Table of Contents


```
% Joshua Oates
% lab 6, numerical integration and differentiation
```

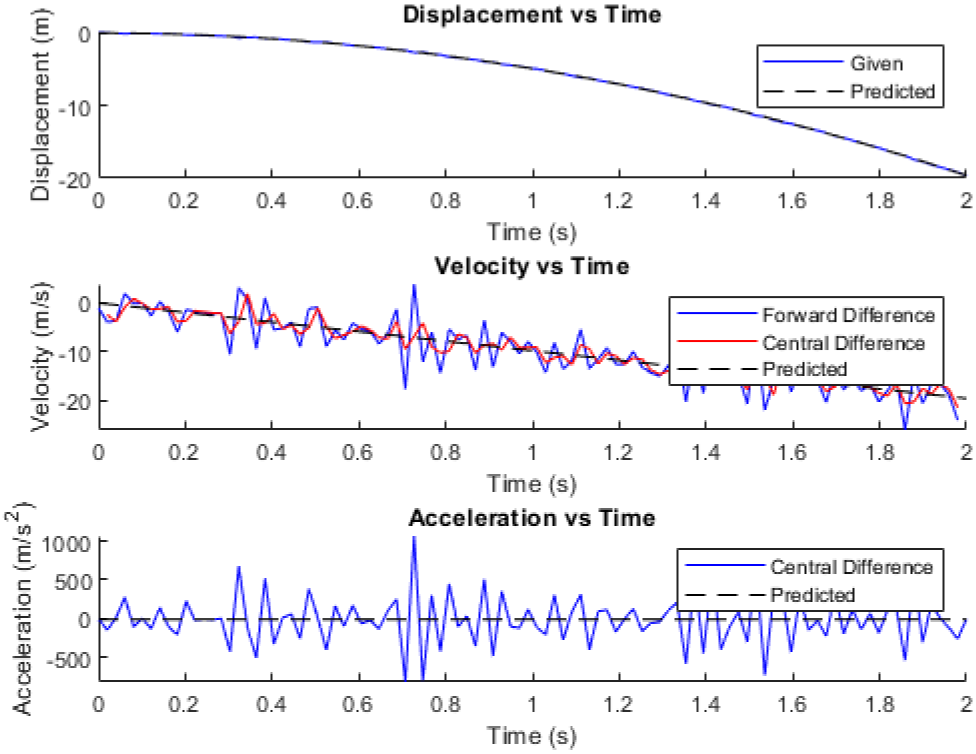
section 0 - clean up

```
clear all;
close all;
clc;
addpath('Data')
```

section 1 - numerical differentiation

```
dis = load("dispData.mat");
dis = [dis.t',dis.y']; % reformat dis
[m,~] = size(dis);

fpDis = zeros(m,2);
for i = 1:m-1
    y1 = dis(i,2);
    y2 = dis(i+1,2);
    h = dis(i+1,1)-dis(i,1);
    fpDis(i,1) = TwoPForwardDiff(y1,y2,h); % first derivative of displacement
 is velocity, find useing forward difference
end
for i = 2:m-1
    y0 = dis(i-1,2);
    y2 = dis(i+1,2);
    h = (dis(i+1,1)-dis(i-1,1))/2;
    fpDis(i,2) = ThreePCenDiff1(y0,y2,h); % find using central difference
end

fppDis = zeros(m,1);
for i = 2:m-1
    y0 = dis(i-1,2);
    y1 = dis(i,2);
    y2 = dis(i+1,2);
    h = (dis(i+1,1)-dis(i-1,1))/2;
    fppDis(i) = ThreePCenDiff2(y0,y1,y2,h); % second derivative of
 displacement is acceleration, find using Three Point Central differenece for
 second derivative
end
```

```
clear y0 y1 y2 h i

t = linspace(0,2); % create a t vector to plot against for prediction values
a = -9.80665; % create other prediction vectors
V = @(t) a*t;
y = @(t) .5*a*t.^2;
a = ones(length(t),1) * a;


figure % plot both the given and predicted displacements
subplot(3,1,1)
hold on
plot(dis(:,1),dis(:,2),'b')
plot(t,y(t),'k--')
legend("Given","Predicted")
xlabel("Time (s)")
ylabel("Displacement (m)")
title("Displacement vs Time")

subplot(3,1,2) % plot modelded and predicted values for velocity
hold on
plot(dis(1:m-1,1),fpDis(1:m-1,1),'b') % one data point removed because there
 is one less panel than data points
plot(dis(2:m-1,1),fpDis(2:m-1,2),'r')
plot(t,V(t),'k--')
legend("Forward Difference","Central Difference","Predicted")
xlabel("Time (s)")
ylabel("Velocity (m/s)")
title("Velocity vs Time")

subplot(3,1,3) % plost modeled and predicted values for acceleration
hold on
plot(dis(:,1),fppDis,'b')
plot(t,a,'k--')
legend("Central Difference", "Predicted")
xlabel("Time (s)")
ylabel("Acceleration (m/s^2)")
title("Acceleration vs Time")

clear all
```



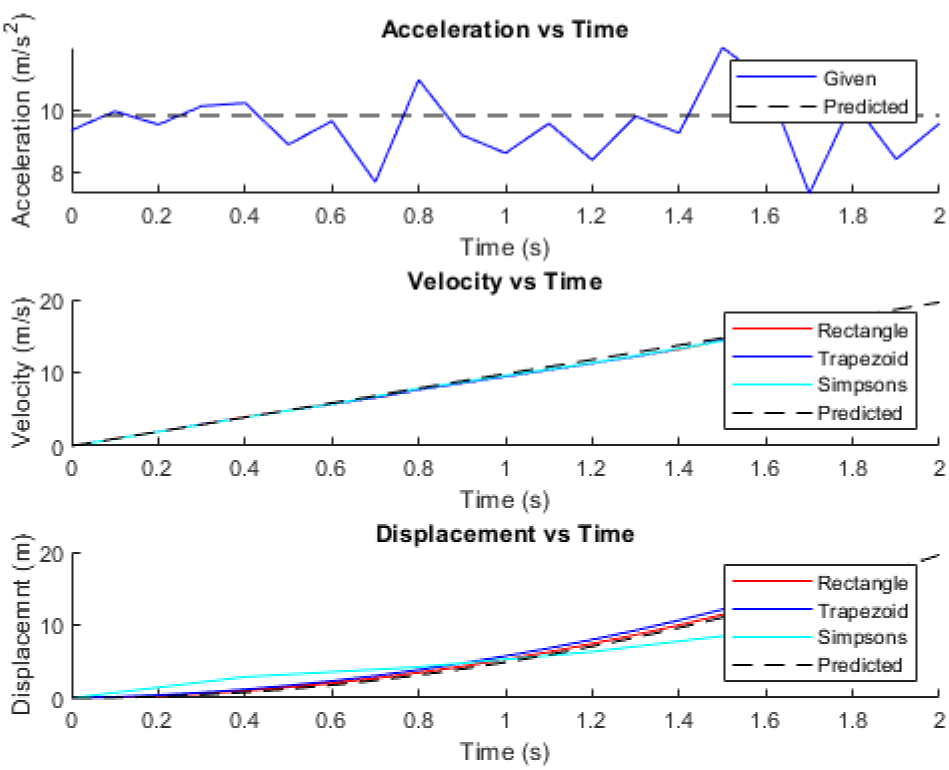
section 2 - numerical integration

```
acc = load("accelData.mat"); % load and format acc
acc =  [acc.t',acc.acc];

% take integrals, store in both a long and short list for each method

[m,~] = size(acc);
IaccLong = zeros(m,3);
IaccShort1 = [0];
IaccShort2 = [0];
IaccShort3 = [0];
IT1 = [0];
IT3 = [0];

for i= 2:m-1 % use rectangular rule for the first integral (I) and store it in
 a long version for my use and a short for plotting
    yI1 = IaccLong(i-1,1);
    y1 = acc(i,2);
    h = acc(i+1,1)-acc(i,1);
    yI2 = RectangularRule(yI1,y1,h);
    IaccLong(i,1) = yI2;
    if yI2 ~= 0
        IaccShort1 = [IaccShort1;yI2];
        IT1 = [IT1;acc(i,1)];
```

```
    end
end

for i= 2:m-1 % repeat for trapezoid rule
    yI1 = IaccLong(i-1,2);
    y1 = acc(i,2);
    y2 = acc(i+1,2);
    h = acc(i+1,1)-acc(i,1);
    yI2 = TrapezoidRule(yI1,y1,y2,h);
    IaccLong(i,2) = yI2 ;
    if yI2 ~= 0
        IaccShort2 = [IaccShort2;yI2];
    end
end

for i= 3:2:m-1 % repeat for simpsons rule, it will only produce half of the
 points because it takes 3 instead of 2 to create a value
    yI1 = IaccLong(i-2,3);
    y0 = acc(i-1,2);
    y1 = acc(i,2);
    y2 = acc(i+1,2);
    h = acc(i+1,1)-acc(i,1);
    yI2 = Simpsons(yI1,y0,y1,y2,h);
    IaccLong(i,3) = yI2;
    if yI2 ~= 0
        IaccShort3 = [IaccShort3;yI2];
        IT3 = [IT3;acc(i,1)];
    end
end

% repeat for second integral (II)

IIaccLong = zeros(m,3);
[m1,~] = size(IaccShort1);
m2 = m1;
[m3,~] = size(IaccShort3);
IIaccShort1 = [0];
IIaccShort2 = [0];
IIT1 = [0];
IIT3 = [0];

for i= 2:m1-1 % similar to above but uses the I integral instead of acc
    yI1 = IIaccLong(i-1,1);
    y1 = IaccLong(i,2);
    h = IT1(i+1,1)-IT1(i,1);
    yI2 = RectangularRule(yI1,y1,h);
    IIaccLong(i,1) = yI2;
    if yI2 ~= 0
        IIaccShort1 = [IIaccShort1;yI2];
        IIT1 = [IIT1;IT1(i)];
    end
end

for i= 2:m2-1 % similar again
```

```
    yI1 = IIaccLong(i-1,2);
    y1 = IaccLong(i,2);
    y2 = IaccLong(i+1,2);
    h = IT1(i+1,1)-IT1(i,1);
    yI2 = TrapezoidRule(yI1,y1,y2,h);
    IIaccLong(i,2) = yI2 ;
    if yI2 ~= 0
        IIaccShort2 = [IIaccShort2;yI2];
    end
end

f = IaccShort3; % here I was having a lot of issues with indexing so I
 eventually just used Deffo's algorithim and did my best to implement it
n = 5;
h = .1;

IIaccShort3 =0;

for j = 1:n-1
    IIT3= [IIT3;4*h*j];
    fa = f(1);
    fb = f(j+1);
    I = fa + fb;
    for i=1:j
        I = I + 2*f(2*i+1) + 4*f(2*i);
    end
    I = (I + 4*IaccShort3(2*n))*h/3;
    IIaccShort3 =[IIaccShort3;I];
end
% I believe that this is the correct implementation


clear m m1 m2 m3 y0 y1 y2 yI1 yI2 i

t = linspace(0,2);
a = 9.80665;
V = @(t) a*t;
y = @(t) .5*a*t.^2;
a = ones(length(t),1) * a;

figure
subplot(3,1,1)
hold on
plot(acc(:,1),acc(:,2),'b')
plot(t,a,'k--')
legend("Given","Predicted")
xlabel("Time (s)")
ylabel("Acceleration (m/s^2)")
title("Acceleration vs Time")


subplot(3,1,2)
hold on
plot(IT1,IaccShort1,'r')
```

```
plot(IT1,IaccShort2,'b')
plot(IT3,IaccShort3,'c')
plot(t,V(t),'k--')
legend("Rectangle","Trapezoid","Simpsons","Predicted")
xlabel("Time (s)")
ylabel("Velocity (m/s)")
title("Velocity vs Time")

subplot(3,1,3)
hold on
plot(IIT1,IIaccShort1,'r')
plot(IIT1,IIaccShort2,'b')
plot(IIT3,IIaccShort3,'c')
plot(t,y(t),'k--')
legend("Rectangle","Trapezoid","Simpsons","Predicted")
xlabel("Time (s)")
ylabel("Displacemnt (m)")
title("Displacement vs Time")
```



section 3 - discussion

```
disp("Because it is more accurate to integrate if using the correct
 algorithim, I would prefer to start with acceleration to find speed rather
 than postion. Numerical integration has the tendancy of smoothing out errors
 in data, while numerical differentiation has the tendancy to exacerbate
 them.")
```

Because it is more accurate to integrate if using the correct algorithim, I would prefer to start with acceleration to find speed rather than postion. Numerical integration has the tendancy of smoothing out errors in data, while numerical differentiation has the tendancy to exacerbate them.

section 4 - function definitions

```
function [yI2] = RectangularRule(yI1,y1,h)
    % this function will return the running total integral yI2 for a
    % preivous total yI1, y1 and an h value
    yI2 = yI1 + y1*h;
end

function [yI2] = Simpsons(yI0,y0,y1,y2,h)
% takes previous running total, and 3 data points and an h and gives the
% next running total for the integral
yI2 = yI0 + (h/3)*(y2 + 4*y1 + y0);
end

function [fp] = ThreePCenDiff1(y0,y2,h)
% takes y0 and y2 with one data point in the middle and the distance h
% y0 = f(x-h)
% y2 = f(x+h)
fp = (y2-y0)/(2*h);
end

function [fpp] = ThreePCenDiff2(y0,y1,y2,h)
% takes y0 y2 y3 with one data point in the middle and the distance h
% y0 = f(x-h)
% y1 = f(x)
% y2 = f(x+h)
fpp = (y0-2*y1+y2)/(h^2);
end

function [yI2] = TrapezoidRule(yI1, y1, y2,h)
% this function takes an the previous value yI1, y1, y2, and h and returns
% runningn integral total
yI2 = yI1 + (h/2)*(y1+y2);
end

function [fp] = TwoPForwardDiff(y1,y2,h)
% takes forward difference at one point
% y1 = f(x)
% y2 = f(x+h)
fp = (y2-y1) / h;
end
```

Published with MATLAB® R2022a