Table of Contents

..... 1
problem 1 1
problem 2 2
problem 3 3
problem 4 5
problem 5 6


```
% INSTRUCTIONS:
% 1) save this file into the directory that holds your work
% 2) make sure it runs without error
% 3) publish this file to PDF.
% 4) There should then be a PDF file in the subdirectory 'html'
%    that you can upload to TurnItIn.
%    PLEASE DOUBLE-CHECK it has everything you need!!

% prepare to run...
clear all  % clear out all variables
close all  % close any open figures
```

problem 1

```
type('GradientUneven')
```

```
function deriv = GradientUneven(y,x)
%{
takes in 2 vectors representing values y(x) and
calculates deriv of function using forward, back,
and centered diff based on index and stores it
into a vector "deriv"

Alyssa Rose    HW5   2-28-18
%}
%checks if lengths are the same
if length(y)~=length(x)
    deriv = NaN
    return
end
%preallocates, calcs. forward diff of 1st element
m = length(x);
deriv = zeros(1,m);
deriv(1) = (y(2) - y(1))/(x(2)-x(1));
%calcs. backward diff of last element
deriv(end) = (y(end) - y(end-1))/(x(end)-x(end-1));
%calculates centered difference
for z = 2:m-1
    deriv(z) = (y(z+1)-y(z-1))/((x(z+1)-x(z-1)));
end
return
```

problem 2

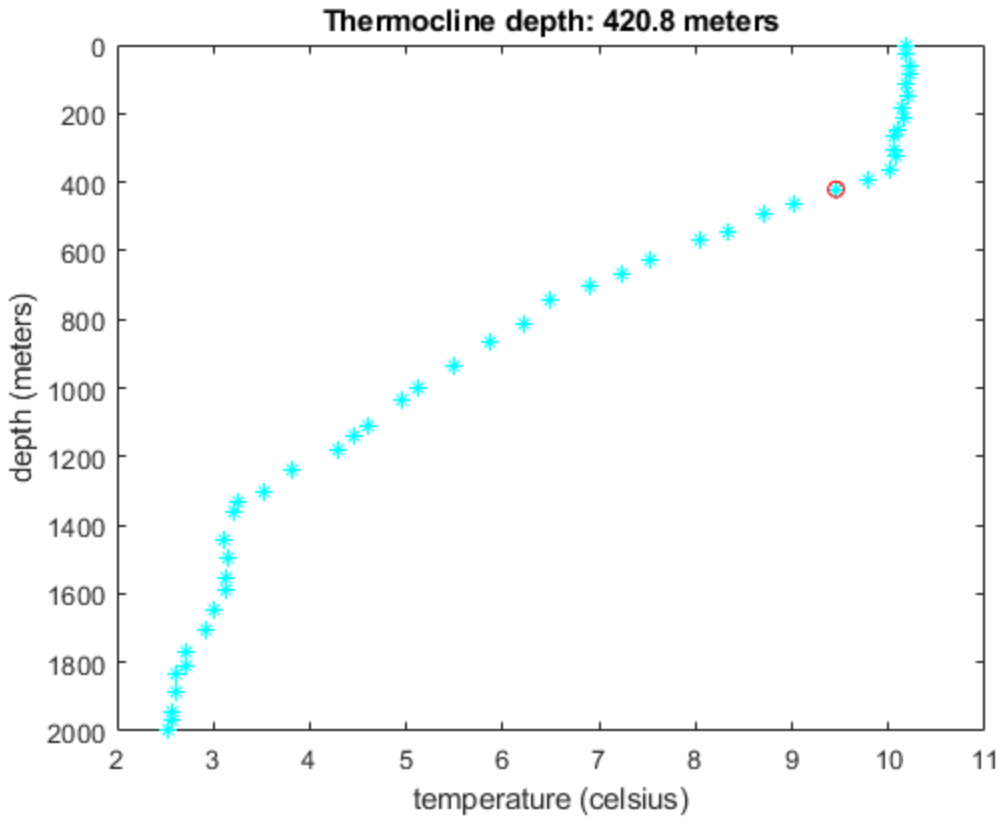
```
type('RmsError')
type('TestUnevenDeriv')
TestUnevenDeriv


function err = RmsError(y1,y2)
%{
Takes in 2 output value vectors (the derivative)
and determines the error in difference between the 2

Alyssa Rose    HW5   2-28-18
%}
z = length(y1);
w = length(y2);

%checks if same length and vector at least size 1
if z==w && z >=1
    %computes the error between the 2 vectors from first
    %element until last one using equation
    err = sqrt((1/z).*(sum((y1(1:z)-y2(1:z)).^2)));
else
    err = NaN;
    return
end

%{
script evaluating GradientUneven vs gradient function

Alyssa Rose    HW5   2-28-18
%}

%creates equation evaluated respect to x
n = @(x) 3*x.^4 - 1.1*x.^2 + 4;
%sets up vector of 41 x values w/ 0.1 spacing
x = 1:0.1:5;
%stores equation values at each x into y
y = n(x);
%creates derivatives using the 2 seperate functions
dz = gradient(y)/0.1;
dg = GradientUneven(y,x);
%calcs. error between 2 and prints out to 20 decimals
fprintf('Error of even spacing: %.20f\n', RmsError(dg,dz))


%uneven spacing
a = [0.1 0.10017 0.1003 0.1004 0.10042];
%creates original equation and deriv. equation
normEq = @(x) 3*x.^4 - 1.1*x.^2 + 4;
derivEq = @(x) 12*x.^3 - 2.2*x;
```

```
%finds values of equations at each x value
b = normEq(a);
c = derivEq(a);
%calculates deriv. with GradientUneven and compares
%to actual deriv. equation
derivNorm = GradientUneven(b,a);
fprintf('Error of uneven spacing: %.20f\n', RmsError(derivNorm,c))


%uneven w/ smaller spacing
aa = [0.1 0.10000017 0.1000003 0.1000004 0.10000042];
normEq = @(x) 3*x.^4 - 1.1*x.^2 + 4;
derivEq = @(x) 12*x.^3 - 2.2*x;
b = normEq(aa);
c = derivEq(aa);
derivNorm = GradientUneven(b,aa);
fprintf('Error of smaller spacing: %.20f\n', RmsError(derivNorm,c))



%{
The error cannot be exactly 0 due to rounding errors
and the precision of the numbers. Double precision can
only give about 15 to 16 decimal points of error, which
is less than the 20 being printed out. Since the #'s
aren't exact and machine epsilon has to be accounted for
(precision of MatLab), then it makes sense that the error
wouldn't be exactly zero.

The smaller the spacing is, the smaller the error will
be. Since the deriv (or tangent line speaking mathematically)
becomes smaller, the approximation becomes more precise.
%}
Error of even spacing: 0.00000000000116293464
Error of uneven spacing: 0.00000837968252253427
Error of smaller spacing: 0.00000008052932376817
```

problem 3

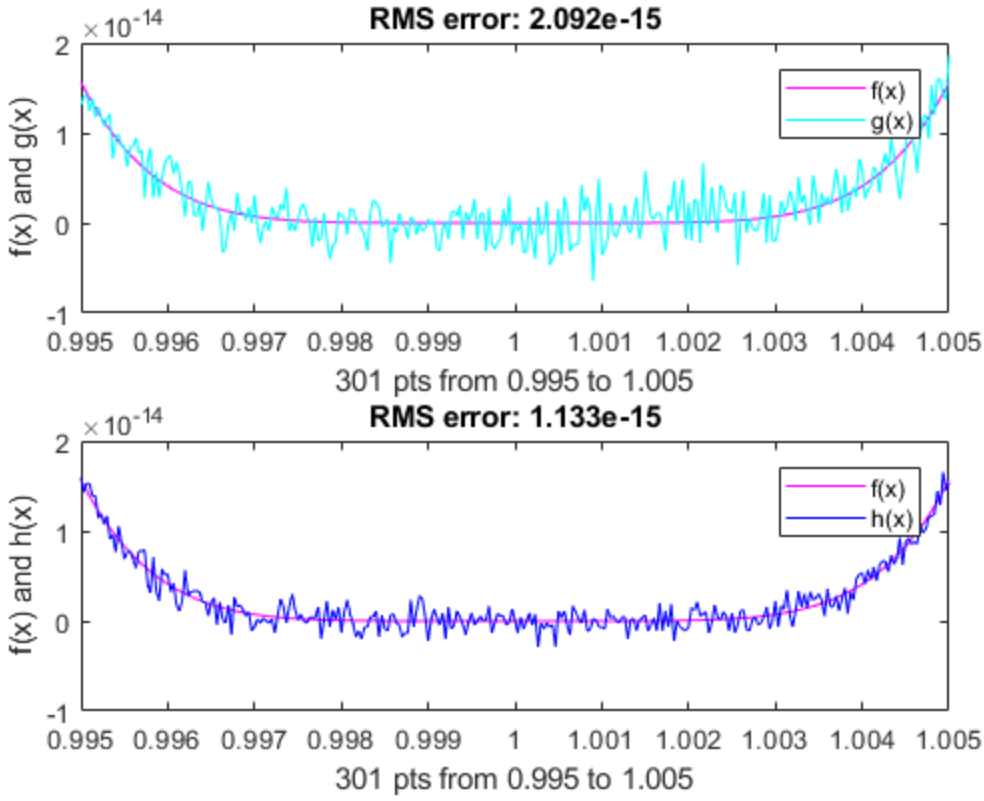
```
type('FindThermocline')
FindThermocline


%{
Using depth (meters) and temp(celsius), the thermocline
(area where temp change is greatest) can be found and
is plotted as a red dot with the title telling the depth
at which the thermocline occurs

Alyssa Rose    HW5   2-28-18
%}

load thermoclineData.mat
```

```
%preallocates
derivThermo = zeros(1,length(depthMeters));
%finds derivative
derivThermo = GradientUneven(tempCelcius, depthMeters);
[thermocline,t] = min(derivThermo)
%t is index # where thermocline occurs
%t is used to find temp and depth @thermocline
x = tempCelcius(t);
y = depthMeters(t);

%plots graph and thermocline as red dot
plot(tempCelcius,depthMeters,'c*', x, y,'ro')
set(gca,'Ydir','reverse');
xlabel('temperature (celsius)')
ylabel('depth (meters)')
title(sprintf('Thermocline depth: %.1f meters',y))

thermocline =

   -0.0117


t =

    15
```



problem 4

```
type('EquivFormulas')
EquivFormulas


%{
Rewrites same function in multiple ways using Symbolic
Math toolbox

Alyssa Rose    HW5   2-28-18
%}

syms x
f = (1 - x)^6;
g = expand(f);
h = horner(f);
%prints to screen
pretty(f)
pretty(g)
pretty(h)

%converts symbolic to anonymous functions
ff =@(x) (1 - x).^6;
gg = matlabFunction(g);
hh = matlabFunction(h);
x = linspace(0.995, 1.005, 301);
ff = ff(x);
gg = gg(x);
hh = hh(x);
%compares f(x) and g(x) graphs and finds error
figure
subplot(2,1,1)
plot(x,ff,'m-',x,gg,'c-')
xlabel('301 pts from 0.995 to 1.005')
ylabel('f(x) and g(x)')
title(sprintf('RMS error: %.3e',RmsError(ff,gg)))
legend('f(x)','g(x)')
%compares f(x) and h(x) graphs and finds error
subplot(2,1,2)
plot(x,ff,'m-',x,hh,'b-')
xlabel('301 pts from 0.995 to 1.005')
ylabel('f(x) and h(x)')
title(sprintf('RMS error: %.3e',RmsError(ff,hh)))
legend('f(x)','h(x)')
       6
(x - 1)

 6      5       4       3       2
x  - 6 x  + 15 x  - 20 x  + 15 x  - 6 x + 1

x (x (x (x (x (x - 6) + 15) - 20) + 15) - 6) + 1
```

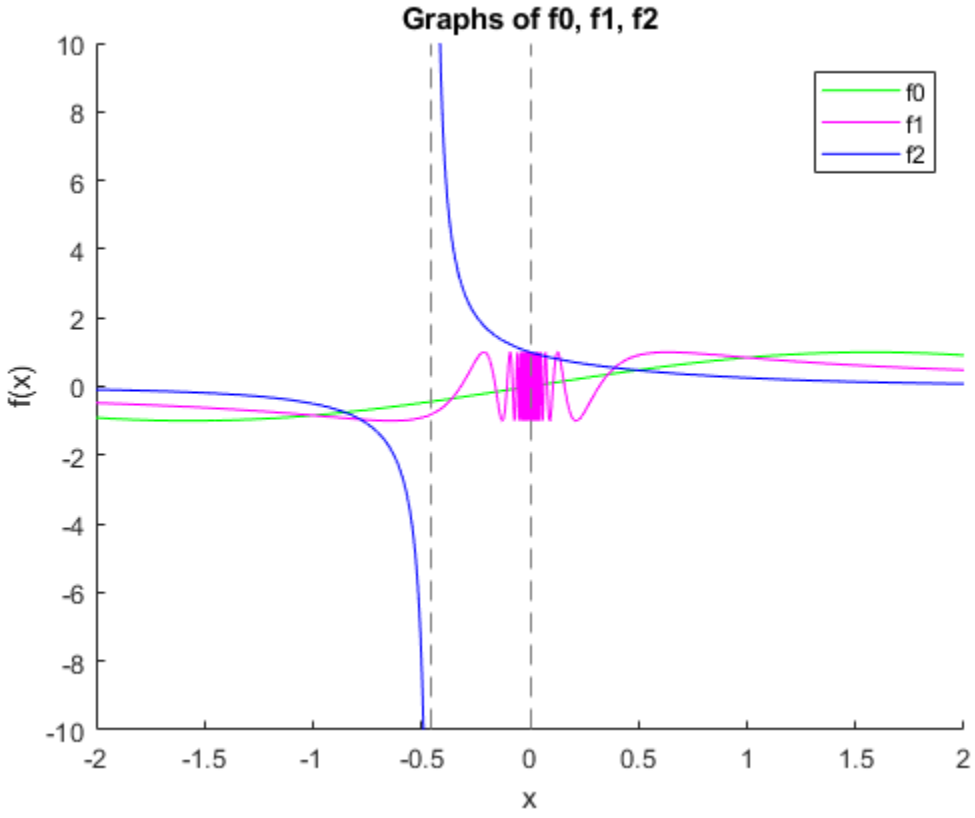


problem 5

```
type('IntegrationProblem')
IntegrationProblem


%{
Computes the integrals of 3 functions

Alyssa Rose    HW5   2-28-18
%}

%creates symbolic function respect to x
syms x
f0 = sin(x);
f1 = sin(1./x);
f2 = 1./(1 + 2*x + x.^3);
%plots all 3 graphs on one plot
figure
xlabel('x')
ylabel('f(x)')
title('Graphs of f0, f1, f2')
hold on
fplot(f0,[-2,2],'g-')
fplot(f1,[-2,2],'m-')
fplot(f2,[-2,2],'b-')
```

```
legend('f0','f1','f2')
hold off
%finds integral from x=0.05 to 1
intf0 = int(f0,x,0.05,1);
intf1 = int(f1,x,0.05,1);
intf2 = int(f2,x,0.05,1);
%prints to screen
pretty(intf0)
pretty(intf1)
pretty(intf2)
%prints numerical value of integral to double precision #
fprintf('integral f0 = %0.4f\n', double(intf0))
fprintf('integral f1 = %0.4f\n', double(intf1))
fprintf('integral f2 = %0.4f\n\n', double(intf2))

%{
MatLab was not able to give an analytical solution
since integration of f2 would've required an x^2 term
in the integral by the rules of u-substitution of calculus.
Since the term wasn't present, there was no way for
the integral to be expressed analytically.
%}

A = @(x) sin(x);
B = @(x) sin(1./x);
C = @(x) 1./(1 + 2*x + x.^3);

intA = integral(A,0.05,1);
intB = integral(B,0.05,1);
intC = integral(C,0.05,1);

fprintf('integral A = %0.4f\n', double(intA))
fprintf('integral B = %0.4f\n', double(intB))
fprintf('integral C = %0.4f\n', double(intC))
   /  1 \
cos| -- | - cos(1)
   \ 20 /

                                    sin(20)
cosint(20) - cosint(1) + sin(1) - -------
                                      20

  1
  /
 |        1
 |  ------------ dx
/    3
  1 x  + 2 x + 1
 --
 20

integral f0 = 0.4584
integral f1 = 0.5028
integral f2 = 0.4682
```

```
integral A = 0.4584
integral B = 0.5028
integral C = 0.4682
```



Published with MATLAB® R2017b