Table of Contents


```
%Roshan Jaiswal-Ferri
%Section - 03
%Aero 300 Lab 6 - Numeric Differentiation and Integration: 5/16/24
```

Workspace Prep

```
format long     %Allows for more accurate decimals
close all;      %Clears all
clear all;      %Clears Workspace
clc;            %Clears Command Window
```

PART 1: Numeric Differentiation

```
disp = load("dispData.mat");

dt = disp.t(1,2)-disp.t(1); %step size (.0202s)
t = disp.t(1:end);
t2 = disp.t(1:end-1);
t3 = disp.t(1:end-2);
f = disp.y;

%Creating Functions for expected values:
p1 = polyfit(t,f,3); %Creating a degree 3 polynomial to fit disp data
p2 = polyder(p1); %taking derivitive to find velocity
p3 = polyder(p2); %taking derivitive to find acceleration

%p4 = polyval(p1,t); Dont need this one cuz we already have measurment data
p5 = polyval(p2,t); %Evaluating the velocity polynomial to find expected
velocity at time t
p6 = polyval(p3,t); %Evaluating the acceleration polynomial to find expected
accel at time t

%Numeric Differentiation:
dsdtF2 = ((f(2:end))-f(1:end-1))/(dt); %two point center diff.
dsdtF3 = ((f(3:end))-f(1:end-2))/(2*dt); %three point center diff.
dsdt2F3 = (f(3:end)-(2*f(2:end-1))+f(1:end-2))/(dt^2); %2nd Deriv

figure('Name','Numeric Differentiation')
```

```
subplot(1,3,1)
plot(t,f,'rx')
hold on
grid on
plot(t,f,'b')
xlabel('Time (s)')
ylabel('Displacement')
title('Displacement vs Time')
legend('Measured Displacement', 'Expected Displacement', 'location', 'best')

subplot(1,3,2)
plot(t2,dsdtF2)
hold on
grid on
plot(t3,dsdtF3)
plot(t,p5,'x')
xlabel('Time (s)')
ylabel('Velocity')
title('Velocity vs Time')
legend('Two Point Forward Diff.', 'Three Point Center Diff.', 'Expected
Velocity', 'location', 'best')

subplot(1,3,3)
plot(t3,dsdt2F3)
hold on
grid on
plot(t,p6,'x')
xlabel('Time (s)')
ylabel('Acceleration')
title('Acceleration vs Time')
legend('Three Point Centered Diff.', 'Expected Accel', 'location', 'best')
```

PART 2: Numeric Integration – Quadrature

```
accel = load("accelData.mat");
accelc = accel.acc;
acc = accelc'; %convert to row
t = accel.t;

%Rectangle Rule
rRV = rectRule(t,acc);
rRD = rectRule(t,rRV);

%Trapeziod Rule
tRV = trapRule(t,acc);
tRD = trapRule(t,tRV); %Toyota Racing Development! jk its trap rule
displacement :/

%Simpson's Rule
sRV = simpRule(t,acc); %sRV = search and rescue vehicle? Nope, simpsons rule
velocity
sRD = simpRule(t,sRV);
```

```
figure('Name','Numeric Integration - Quadrature')
subplot(1,3,1)
plot(t,rRD)
hold on
grid on
plot(t,tRD)
plot(t,sRD)
xlabel('Time (s)')
ylabel('Displacement')
title('Displacement vs Time')
legend('Rectangle Rule', 'Trapezoid Rule', 'Simpsons Rule', 'location',
'best')

subplot(1,3,2)
plot(t,rRV)
hold on
grid on
plot(t,tRV)
plot(t,sRV)
xlabel('Time (s)')
ylabel('Velocity')
title('Velocity vs Time')
legend('Rectangle Rule', 'Trapezoid Rule', 'Simpsons Rule', 'location',
'best')

subplot(1,3,3)
plot(t,acc)
hold on
grid on
plot(t,acc,'x')
xlabel('Time (s)')
ylabel('Acceleration')
title('Acceleration vs Time')
legend('Measured Accel', 'Expected Accel', 'location', 'best')
```

Part 3: Discussion

```
%As far as differentiation goes the three point center diff. method is the
%most accurate of the given methods and the same goes for simpson's rule
%for integration. This is simply because there are more test points in
%these methods compared to their counter parts, which allow for higher
%accuracy. Regarding which of these two to use, I think depends on the
%data, if you have cleaner acceleration data, then use simpson's rule, if
%the displacement data is more accurate use the three point center diff.
%method.
```

Function: Rectangle Rule

```
function [int] = rectRule(t,f) %t is time, f is the function to integrate
must be row vector

    int = zeros(size(t)); %initializes the integral vector starting at 0
```

```
    for i = 1:length(t)-1
        h = t(1,i+1) - t(1,i);
        int(1,i+1) = int(1,i)+h*f(1,i+1);
    end

end
```

Function: Trapezoid Rule

```
function [int] = trapRule(t,f) %t is time, f is the function to integrate
must be row vector

    int = zeros(size(t)); %initializes the integral vector starting at 0

    for i = 1:length(t)-1
        h = t(1,i+1) - t(1,i);
        int(1,i+1) = int(1,i)+(h/2)*(f(1,i+1)+f(1,i));
    end

end
```

Function: Simpson's Rule

```
function [int] = simpRule(t,f) %t is time, f is the function to integrate
must be row vector

    int = zeros(size(t)); %initializes the integral vector starting at 0

    for i = 1:length(t)-1
        h = t(1,i+1) - t(1,i);
        if i == 1
            int(1,i+1) = int(1,i)+(h/3)*(f(1,i+1)+4*f(1,i)+0); %plus zero if
i is 1 because if you cannot have an array position < 0
        else
            int(1,i+1) = int(1,i)+(h/3)*(f(1,i+1)+(4*f(1,i))+f(1,i-1));
        end
    end

end
```

Published with MATLAB® R2023b