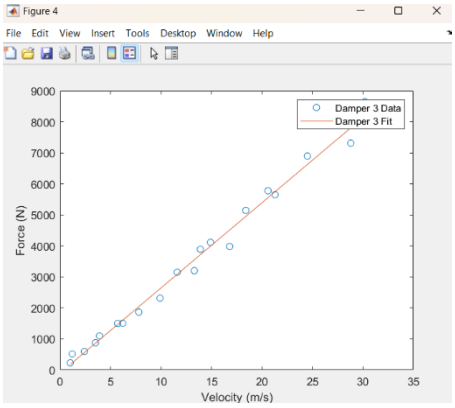
```
figure;
plot(tp1,yp1(:,1))
hold on
plot(tp2,yp2(:,1))
hold on
plot(tp3,yp3(:,1))
xlabel('time')
ylabel('Displacement (m)')
legend('C1 Displacement', 'C2 Displacement', 'C3 Displacement');

figure;
plot(tp1,yp1(:,2))
hold on
plot(tp2,yp2(:,2))
hold on
plot(tp3,yp3(:,2))
xlabel('time')
ylabel('Velocity (m/s)')
legend('C1 Velocity', 'C2 Velocity', 'C3 Velocity');
```

Finally, I set up my state space functions. I pass in the variables for mass, the damping coefficients and the spring constant so the math can be done in the state space calculations.

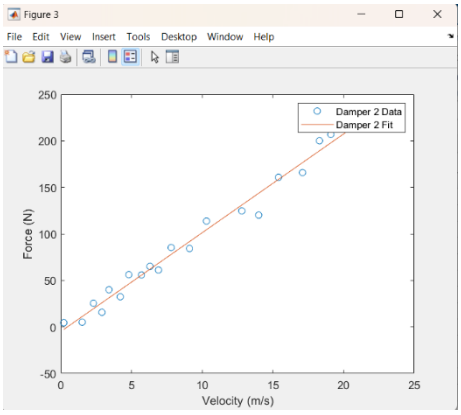
```
function xdot = dxdt_c11(t,x,m,cc1, l) %pass in same order

xdot=[0 1;-l/m -cc1/m]*x';

end

function xdot = dxdt_cc2(t,x,m,cc2,l)

xdot=[0 1;-l/m -cc2/m]*x';

end

function xdot = dxdt_cc3(t,x,m,cc3,l)

xdot=[0 1;-l/m -cc3/m]*x';

end
```

Plots of testing data and linear regression models.

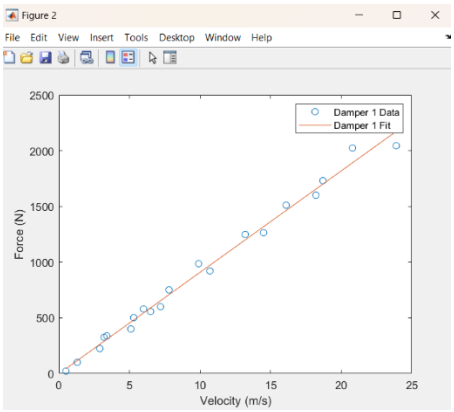
Damper 3 plot



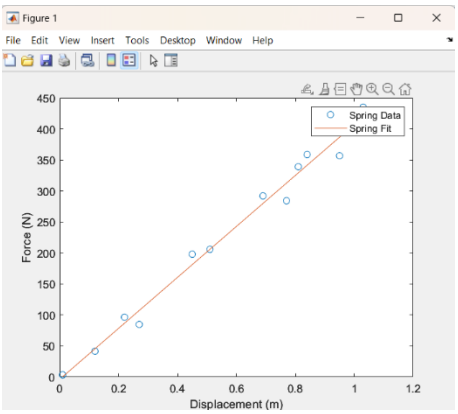
Damper 2 plot



Damper 1 plot



Spring plot



Values

K = 410.9260

Damper 1 coefficient = 91.0587

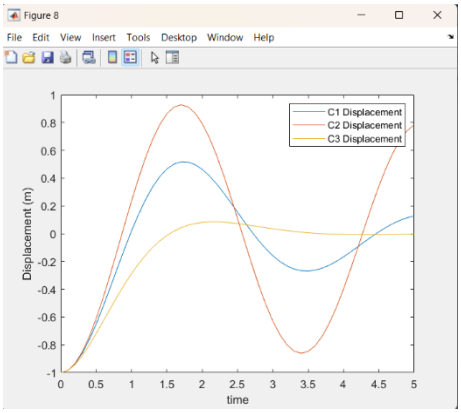
Damper 2 coefficient = 10.6057

Damper 3 coefficient = 274.4261

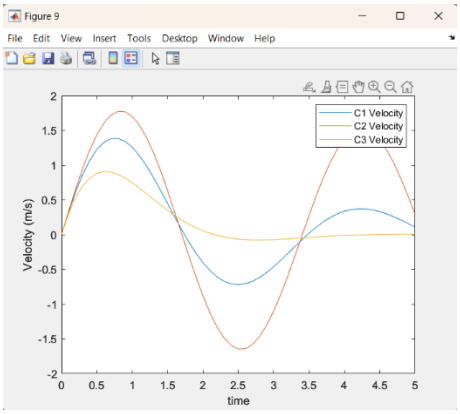
Mass of seat = .9116 kg

Plots of seat locations and velocities

Displacement vs Time



Velocity vs Time



Choice of Damper

The damper that is best suited for the given criteria is C1. C2 has a velocity that exceeds the maximum 1.5 m/s, and is therefore not suitable for the ride. C3 doesn't cross the equilibrium threshold 3 times as per the given critera, and is therefore not suitable. C2 meets all criteria, crossing the threshold 3 times in 5 seconds and having a max velocity under 1.5 m/s.

Code

```
% Step 1: Load the data files
```

```
KData = load('C:\Users\johnm\OneDrive\Desktop\KData.txt');
C1Data = load('C:\Users\johnm\OneDrive\Desktop\C1Data.txt');
C2Data = load('C:\Users\johnm\OneDrive\Desktop\C2Data.txt');
C3Data = load('C:\Users\johnm\OneDrive\Desktop\C3Data.txt');

% Step 2: Use linear regression to determine the spring constant and damping coefficients
k = polyfit(KData(:,1), KData(:,2), 1);
c1 = polyfit(C1Data(:,1), C1Data(:,2), 1);
c2 = polyfit(C2Data(:,1), C2Data(:,2), 1);
c3 = polyfit(C3Data(:,1), C3Data(:,2), 1);

l=k(1);

cc1 = c1(1);
cc2 = c2(1);
cc3 = c3(1);

% Step 3: Generate plots of the testing data for the spring and dampers
figure;
plot(KData(:,1), KData(:,2), 'o');
hold on;
plot(KData(:,1), polyval(k, KData(:,1)), '-');
xlabel('Displacement (m)');
ylabel('Force (N)');
legend('Spring Data', 'Spring Fit');

figure;
plot(C1Data(:,1), C1Data(:,2), 'o');
hold on;
plot(C1Data(:,1), polyval(c1, C1Data(:,1)), '-');
xlabel('Velocity (m/s)');
ylabel('Force (N)');
legend('Damper 1 Data', 'Damper 1 Fit');

figure;
plot(C2Data(:,1), C2Data(:,2), 'o');
hold on;
plot(C2Data(:,1), polyval(c2, C2Data(:,1)), '-');
xlabel('Velocity (m/s)');
ylabel('Force (N)');
legend('Damper 2 Data', 'Damper 2 Fit');

figure;
plot(C3Data(:,1), C3Data(:,2), 'o');
hold on;
plot(C3Data(:,1), polyval(c3, C3Data(:,1)), '-');
xlabel('Velocity (m/s)');
ylabel('Force (N)');
legend('Damper 3 Data', 'Damper 3 Fit');

% Step 4: Use numerical integration to compute the mass of a Rockin' Roller seat
w= .5;
p = 100;

F = @(x)(3.*(x.^2).*cosd(1.2./x.^(1/2))+.7);

m1 = quad(F,.05,.75);

a=.05; b= .75

co = (b-a)/2
rem = (a+b)/2

x1 = co*(-1/sqrt(3))+rem
x2 = co*(1/sqrt(3))+rem

eq1 = (3.*(x1.^2).*cosd(1.2./x1.^(1/2))+.7);
eq2 = (3.*(x2.^2).*cosd(1.2./x2.^(1/2))+.7);
ms = co*(eq1)+co*(eq2)
```

```
mseat = ms*.5*100
m = mseat + 75

%step 5 numerically solve for the motion of seat



[tp1,yp1] = rk4sys(@dxdt_c11,[0,5],[-1,0],.1,m,cc1,l)

figure;
plot(tp1,yp1)
hold on
xlabel('time')
ylabel('Velocity and Displacement')
legend('C1 Displacement', 'C1 Velocity');


[tp2,yp2] = rk4sys(@dxdt_cc2,[0,5],[-1,0],.1,m,cc2,l)



figure;
plot(tp2,yp2)
hold on
xlabel('time')
ylabel('Velocity and Displacement')
legend('C2 Displacement', 'C2 Velocity');

[tp3,yp3] = rk4sys(@dxdt_cc3,[0,5],[-1,0],.1,m,cc3,l)
%[tp,yp] = rk4sys(@dxdt_c3,[0,5],[-1,0],.1,m,c)


figure;
plot(tp3,yp3)
hold on
xlabel('time')
ylabel('Velocity and Displacement')
legend('C3 Displacement', 'C3 Velocity');

figure;
plot(tp1,yp1(:,1))
hold on
plot(tp2,yp2(:,1))
hold on
plot(tp3,yp3(:,1))
xlabel('time')
ylabel('Displacement (m)')
legend('C1 Displacement', 'C2 Displacement', 'C3 Displacement');

figure;
plot(tp1,yp1(:,2))
hold on
plot(tp2,yp2(:,2))
hold on
plot(tp3,yp3(:,2))
xlabel('time')
ylabel('Velocity (m/s)')
legend('C1 Velocity', 'C2 Velocity', 'C3 Velocity');



function xdot = dxdt_c11(t,x,m,cc1, l) %pass in same order

xdot=[0 1;-l/m -cc1/m]*x';

end
```

```
function xdot = dxdt_cc2(t,x,m,cc2,l)

xdot=[0 1;-l/m -cc2/m]*x';

end

function xdot = dxdt_cc3(t,x,m,cc3,l)

xdot=[0 1;-l/m -cc3/m]*x';

end
```