Robert Heeter
BIOE 391 Numerical Methods – Due 1 February 2022

Problem Set 2 Solutions

1. **Output**
   e = 2.220446e-16
   eps = 2.220446e-16
   2^(-52) = 2.220446e-16

   The value from my algorithm (2.220446e-16, see MATLAB code) is equal to the built-in function *eps*.

2. **Output**
   e = 4.940656e-324
   realmin = 2.225074e-308
   eps*realmin = 4.940656e-324

   log_2(e) = -1074
   log_2(realmin) = -1022
   log2(eps*realmin) = -1074

   The value calculated from my algorithm (4.940656e-324) is smaller than MATLAB's built-in *realmin* value, as shown in the base-2 logarithm calculation. In fact, *eps*realmin* is equal to my value.

3. 3-digit chopping: df/dx = 216,250
   4-digit chopping: df/dx = 2,048,521

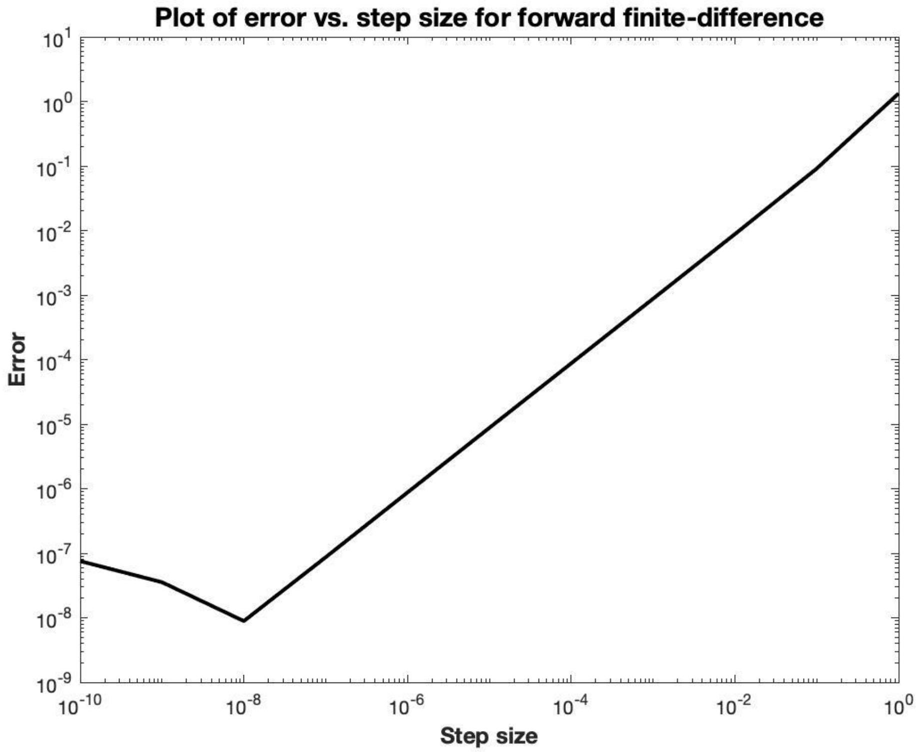
   See my handwritten work at the end of the document.

4. **Output**

Step size:	Finite-diff.:	True error:
1.0000000000	-2.23750000000000	1.3250000000000
0.1000000000	-1.00360000000000	0.0911000000000
0.0100000000	-0.92128509999999	0.0087851000000
0.0010000000	-0.91337535009994	0.0008753500999
0.0001000000	-0.91258750349987	0.0000875034999
0.0000100000	-0.91250875002835	0.0000087500284
0.0000010000	-0.91250087497219	0.0000008749722
0.0000001000	-0.91250008660282	0.0000000866028
0.0000000100	-0.91250000888721	0.0000000088872
0.0000000010	-0.91249996447829	0.0000000355217
0.0000000001	-0.91250007550059	0.0000000755006

   Figure

Robert Heeter
BIOE 391 Numerical Methods – Due 1 February 2022



A minimum in error is reached around a step size of 10^-8 due to decreasing truncation error, after which roundoff error begins to slowly increase the total error again.

5. **Output**

Lagrange polynomial approx. for dy/dx at x = 0: -13.500000
Exact dy/dx at x = 0: -12.000000
Centered finite-difference approx. for dy/dx at x = 0: -12.000000

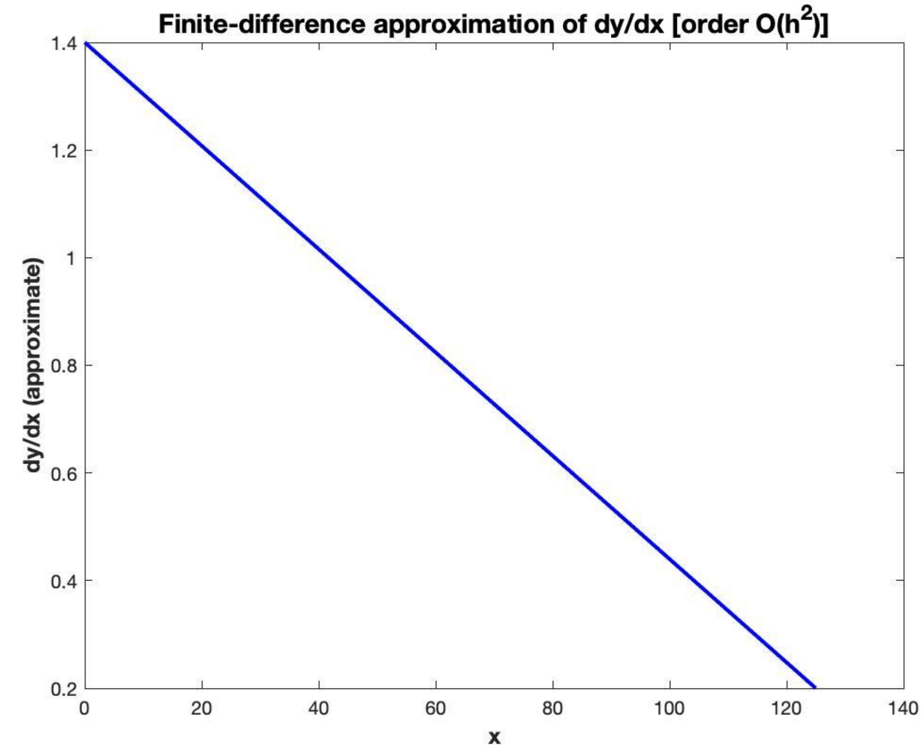
The Lagrange polynomial approximation is not nearly as accurate as the centered finite-difference approximation method in this case. Typically, the Lagrange polynomial approximation is equally as accurate as the centered finite-difference approximation (both are fundamentally the same) given they use the same parameters, but here the centered finite-difference method uses a different set of equally-spaced steps.

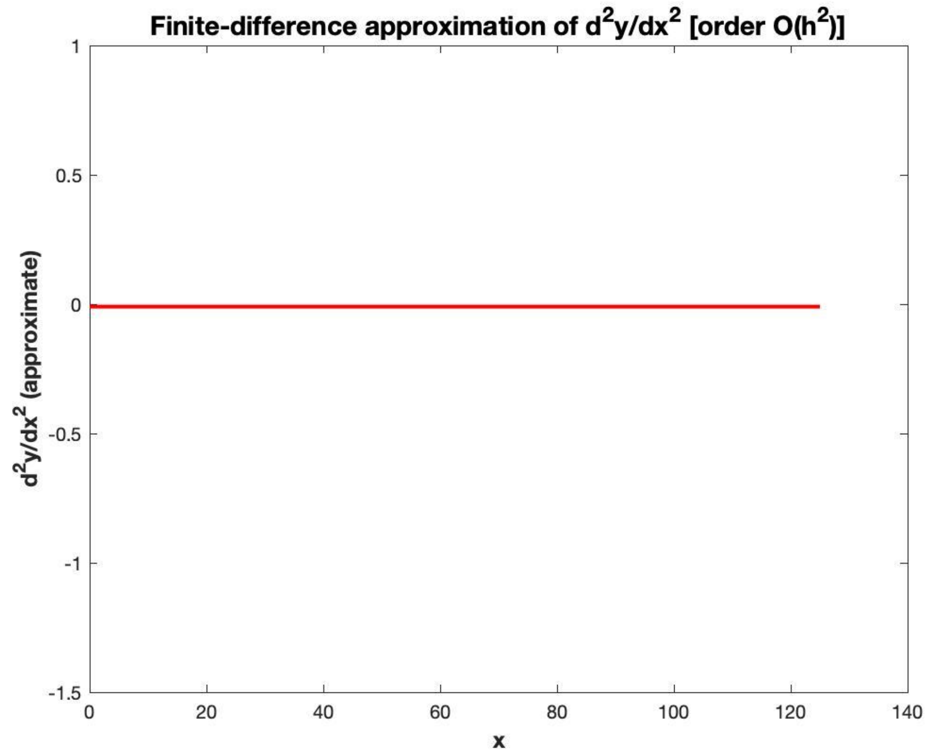
6. **Output**

Time (s):	dy/dt (km/s):	d^2y/dt^2 (km/s^2)
0	1.4000	-0.0096
25.0000	1.1600	-0.0096
50.0000	0.9200	-0.0096
75.0000	0.6800	-0.0096
100.0000	0.4400	-0.0096
125.0000	0.2000	-0.0096

See MATLAB code for function *[dydx, d2ydx2] = diff(e,y)*.

Figures





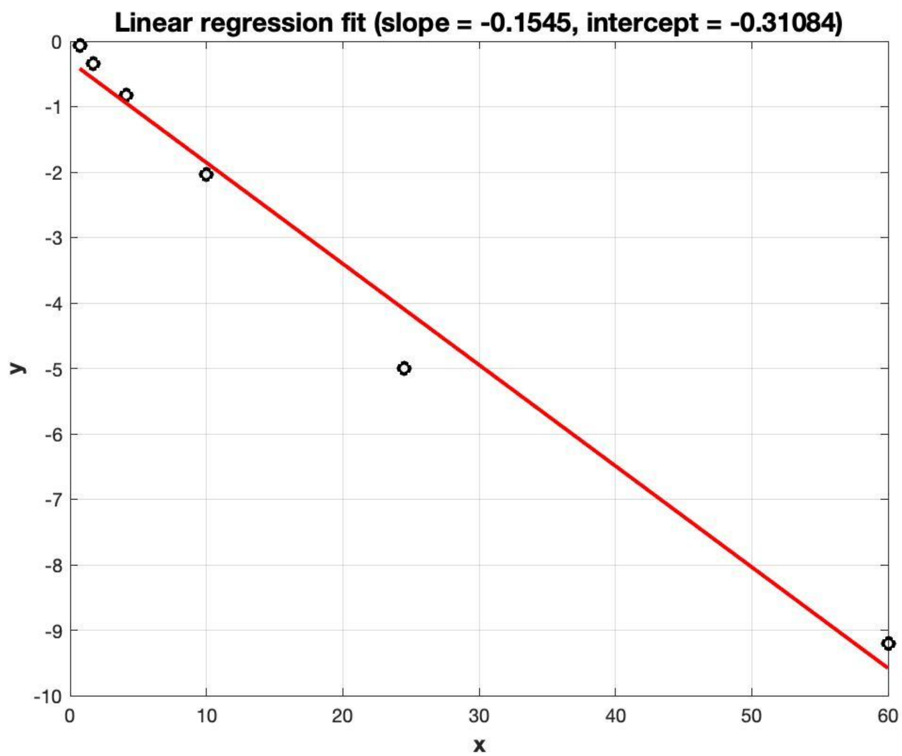
7. **Output**

Time (min): Cooling rate = dT/dt (°C/min):

0	-9.2000
5.0000	-5.0000
10.0000	-2.0400
15.0000	-0.8300
20.0000	-0.3400
25.0000	-0.0600

Constant for Newton's law of cooling [dT/dt = -k(T-T_a)], k = 0.154505 (units are min^-1)

Figure



8. Output

Temp. (K): c_p = dh/dT (J/(kg*K)):

750	1.1493
800	1.1684
900	1.2031
1000	1.2345

Robert Heeter
BIOE 391 Numerical Methods – Due 1 February 2022

Complete MATLAB Code

```
% Robert Heeter
% BIOE 391 Numerical Methods
% HOMEWORK 2 MATLAB SCRIPT

clc, clf, clear, close all

%% P1. PROBLEM 4.4
disp('P1. PROBLEM 4.4');

e = 1;
while (1+e) > 1
    e = e/2;
end
e = 2*e;

% Display results and compare to other values
fprintf('e = %d\n', e);
fprintf('eps = %d\n', eps);
fprintf('2^(-52) = %d\n\n', (2^(-52)));


%% P2. PROBLEM 4.5
disp('P2. PROBLEM 4.5');

e = 1;
while ((e/2)-0) ~= 0
    e = e/2;
end

% Display results and compare to other values
fprintf('e = %d\n', e);
fprintf('realmin = %d\n', realmin);
fprintf('eps*realmin = %d\n\n', (eps*realmin));

% Base-2 logarithms
fprintf('log_2(e) = %d\n', log2(e));
fprintf('log_2(realmin) = %d\n', log2(realmin));
fprintf('log2(eps*realmin) = %d\n\n', log2(eps*realmin));


%% P3. PROBLEM 4.8
disp('P3. PROBLEM 4.8');

% No MATLAB code for this problem


%% P4. PROBLEM 4.23
disp('P4. PROBLEM 4.23');

ff = @(x) -0.1.*x.^4-0.15.*x.^3-0.5.*x.^2-0.25.*x+1.2; % original eqn.
df = @(x) -0.4.*x.^3-0.45.*x.^2-x-0.25; % derivative eqn.
[~] = fwd_diff(ff,df,0.5,11,true,true); % use fwd_diff function (below) for forward finite-difference
and display results and plot
disp(' ');


%% P5. PROBLEM 21.6
disp('P5. PROBLEM 21.6');

y = @(x) 2.*x.^4 - 6.*x.^3 - 12.*x - 8; % original eqn.
dydx = @(x) 8.*x.^3 - 18.*x.^2 - 12; % derivative eqn.

xi = [-0.5;1;2]; % x-points
```

```
yi = y(xi); % corresponding y-points

dydx_aprx_lagr = lagr_diff(0,xi,yi); % use lagr_diff function (below) for Lagrange equation (21.21)
with x- and y-points

dydx_exact = dydx(0); % use derivative eqn. for exact value
dydx_aprx_cen_diff = cen_diff(y,dydx,0,10,false,false); % use cen_diff function (below) for centered
finite-difference and suppress results/plot

% Display results for each method
fprintf('Lagrange polynomial approx. for dy/dx at x = 0: %f\n', dydx_aprx_lagr);
fprintf('Exact dy/dx at x = 0: %f\n', dydx_exact);
fprintf('Centered finite-difference approx. for dy/dx at x = 0: %f\n\n', dydx_aprx_cen_diff);


%% P6. PROBLEM 21.10
disp('P6. PROBLEM 21.10');

t = [0:25:125]'; % time interval (s)
y = [0 32 58 78 92 100]'; % corresponding y-values (km)

[dydt, d2ydt2] = diff(t,y,true); % use diff function (below) to find derivative with finite-difference
[O(h^2)]

% Display results
disp('  Time:       dy/dt:   d^2y/dt^2:')
disp([t,dydt,d2ydt2]);


%% P7. PROBLEM 21.28
disp('P7. PROBLEM 21.28');

t = [0:5:25]'; % time interval (min)
T = [80 44.5 30.0 24.1 21.7 20.7]'; % corresponding temperatures (Â°C)
T_a = 20; % ambient temperature (Â°C)
deltaT = T-T_a;

[dTdt, ~] = diff(t,T,false); % use diff function (below)

% Display results
disp('   Time:     Cooling rate = dT/dt:')
disp([t,dTdt]);

% Find cooling constant (k) with linear regression and plot
[a,r2] = linregr(deltaT,dTdt); % use linregr function (below) for regression
fprintf('Constant for Newton''s law of cooling [dT/dt = -k(T-T_a)], k = %f\n\n', -1*a(1));


%% P8. PROBLEM 21.33
disp('P8. PROBLEM 21.33');

T = [750 800 900 1000]'; % temperatures (K)
h_kJkmol = [29629 32179 37405 42769]'; % enthalpy (in kJ/kmol)

M_carb = 12.011; % weight of carbon (g/mol)
M_oxy = 15.9994; % weight of oxygen (g/mol)

h = h_kJkmol./(M_carb + (2*M_oxy)); % convert enthalpy to units of kJ/kg

dhdT = zeros(4,1);
dhdT(1) = lagr_diff(T(1),T(1:3),h(1:3)); % use lagr_diff function (below) for Lagrange equation (21.21)
with nearby points
dhdT(2) = lagr_diff(T(2),T(1:3),h(1:3));
dhdT(3) = lagr_diff(T(3),T(2:4),h(2:4));
dhdT(4) = lagr_diff(T(4),T(2:4),h(2:4));
```

```
% Display results
fprintf('  Temp.:   c_p = dh/dT: \n');
fprintf(' %5.0f     %6.4f\n',[T(:),dhdT(:)]');
disp(' ');


%% Additional Functions

function [dfdx] = fwd_diff(func,dfunc,x,n,maketable,makeplot)
% ABOUT: Forward finite-difference function, adapted from provided .m file
% in example 4.5. Equation is also in Figure 21.3.
% INPUTS: func = function; dfunc = derivative of func; x = point to
% calculate derivative at; n = iterations; maketable = boolean to display
% table of iterations; makeplot = boolean to make plot of error vs. step
% size
% OUTPUTS: dfdx = final derivative value at x after n iterations

format long
dftrue = dfunc(x);

h = 1; % step size
H = zeros(n,1); % preallocate
D = zeros(n,1);
E = zeros(n,1);

for i = 1:n
    H(i) = h;
    D(i) = (func(x+h) - func(x))/(h); % forward finite-difference formula
    E(i) = abs(dftrue - D(i)); % true error
    h = h/10;
end

% Display table of iterations?
if maketable == true
    L = [H(:), D(:), E(:)]';
    fprintf('  Step size:     Finite-diff.:        True error:\n');
    fprintf('%14.10f   %16.14f   %16.13f\n',L);
end

% Make plot of error vs. step size?
if makeplot == true
    figure
    loglog(H,E,'-k','LineWidth',2);
    xlabel('Step size','FontSize',12,'FontWeight','bold');
    ylabel('Error','FontSize',12,'FontWeight','bold');
    title('Plot of error vs. step size for forward
finite-difference','FontSize',14,'FontWeight','bold');
end

dfdx = D(n);
format short

end


function [dfdx] = cen_diff(func,dfunc,x,n,maketable,makeplot)
% ABOUT: Forward finite-difference function, adapted from provided .m file
% in example 4.5. Equation is also in Figure 21.5.
% INPUTS: func = function; dfunc = derivative of func; x = point to
% calculate derivative at; n = iterations; maketable = boolean to display
% table of iterations; makeplot = boolean to make plot of error vs. step
% size
% OUTPUTS: dfdx = final derivative value at x after n iterations

format long
dftrue = dfunc(x);
```

Robert Heeter
BIOE 391 Numerical Methods – Due 1 February 2022

```
h = 1; % step size
H = zeros(n,1); % preallocate
D = zeros(n,1);
E = zeros(n,1);

for i = 1:n
    H(i) = h;
    D(i) = (func(x+h) - func(x-h))/(2*h); % centered finite-difference formula
    E(i) = abs(dftrue - D(i)); % true error
    h = h/10;
end

% Display table of iterations?
if maketable == true
    L = [H(:), D(:), E(:)]';
    fprintf('  Step size:   Finite-diff.:      True error:\n');
    fprintf('%14.10f %16.14f %16.13f\n',L);
end

% Make plot of error vs. step size?
if makeplot == true
    figure
    loglog(H,E,'-k','LineWidth',2);
    xlabel('Step size','FontSize',12,'FontWeight','bold');
    ylabel('Error','FontSize',12,'FontWeight','bold');
    title('Plot of error vs. step size for forward
finite-difference','FontSize',14,'FontWeight','bold');
end

dfdx = D(n);
format short

end


function [dydx,d2ydx2] = diff(x,y,makeplot)
% ABOUT: Finite-difference function for a set of points using forward,
% centered, and backward finite-difference formulas for the order O(h^2).
% Equations from Figures 21.3-21.5.
% INPUTS: x = set of x-points, y = corresponding set of y-points, makeplot
% = boolean to display plot of first and second derivatives vs. x
% OUTPUTS: dydx = vector of derivatives corresponding to each element of x;
% d2ydx2 = vector of second derivatives corresponding to each element of x;

format long

x = x(:); % set to column vectors
y = y(:);

% Check inputs are valid
if length(x) ~= length(y)
    error('Input vectors of independent and dependent variables are different lengths.');
end
if length(x) < 4
    error('Input vectors have less than 4 values.');
end

h = x(2)-x(1); % step size
for i = 2:length(x)
    if (x(i)-x(i-1) ~= h) || (h == 0)
        error('Values for indepndent variable are not equally spaced and non-zero.');
    end
end

dydx = zeros(size(x)); % preallocate
```

```
d2ydx2 = zeros(size(x));

% Forward finite-difference for first x-value
dydx(1) = ((-1*y(3))+(4*y(2))-(3*y(1)))/(2*h);
d2ydx2(1) = ((-1*y(4))+(4*y(3))-(5*y(2))+(2*y(1)))/(h^2);

% Centered finite-difference for middle x-values
for i = 2:(length(x)-1)
    dydx(i) = (y(i+1)-y(i-1))/(2*h);
    d2ydx2(i) = (y(i+1)-(2*y(i))+y(i-1))/(h^2);
end

% Backward finite-difference for final x-value
n = length(x);
dydx(n) = ((3*y(n))-(4*y(n-1))+y(n-2))/(2*h);
d2ydx2(n) = ((2*y(n))-(5*y(n-1))+(4*y(n-2))-y(n-3))/(h^2);

% Display plot of first and second derivatives vs. x?
if makeplot == true
    figure
    plot(x,dydx,'-b','LineWidth',2);
    xlabel('x','FontSize',12,'FontWeight','bold');
    ylabel('dy/dx (approximate)','FontSize',12,'FontWeight','bold');
    title('Finite-difference approximation of dy/dx [order O(h^2)]','FontSize',14,'FontWeight','bold');

    figure
    plot(x,d2ydx2,'-r','LineWidth',2);
    xlabel('x','FontSize',12,'FontWeight','bold');
    ylabel('d^2y/dx^2 (approximate)','FontSize',12,'FontWeight','bold');
    title('Finite-difference approximation of d^2y/dx^2 [order
O(h^2)]','FontSize',14,'FontWeight','bold');
end

format short

end


function [a, r2] = linregr(x,y)
% ABOUT: Linear regression function, adapted from provided .m file provided
% on Canvas. Uses least squares fit by solving normal equations.
% INPUTS: x = set of x-points, y = corresponding set of y-points
% OUTPUTS: a(1) = slope; a(2) = intercept; r2 = coefficient of
% determination

x = x(:); % set to column vectors
y = y(:);

n = length(x);

% Check inputs are valid
if length(y) ~= n
    error('Input vectors of x and y variables are different lengths.');
end

% Solve normal equations
sx = sum(x); sy = sum(y);
sx2 = sum(x.*x); sxy = sum(x.*y); sy2 = sum(y.*y);

a(1) = (n*sxy-sx*sy)/(n*sx2-sx^2);
a(2) = sy/n-a(1)*sx/n;

r2 = ((n*sxy-sx*sy)/sqrt(n*sx2-sx^2)/sqrt(n*sy2-sy^2))^2;

% Create plot of data and best fit line
xp = linspace(min(x),max(x),2);
```

Robert Heeter

BIOE 391 Numerical Methods – Due 1 February 2022

```
yp = a(1)*xp+a(2);

figure
plot(x,y,'ok',xp,yp,'-r','LineWidth',2);
xlabel('x','FontSize',12,'FontWeight','bold');
ylabel('y','FontSize',12,'FontWeight','bold');
title(['Linear regression fit (slope = ',num2str(a(1)),', intercept = ', num2str(a(2)),
')'],'FontSize',14,'FontWeight','bold');
grid on

end


function [dydx] = lagr_diff(xval,x,y)
% ABOUT: Derivative of Lagrange polynomial fit to three unequally-spaced
% points to calculate derivative. Equation from Eqn. 21.6.
% INPUTS: xval = point to calculate derivative at; x = x-points; y =
% corresponding y-points
% OUTPUTS: dydx = final derivative value at point xval

dydx_lagr = @(x_p) y(1)*((2*x_p)-x(2)-x(3))/((x(1)-x(2))*(x(1)-x(3))) +
y(2)*((2*x_p)-x(1)-x(3))/((x(2)-x(1))*(x(2)-x(3))) +
y(3)*((2*x_p)-x(1)-x(2))/((x(3)-x(1))*(x(3)-x(2)));
dydx = dydx_lagr(xval);

end
```

Robert Heeter
BIOF 391
2/1/22

Problem Set #2

3. $f(x) = 1/(1-3x^2)$

$f'(x) = \frac{6x}{(1-3x^2)^2} \rightarrow f'(0.577) = 2,352,911$

3-digit chopping:

$6x = 6(0.577) = 3.462 \longrightarrow 3.46$

$x^2 = 0.33293 \longrightarrow 0.332$

$3x^2 = 0.996$

$1-3x^2 = 0.004 \longrightarrow \frac{6x}{(1-3x^2)^2} = \boxed{216250} \rightarrow 90.8\%$ error

4-digit chopping:

$6x = 6(0.577) = 3.462 \rightarrow 3.462$

$x^2 = 0.33293 \rightarrow 0.3329$

$3x^2 = 0.9987$

$1-3x^2 = 0.0013 \longrightarrow \frac{6x}{(1-3x^2)^2} = \boxed{2048521} \rightarrow 12.9\%$ error

We have difficulty evaluating this function at $x = 0.577$ because it has a vertical asymptote at $\sqrt{3}/3 = 0.57735$, meaning its value approaches extreme values near the asymptote; with chopping, small errors in the calculation are magnified greatly as $f'(x)$ has such a steep slope near the asymptote.