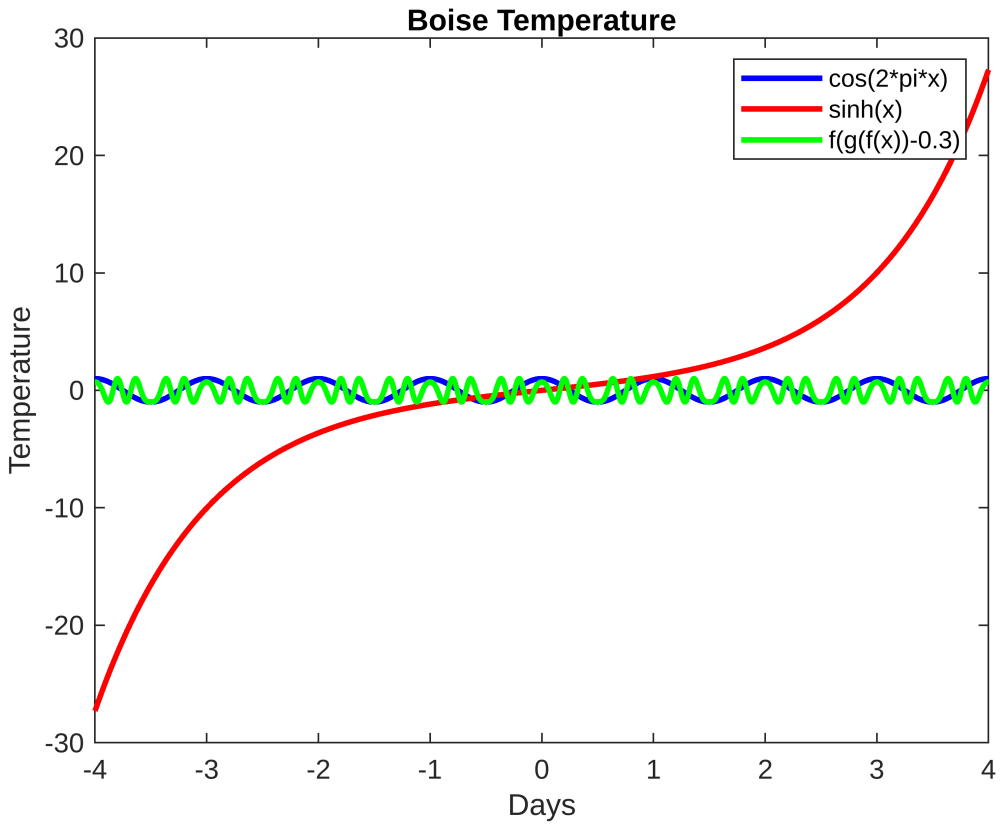
Problem 1

1. (Simple anonymous function handle; plotting, 10 points) Use anonymous function to define the functionsf (x) = cos(2πx), g(x) = sinh(x), h(x) = f (g(f (x)) − 0.3)
2. You should use f and g in your code for defining h and not determine the expression for h byhand.(a) Plot the function f, g, and h over the domain [−4,4] on the same figure. Use the Mat- lab command linspace to construct a sequence of 2000 equally spaced points in the given domain to make the plots.(b) Add a title, axis labels, and a legend to your plot. (c) Compute h(2.15) and report this in your write-up.

```
f = @(x) cos(2*pi*x);% function handles
g = @(x) sinh(x);
h = @(x) f(g(f(x))-0.3);

x = 2.15;% Compute h(2.15)
result = h(x);
fprintf('h(2.15) = %.4f\n', result); % Display the result
```

```
h(2.15) = -0.4384
```

```
new_Aprox = 2000;% Aproximation
x_val = linspace(-4,4,new_Aprox);% x-values

yf = f(x_val);% y-values
yg = g(x_val);
yh = h(x_val);

plot(x_val,yf,'b','LineWidth',2);% plot figures
hold on;
plot(x_val,yg,'r','LineWidth',2);
hold on;
plot(x_val,yh,'g','LineWidth',2);


title('Boise Temperature')
ylabel('Temperature')
xlabel('Days')
legend('cos(2*pi*x)','sinh(x)','f(g(f(x))-0.3)')
```



Problem 2

1. (Simple vectorization, 15 points) Create two arrays x and y, whose entries are defined as xi = 3i, i=1,2,3,4,5.yj = (j+1)/4, j=1,2,3,4,5.Then, compute the squared sum of these two vectors in two different ways. First, use a forloop to construct the vector z aszi =(xi −yi)2, i=1,2,3,4,5.Second, “vectorize” this statement so that no for loop is used at all. This should require one line of Matlab code. Include the output in your written work from both techniques and verify that they are the same.This problem illustrates one of the most powerful features of Matlab . Many complicated expressions can be vectorized to create code that is faster and more compact than the equivalent version using loops required for other compiled languages.

```
% for loop
i = 1:5; x = 3*i; %Define the values of x and y
j = 1:5; y = (j+1)/4;
disp('Array x'),disp(x); %Display the arrays x and y
```

```
Array x
     3     6     9    12    15
```

```
disp('Array y'),disp(y);
```

```
Array y
   0.500000000000000   0.750000000000000   1.000000000000000   1.250000000000000   1.500000000000000
```

```
for k=1:5
    z1 = (x(k)-y(k)).^2; %squared sum of these two vector with for-loop
end

z2 = (x - y).^2 %squared sum of two vectors
```

```
z2 = 1×5
10^2 ×
   0.062500000000000   0.275625000000000   0.640000000000000   1.155625000000000 ···
```

Problem 3

1. (Plotting and simple statistics of data stored in a file, 20 points) Download the data file BoiseTempJuly2022.csv from the course canvas site (or from slack) and save it in the folder you are using to compile this homework assignment. The file contains the maximum, minimum, and average daily temperatures for July 2022, as recorded at the Boise airport. You can load the file into Matlab using the following: X = load('BoiseTempJuly2022.csv');

This will store the data in the matrix X with the values in each row corresponding to the day of the month (1-31), the maximum temperature, the minimum temperature, and the mean temperature (all in Fahrenheit).

1. (a) Plot the maximum, minimum, and average temperatures for the whole month on one plot. Include appropriate labels and a legend.
2. (b) Use the Matlab functions mean and std to compute the mean and standard deviation of the minimum temperature and report these in your document.
3. (c) Use the Matlab functions min and max to determine the maximum average temperature over the month and what day it occurred.

Use help in Matlab to see how these functions work.

```
X = load('BoiseTempJuly2022.csv')% loading the data
```

```
X = 31×4
10^2 ×
   0.010000000000000   0.940000000000000   0.610000000000000   0.775000000000000
   0.020000000000000   0.950000000000000   0.620000000000000   0.785000000000000
   0.030000000000000   0.880000000000000   0.640000000000000   0.760000000000000
   0.040000000000000   0.820000000000000   0.600000000000000   0.710000000000000
   0.050000000000000   0.910000000000000   0.610000000000000   0.760000000000000
   0.060000000000000   0.920000000000000   0.620000000000000   0.770000000000000
   0.070000000000000   0.950000000000000   0.650000000000000   0.800000000000000
   0.080000000000000   0.960000000000000   0.620000000000000   0.790000000000000
   0.090000000000000   0.940000000000000   0.610000000000000   0.775000000000000
```

```
    0.100000000000000    0.890000000000000    0.630000000000000    0.760000000000000
      ⋮
      ⋮
```

```
%Plot the maximum, minimum, and average temperatures for the whole month.
Maximum = max(X(:,2))
```

```
Maximum =
   106
```

```
Minimum = min(X(:,3))
```

```
Minimum =
    57
```

```
Average = mean(X(:,4))
```

```
Average =
  80.741935483870961
```

```
Mean = mean(X(:,3))% mean
```

```
Mean =
  64.419354838709680
```

```
Standard_Deviation = std(X(:,3))% Standard Deviation
```

```
Standard_Deviation =
   4.364815334378513
```

```
Max_Average = max(X(:,4))
```

```
Max_Average =
    90
```

```
Min_Average = min(X(:,4))
```

```
Min_Average =
    71
```

```
daysOfMonth = X(:,1);
dailyTemperatures = X(:,4)
```

```
dailyTemperatures = 31×1
  77.500000000000000
  78.500000000000000
  76.000000000000000
  71.000000000000000
  76.000000000000000
  77.000000000000000
  80.000000000000000
  79.000000000000000
  77.500000000000000
  76.000000000000000
```

⋮

```
AverageTemperatures = mean(dailyTemperatures,4);

% maximum

[maxAverageTemperature,maxAverageTemperatureDay] = max(AverageTemperatures);

disp(['Max Average Temp over month is ',num2str(maxAverageTemperature), '
degrees F.']);
```

```
Max Average Temp over month is 90 degrees F.
```

```
disp(['Occures on day ',num2str(maxAverageTemperatureDay),' of the month']);
```

```
Occures on day 31 of the month
```

```
% minimum

[minAverageTemperature,minAverageTemperatureDay] = min(AverageTemperatures);

disp(['Min Average Temp over month is ',num2str(minAverageTemperature), '
degrees F.']);
```

```
Min Average Temp over month is 71 degrees F.
```

```
disp(['Occures on day ',num2str(minAverageTemperatureDay),' of the month']);
```

```
Occures on day 4 of the month
```

Problem 4

1. (Theconstante,15points)Themathematicalconstante=2.71828182845904523536028 74...,also known as Euler's number, can be defined ase= lim(1+h)1/h.h→0If you fix h to be some small number, then we may expect to be able to approximate e ase ≈ (1 + h)1/h.
2. (a) Determine a value of h so that the approximate value you obtain has an absolute relative error less than 5.0 · 10−9. You can use the Matlab expression exp(1) to test your approximate solution against.
3. (b) Can you find a value of h so that the approximation has an absolute relative error less than 1 · 10−15?
4. (c) What happens to the approximation as h goes to zero? Make a plot of the relative error vs. h to help you answer this questions.

```
tolerance1 = 5.0e-9;
h = 1;
approximate_value = (1+h)^(1/h);
true_value = exp(1);
```

```
error1 = abs(approximate_value - true_value)
```

```
error1 =
   0.718281828459045
```

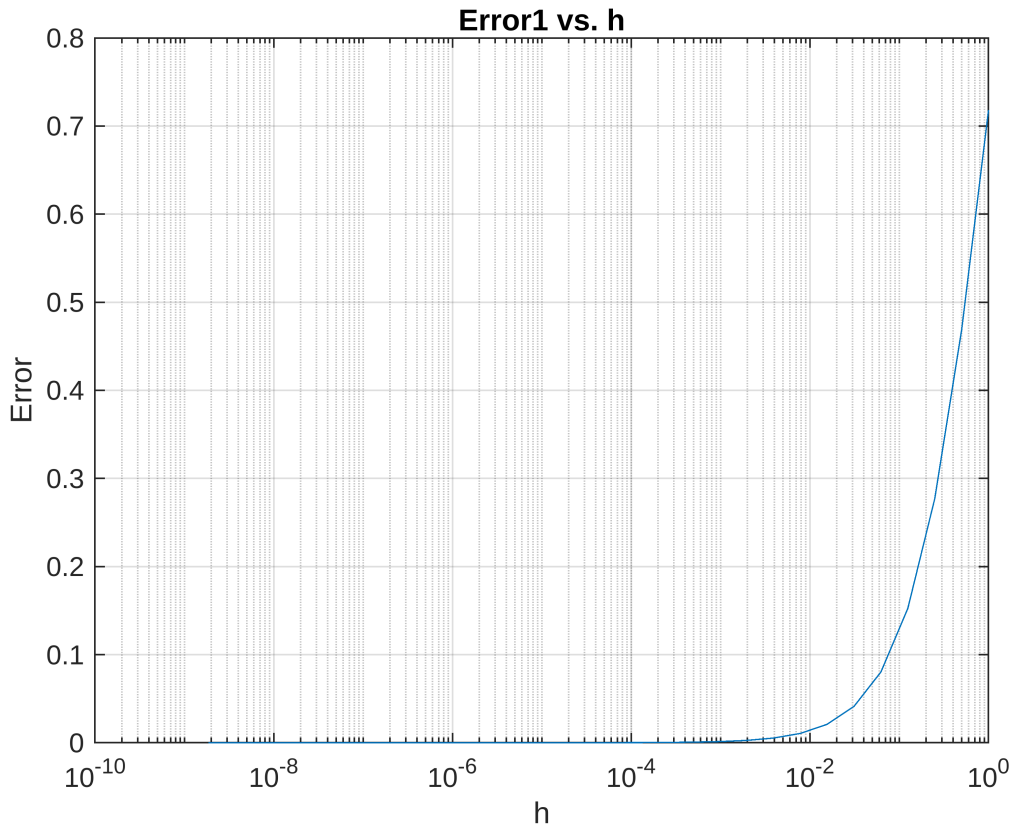
```
h_values = [];% store h values
error_values = [];% store error values.

while true
    approximate_value = (1+h)^(1/h);
    error1 = abs(approximate_value - true_value);

    h_values = [h_values, h];
    error_values = [error_values, error1];

    if error1 <= tolerance1
        break; % Exit loop when the error is below the tolerance
    else
        h = h/2; % h/2 for the next iteration
    end
end
fprintf('The value of h that meets the tolerance is: %.9f\n', h);
```

```
The value of h that meets the tolerance is: 0.000000002
```

```
fprintf('The value of error1 that meets the tolerance is: %.9f\n', error1);
```

```
The value of error1 that meets the tolerance is: 0.000000003
```

```
figure;
semilogx(h_values, error_values, '-');
title('Error1 vs. h');
xlabel('h');
ylabel('Error');
grid on;
```




```
tolerance1 = 1e-15;
h = 1;
approximate_value = (1+h)^(1/h);
true_value = exp(1);
error2 = abs(approximate_value - true_value)
```

```
error2 =
   0.718281828459045
```

```
h_values = [];% store h values
error_values = [];% store error values.

while true
    approximate_value = (1+h)^(1/h);
    error2 = abs(approximate_value - true_value);

    h_values = [h_values, h];
    error_values = [error_values, error2];

    if error2 <= tolerance1
        break; % Exit loop when the error is below the tolerance
    else
        h = h/2; % h/2 for the next iteration
    end
end
```

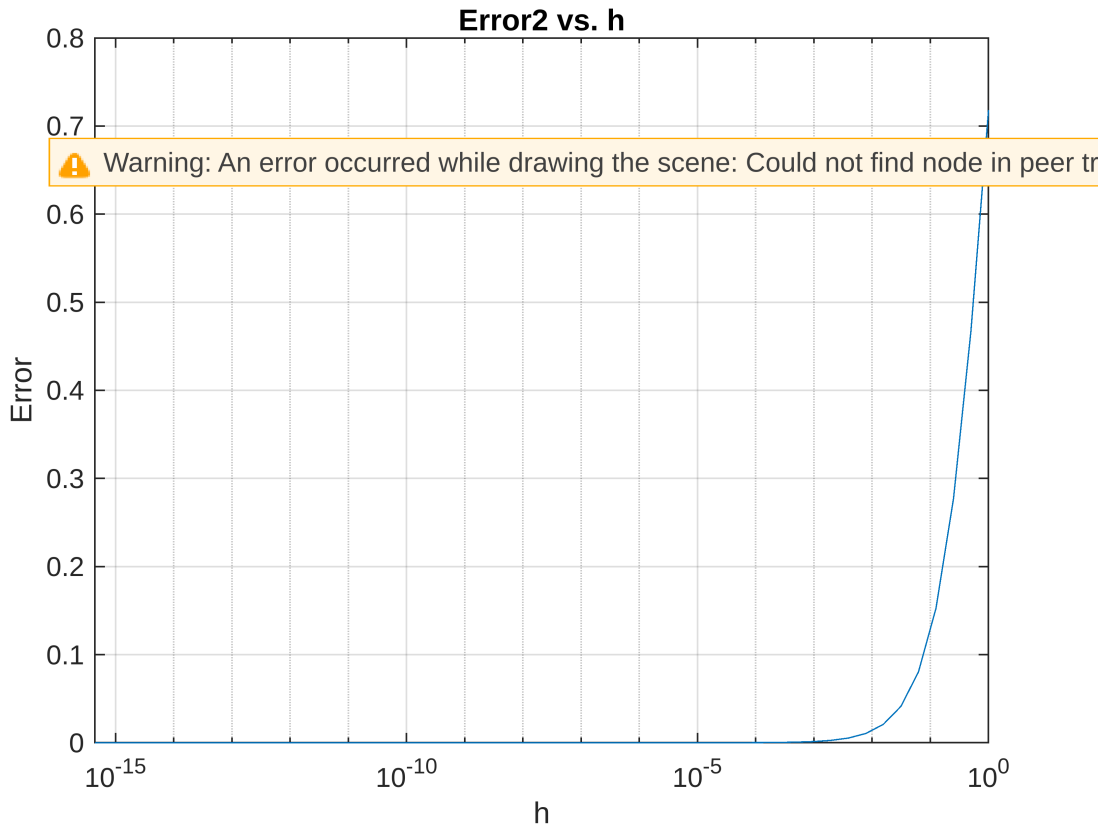
```
fprintf('The value of h that meets the tolerance is %.9f\n', h);
```

```
The value of h that meets the tolerance is 0.000000000
```

```
fprintf('The value of error2 that meets the tolerance is %.9f\n', error2);
```

```
The value of error2 that meets the tolerance is 0.000000000
```

```
figure;
semilogx(h_values, error_values, '-');
title('Error2 vs. h');
xlabel('h');
ylabel('Error');
grid on;
```



Problem 5

1. (Lambert W function, 15pts) Consider the following problem: Given a value of c ≥ −1/e, find the value of x such thatxex = c. (1) This is related to the Lambert W function, which has many applications (https://en.wikipedia.org/wiki/Lambert_W_function).Use Newton's method to solve equation (1) for values of c ≥ −1/e (Hint: solve the problem c − xex = 0). Produce three tables showing how Newton's method convergences to the correct answer for c = 0.5, c = 1, and c = 10 by comparing the iterations to the values produced by the function Matlab lambertw. Set the stopping criteria to ε = 10−14. You may have to experiment around with making initial guesses.

```
c1 = 0.5;
c2 = 1;
c3 = 10;
x0 = 1;
tol = 10e-14;

f1 = @(x) x.*exp(x)-c1;
fprime1 = @(x) (exp(x))*x + exp(x)
```

```
fprime1 = function_handle with value:
    @(x)(exp(x))*x+exp(x)
```

```
x1 = newtons_method(f1,fprime1,x0,tol)
```

```
x1 = 1×8
   1.000000000000000   0.591969860292861   0.393880900161210   0.353229140090994 ···
```

```
f2 = @(x) x.*exp(x)-c2;
fprime2 = @(x) (exp(x))*x + exp(x)
```

```
fprime2 = function_handle with value:
    @(x)(exp(x))*x+exp(x)
```

```
x2 = newtons_method(f2,fprime2,x0,tol)
```

```
x2 = 1×7
   1.000000000000000   0.683939720585721   0.577454477154450   0.567229737730117 ···
```

```
f3 = @(x) x.*exp(x)-c3;
fprime3 = @(x) (exp(x))*x + exp(x)
```

```
fprime3 = function_handle with value:
    @(x)(exp(x))*x+exp(x)
```

```
x3 = newtons_method(f3,fprime3,x0,tol)
```

```
x3 = 1×8
   1.000000000000000   2.339397205857212   1.927484544274192   1.766135502927263 ···
```

```
W1 = lambertw(c1)
```

```
W1 =
   0.351733711249196
```

```
W2 = lambertw(c2)
```

```
W2 =
   0.567143290409784
```

```
W3 = lambertw(c3)
```

```
W3 =
   1.745528002740699
```

```
error_1 = abs(x1 - W1)
```

```
error_1 = 1×8
   0.648266288750804   0.240236149043665   0.042147188912015   0.001495428841798 ···
```

```
error_2 = abs(x2 - W2)
```

```
error_2 = 1×7
   0.432856709590216   0.116796430175937   0.010311186744666   0.000086447320333 ···
```

```
error_3 = abs(x3 - W3)
```

```
error_3 = 1×8
   0.745528002740699   0.593869203116513   0.181956541533493   0.020607500186564 ···
```

```
c = {c1;c2;c3};
f = {f1;f2;f3};
fprime = {fprime1;fprime2;fprime3};
x_ = {x1;x2;x3};
W = {W1;W2;W3};

A = table(c,f,fprime,x_,W, ...
    'VariableNames', {'c','f','fprime','x','W'});

disp(A)
```

c	f	fprime	
{[0.500000000000000]}	{@(x)x.*exp(x)-c1}	{@(x)(exp(x))*x+exp(x)}	{[1 0.591969860292861 0.3938
{[1]}	{@(x)x.*exp(x)-c2}	{@(x)(exp(x))*x+exp(x)}	{[1 0.6839
{[10]}	{@(x)x.*exp(x)-c3}	{@(x)(exp(x))*x+exp(x)}	{[1 2.339397205857212 1.9274

Functions

```
function x = newtons_method(f,fprime,x0,tol)
max_iterations = 100;

x(1) = x0;
for n = 1:max_iterations
    x(n+1) = x(n) - f(x(n))/fprime(x(n));
    %stop criteria
    if abs(x(n+1)-x(n)) <= (1+abs(x(n+1)))*tol
break
    end
end
end
```