Jonathan Pilling

HW2 Writeup

CS 3200

1. Problem 1 Finite Differences
    a. Starting with a sample function x^2, I set up some arbitrarily picked values for x0, x1, and x2. Using these values in the LaGrange formulas, and then plugging this into our polynomial interpolant formula I got a function that was similar to the starting function.

```
%Problem 1 part a

syms x0 x1 x2 x f(x);
f(x) = x^2; %Sample function for testing

%Arbitrarily picked values
x0 = 1;
x1 = 2;
x2 = 10;

%Lagrange Values
l0 = ((x-x1)*(x-x2))/((x0-x1)*(x0-x2));
l1 = ((x-x0)*(x-x2))/((x1-x0)*(x1-x2));
l2 = ((x-x0)*(x-x1))/((x2-x0)*(x2-x1));

%Base polynomial interpolation function
Q = f(x0) * l0 + f(x1) * l1 + f(x2) * l2;
```

Taking the derivative of this Q function gave me the derivative of the original function. Using the finite distance formulas for points I was able to compute the values for x0, x1, and x2.

```
%Derivative of the approximation
Q_approx(x) = simplify(diff(Q,x));

%Finite Difference formulas for three points
h = 0.1; %Given 0.1 for testing
fd_x0 = (f(x0 + h) - f(x0 - h)) / 2*h;
fd_x1 = (f(x1 + h) - f(x1 - h)) / 2*h;
fd_x2 = (f(x2 + h) - f(x2 - h)) / 2*h;
```

    b. Manipulating Equation 5.3 from the notes gave me a formula where you subtract from the terms with h's in it, the derived, derivative approximation. Given below, with test points this gave me error estimate equal to h.

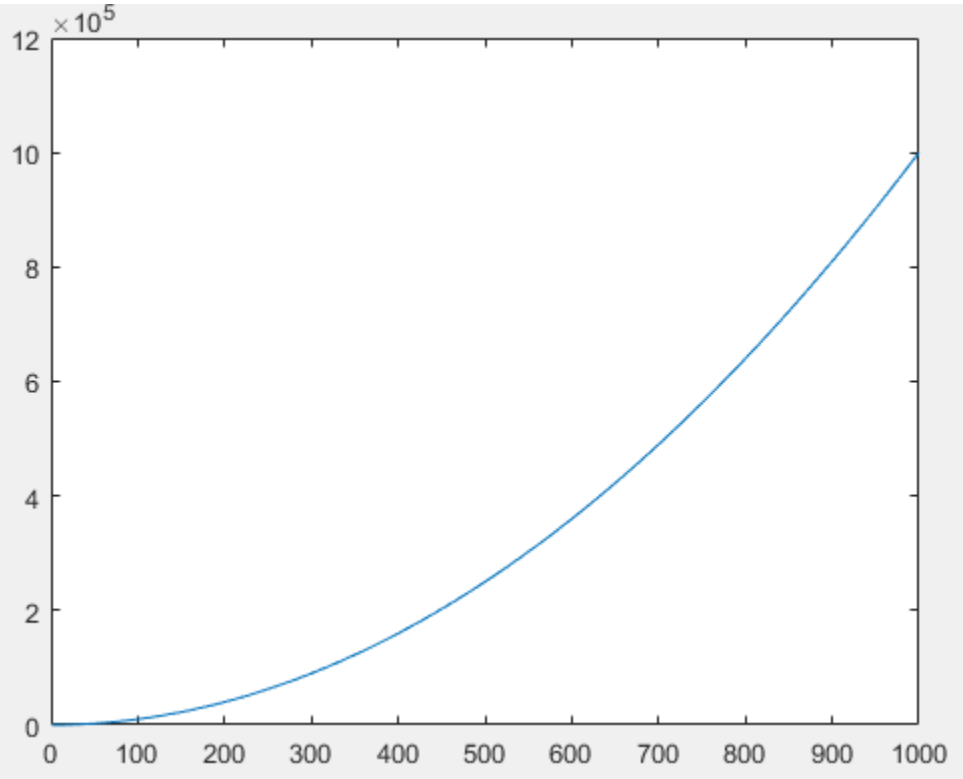
```
%Part b
% Manipulating 5.3 equation so we equate to error term
Error = (f(x+h) - f(x))/h - Q_approx(x)
```

c. Writing the FirstDer() function in Matlab, I wrote it as specified in the assignment. Taking two inputs x and y, I iterated through these valid values up to three points. With these grabbed points I calculated LaGrange values and calculated three outputs at a time. In DriverDer.m I did a similar thing. Getting y values for the anonymous function in a for loop, I used these values to approx. derivatives by calling FirstDer function. Then in another loop I calculated all the actual derivatives, and found the error between approximations and actual derivative values. Below is some snippets of my code from FirstDer() and DriverDer.

```
function [ Output ] = FirstDer( x, y )
%FIRSTDER Accepts set of all inputs x, and the values at those points y

%Set size of output
Output = zeros(1,length(x));

for i = 1:3:length(x)
    %Grabs the x values from x

    %Assure that we do not go out of bounds
    if(i+2 < length(x))
       x0 = x(i);
       x1 = x(i+1);
       x2 = x(i+2);

    %Lagrange Values
    l0 = ((x(i)-x1)*(x(i)-x2))/((x0-x1)*(x0-x2));
    l1 = ((x(i)-x0)*(x(i)-x2))/((x1-x0)*(x1-x2));
    l2 = ((x(i)-x0)*(x(i)-x1))/((x2-x0)*(x2-x1));

    %Compute Outputs three at a time
    Output(i) = y(i) * l0 + y(i) * l1 + y(i) * l2;
    Output(i+1) = y(i+1) * l0 + y(i+1) * l1 + y(i+1) * l2;
    Output(i+2) = y(i+2) * l0 + y(i+2) * l1 + y(i+2) * l2;
    end


end
end
```

```
%% Driver Der File

% Anonymous function for testing
anon_f = @(x) x^2 + 10;

syms f(x)
f(x) = x^2 + 10;

xPlot = zeros(1,1000);
yPlot = zeros(1,1000);
yValues = zeros(1,1000);

%Gives us list of function inputs and outputs
for i=1:1000
    xPlot(i) = i;
    yValues(i) = anon_f(i);
end

%Approx Derivatives
yPlotApprox = FirstDer(xPlot, yPlot);

%Calculating Actual Derivatives
for(i=1:length(yPlot))
    yPlot(i) = abs(diff(f(i),x)-yValues(i)); %Subtracting the derivative from approx
end

plot(xPlot, yPlot)
```

d. Plotting the relative error, this did increase as the number of points increased. Matching my derivation from part b.



The x axis here is the x values for FirstDer(). The y axis is the error of approximation.

2. Problem 2 Quadrature
    a. I generated an approx. for the function -x^2 + 1 after declaring the function f(x) I set x0, x1, and x2 as values within the integral. These were sort of given in assignment

specification, I chose the range -10 to 10 for my integral. LaGrange formulas were setup with LaGrange values x0, x1, and x2.

```
%Lagrange Values
l0 = ((x-x1)*(x-x2))/((x0-x1)*(x0-x2));
l1 = ((x-x0)*(x-x2))/((x1-x0)*(x1-x2));
l2 = ((x-x0)*(x-x1))/((x2-x0)*(x2-x1));
```

After this I used the Polynomial Interpolant equation to derive an approximation.

```
f_approx(x) = f(x0) * l0 + f(x1) * l1 + f(x2) * l2;
```

F(x) is in this equation was the original function I was trying to approximate.

b. The error value for the approximation is the Integral of the function minus the Integral of our approximation. If we got a negative value for this, we would need the absolute value. Since my approximation was simple, I always got an error of 0. Adding more points could change this error value.

```
%Derived Integral for x vs. approx of x
integralF = int(f(x),[-10,10]);
integralFApprox = int(f_approx(x),[-10,10]);

%Error for part b
Error = abs(integralF - integralFApprox)
```

c. Similar to interpolation, ends up being good in the middle of the integral but bad at the function extrema. To alleviate, one solution is to divide [a,b] into n parts, use trapezoidal rule on each part individually.