Lab Assignment #11

Due 4/15/2016 at 4pm on bCourses

This assignment will be partially graded by an autograder which will check the values of the required variables, so it is important that your variable names are exactly what they are supposed to be (variable names ARE case-sensitive). Instructions for submitting your assignment are included at the end of this document.

1 Numerical Differentiation

1.1 Helicopter Speed Checks

A helicopter pilot is stationary above a road and timing cars as they pass by visible landmarks. The pilot can determine the speed and acceleration of these cars by recording the time at which they pass the various landmarks. The speed and acceleration can then be computed using finite difference formulas.
Write a function with header

```
function [speed, acceleration] = speedFD(x, t, output_units)
```

where `x` and `t` are $1 \times N$ double arrays representing the x coordinates (in feet) of the landmarks and time (in seconds) when a car passes these landmarks, respectively. The outputs `speed` and `acceleration` should be double arrays with the same size as `x` and `t` giving the speed and acceleration of the car at each landmark. The input argument `output_units` is a `char` array with possible values `'mph'` or `'fps'`. If `output_units` is `'fps'` then the outputs should be given in the units ft/s and ft/s^2. If `output_units` is `'mph'` then the outputs should be given in units mi/hr and mi/hr^2. Note that the inputs will always be given in feet and seconds. Your function should compute the outputs using the central difference method. This will work for all but the edge points, where you should use a forward difference at the first point, and a backward difference at the last point. To compute acceleration, we could use the first derivative of the speed or the second derivative of position. For this assignment, use the first derivative of the speed.

- Forward Difference
$$\frac{df}{dx}|_{x_k} \approx \frac{f(x_{k+1}) - f(x_k)}{x_{k+1} - x_k}$$
- Central Difference
$$\frac{df}{dx}|_{x_k} \approx \frac{f(x_{k+1}) - f(x_{k-1})}{x_{k+1} - x_{k-1}}$$
- Backward difference
$$\frac{df}{dx}|_{x_k} \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

Test Cases:

```
>> t = [0 13 20 24.5 28 31 33.5 35.5 37];
>> x = [0 200 400 600 800 1000 1200 1400 1600];
>> [v1, a1] = speedFD(x, t, 'fps')

v1 =
     15.3846  20.0000  34.7826  50.0000  61.5385
     72.7273  88.8889  114.2857  133.3333

a1 =
     0.3550  0.9699  2.6087  3.3445  3.4965
     4.9728  9.2352  12.6984  12.6984

>> [v2, a2] = speedFD(x, t, 'mph')
v2 =
    10.4895  13.6364  23.7154  34.0909  41.9580
    49.5868  60.6061  77.9221  90.9091
a2 =
    1.0e+004*
    0.0871  0.2381  0.6403  0.8209  0.8582
    1.2206  2.2668  3.1169  3.1169

>> plot(t,v2,t,v1) % see Figure 1

>> t = [0 3 6.5 10.5 15 20 27];
>> x = [0 400 800 1200 1600 2000 2400]
>> [v3, a3] = speedFD(x, t, 'fps')
v3 =
    133.3333  123.0769  106.6667  94.1176 84.2105
    66.6667  57.1429

a3 =
    -3.4188  -4.1026  -3.8612  -2.6419  -2.8896
    -2.2556  -1.3605
```

Alternative second derivative formula (optional)

In class, we derived the expressions for the forward, backward and central differences using Taylor series. We can similarly use this method to derive formulas for the second derivative, again using a linear combination of Taylor series. For a non-uniform grid (as in the example above), the second derivative can be expressed as

$$\frac{d^2f}{dx^2}|_{x_k} = 2\left[\frac{f(x_{k-1})}{h_k(h_k+h_{k+1})} - \frac{f(x_k)}{h_k h_{k+1}} + \frac{f(x_{k+1})}{h_{k+1}(h_k+h_{k+1})}\right] + \mathcal{O}(h_k) \tag{1}$$

where $h_k = x_k - x_{k-1}$ and $h_{k+1} = x_{k+1} - x_k$. You can try out this formula for the second derivative and compare it to your answer above, where you applied a finite difference for the first derivative twice. Note that for a uniform grid where $h = x_{k+1} - x_k = x_k - x_{k-1}$ this reduces to the formula we found in lecture:

$$\frac{d^2f}{dx^2}|_{x_k} = \frac{f(x_{k+1}) - 2f(x_k) + f(x_{k-1})}{h^2} + \mathcal{O}(h^2) \tag{2}$$

for uniform grids.

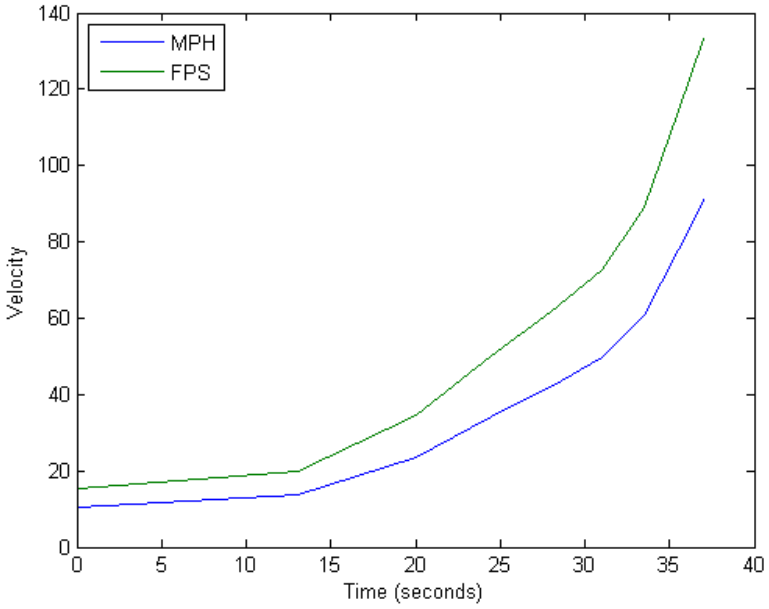

Figure 1: Velocity in mph and fps

1.2 2D Numerical Differentiation and Gradient

Functions often have more than one variable. For example, think of the area of a rectangle that is a function of its length and width. If we call x the length and y the width, we can write the area as a function f of x and y such as:

$$f(x, y) = xy$$

To determine how the area changes if we change the width, we need to take a derivative of f while holding other variables (the length) constant. This is called a **partial derivative** of f with respect to y, all other terms being held constant. The partial derivative with respect to x is denoted $\frac{\partial f}{\partial x}$ and in this example where $f = xy$:

$$\frac{\partial f}{\partial x}(x, y) = y$$

The gradient of $f(x, y)$ is denoted $\mathbf{grad}_f(x, y)$ and is defined as the vector of partial derivatives:

$$\mathbf{grad}_f(x, y) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient can be evaluated at a specific point $(x, y) = (a, b)$ by evaluating the partial derivatives at those points. For the area function above:

$$\mathbf{grad}_f(a, b) = [b, a]$$

Another example: if $g(x, y) = e^x \sin(y)$, then

$$\mathbf{grad}_g(x, y) = [e^x \sin(y), e^x \cos(y)]$$

In terms of numerical differentiation, you can use the central difference formula such that you have:

$$\mathbf{grad}_f(x_j, y_k) = \left[\frac{f(x_{j+1},y_k)-f(x_{j-1},y_k)}{x_{j+1}-x_{j-1}}, \frac{f(x_j,y_{k+1})-f(x_j,y_{k-1})}{y_{k+1}-y_{k-1}}\right]$$

Since the gradient of f is a vector which is spatially variable, we refer to it as a vector field that we can plot in two dimensions (recall Lab 6 when you plotted groundwater velocity vectors). Write a function with header

```
function [grad]=myGradient(f,bbox,N)
```

where `f` is a function handle, `bbox` is a bounding box (a 1×4 `double` of the form `[xmin, xmax, ymin, ymax]` and `N` is the total number of grid points in each of the intervals `[xmin, xmax]` and `[ymin, ymax]` (including the end points). The output `grad` should be a $(N-2) \times (N-2) \times 2$ `double` array representing the gradient of `f`. Note that `grad(:,:,1)` represents $\frac{\partial f}{\partial x}$, and `grad(:,:,2)` represents $\frac{\partial f}{\partial y}$. If you wish, you can define x and y using `meshgrid` in the same way as the code in the test case below.

When computing $\frac{df}{dx}$ with a central difference, you will create a $N \times (N-2)$ matrix, and when computing $\frac{\partial f}{\partial y}$, you will create a $(N-2) \times N$ matrix. Then when combining to create the output `grad`, you will take the $(N-2) \times (N-2)$ subset of each where both derivatives are defined. (Note that in many applications there are boundary conditions applied to the edge values but we are leaving the edge points alone here.)
You can now plot a contour plot of `f` over the bounding box (using Matlab `contourf`) and superimpose on this the vector field of the function gradient: at each point (x, y) there should be an arrow equal to the gradient of `f` at (x, y), using the Matlab `quiver` command (see the test case below).

```
%% Test case 1
>> f=@(x,y) x.^2-y.^2+1;
>> grad=myGradient(f,[-1 1 -1 1], 5);
>> grad(:,:,1)
ans =
    -1     0     1
    -1     0     1
    -1     0     1
>> grad(:,:,2)
ans =
    -1    -1    -1
     0     0     0
     1     1     1

%% Test case 2
>> g=@(x,y)exp(x).*sin(y);
>> grad=myGradient(g,[-1 1 -1 1], 6);
>> grad(:,:,1)
ans =
    0.3182    0.4747    0.7082    1.0565
    0.1120    0.1670    0.2492    0.3717
```

```
   -0.1120   -0.1670   -0.2492   -0.3717
   -0.3182   -0.4747   -0.7082   -1.0565
>> grad(:,:,2)
ans =
    0.4410    0.6579    0.9814    1.4641
    0.5236    0.7812    1.1654    1.7386
    0.5236    0.7812    1.1654    1.7386
    0.4410    0.6579    0.9814    1.4641
```

The following test case should produce the plot shown in Figure 2:

```
%% Test case 3
f=@(x,y) x.^2-y.^2;
bbox=[-2 2 -2 2];
N=21;
grad=myGradient(f,bbox, N);
x=linspace(bbox(1),bbox(2),N);
y=linspace(bbox(4),bbox(3),N); %note the ordering here
[xx,yy]=meshgrid(x,y);
zz=f(xx,yy);
figure
contourf(x,y,zz);
hold on
quiver(x(2:end-1),y(2:end-1),grad(:,:,1),grad(:,:,2));
axis('equal')
title('Gradient Plot of f(x)=x^2-y^2')
xlabel('x');ylabel('y');
legend('Contour of f','Gradient of f')
colorbar
hold off
```

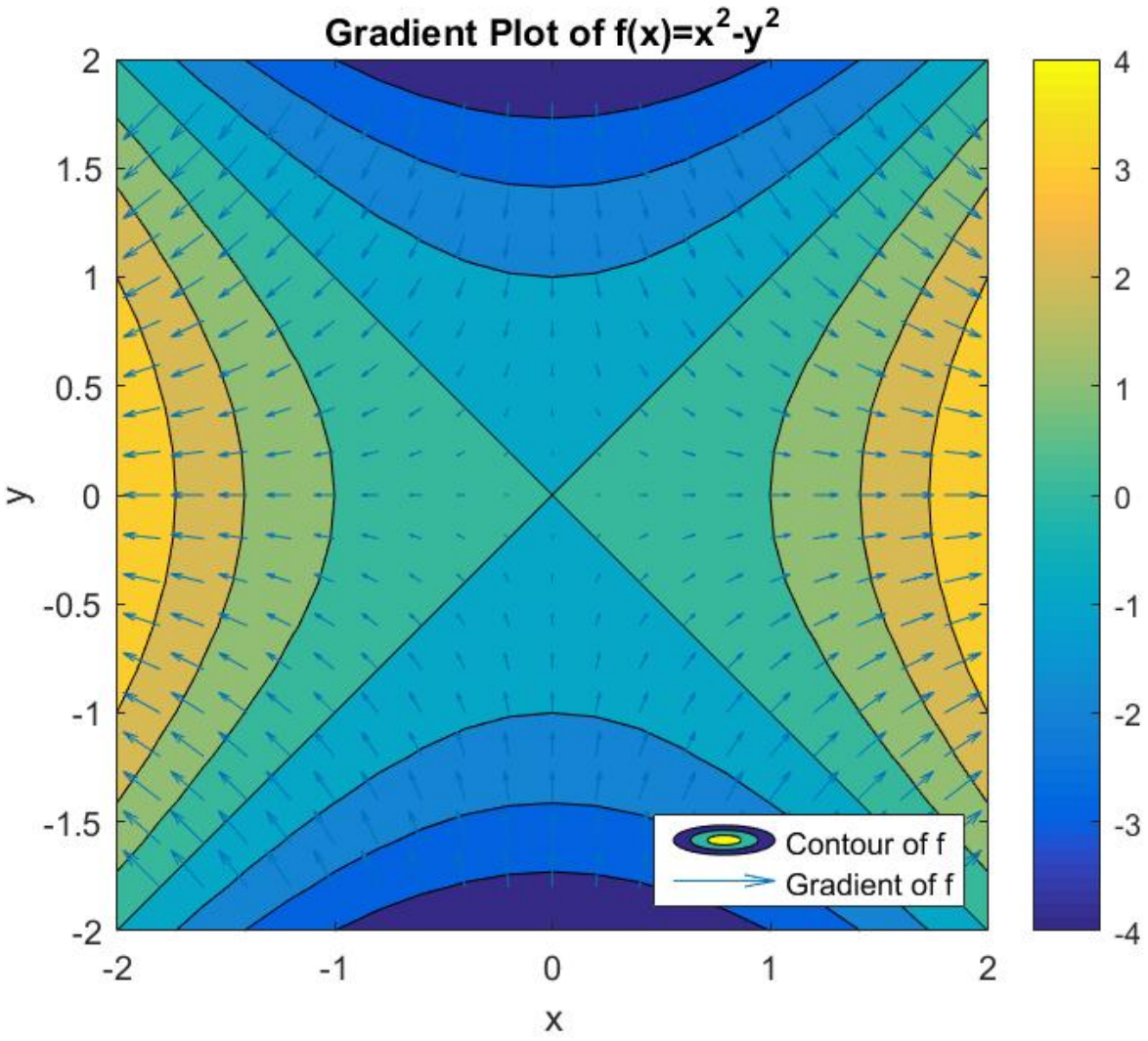

Figure 2: Result from the 3rd test case

2 Numerical Integration

2.1 Gauss Integral

In probability theory, the Gaussian distribution is of crucial importance and plays a role in a large number of problems. The density f of the zero-mean, one-standard deviation Gaussian distribution (shown in Figure 3) is given by:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

In many applications, this function must be integrated to compute the area between -A and A, given by $I = \int_{-A}^{A} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$ (see Figure 4).

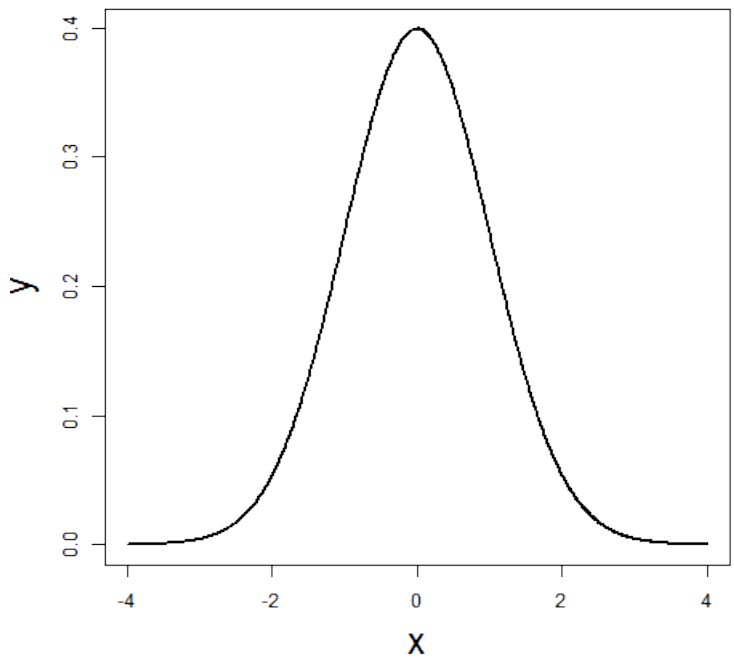

Figure 3: Gaussian Distribution

Write a function with header

```
function [I] = GaussIntegral(A,n)
```

which calculates the area shown in Figure 4 by numerically integrating $f(x)$ between $-A$ and A and dividing the interval $[-A, A]$ into n equal parts. In other words, interval endpoints are $x_1, x_2, ..., x_{n+1}$ and the k^{th} interval is between x_k and x_{k+1} where $x_1 = -A$ and $x_{n+1} = A$.

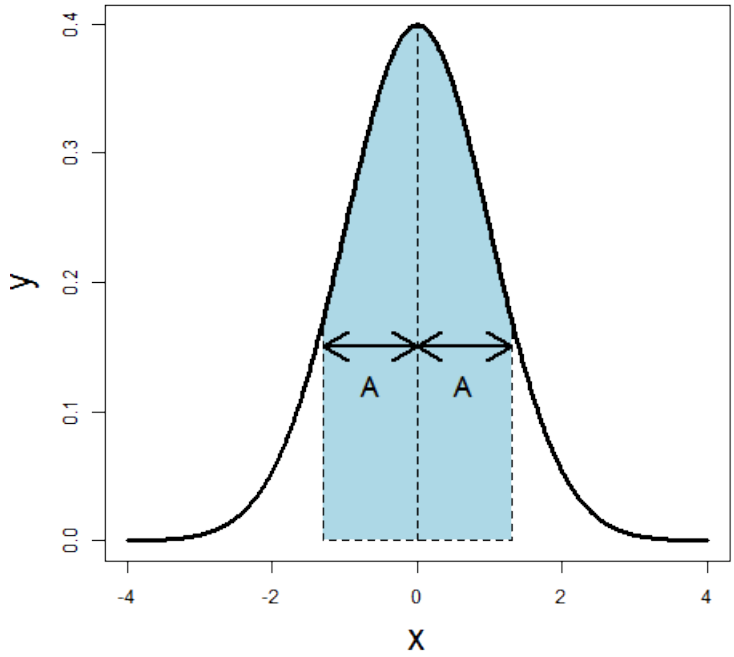

Figure 4: Area [-A,A] of Gaussian Distribution

`A`, `n`, and `I` are scalars of class `double`. To estimate the function value for each interval $[x_k, x_{k+1}]$, take the left endpoint value, i.e. $f(x_k)$, which is a form of a Riemann integral. Test Cases:

```
>> I1 = GaussIntegral(1,25)
I1 =
    0.6824

>> I2 = GaussIntegral(2,40)
I2 =
    0.9543

>> I3 = GaussIntegral(3,50)
I3 =
    0.9973
```

As you notice, as A get bigger and bigger, the integral is closer and closer to 1 (as long as n is sufficiently large). Indeed, it is a characteristic of a probability density function to have its integral between $-\infty$ and $+\infty$ equal to 1.

2.2 Corrugated Sheets

In constructing a roof, many types of materials may be used. One example is corrugated roofing, which is produced by pressing a flat sheet of aluminum into a sheet whose cross section resembles the shape of a sine wave (see Figure 5).

Figure 5: Corrugated Sheet

Consider a corrugated sheet that is L_C inches long, with a wave height of H inches from the center line, and a wavelength of P inches. To manufacture such a sheet, we need to determine the required length L_F of the initial flat sheet. To compute L_F, we determine the arc length of the wave, where the wave is given by $f(x) = H\sin(\frac{2\pi}{P}x)$. Thus, we can compute the arc length L_F using the equation

$$L_F = \int_0^{L_C} \sqrt{1 + (f'(x))^2}\, dx \tag{3}$$

Where $f'(x)$ is given by:

$$f'(x) = \frac{2\pi H}{P}\cos(\frac{2\pi}{P}x)$$

In this problem, you will write a function which computes this arc length (L_F) using:

a Trapezoidal Rule

b Simpson's Rule

c Riemann Integral (using left endpoints, as in the previous problem)

Write a function with header

```
function [L_Trap, L_Simp, L_Riem] = roofSheetLength(L_C, H, P,N)
```

where `N` is the number of equal intervals (like with `GaussIntegral`), and `L_C`, `H`, and `P` are the corrugated length, wave height, and wavelength, respectively (as given in the above equations). `L_Trap`, `L_Simp`, and `L_Riem` are the approximations of L_F given by the Trapezoidal

Rule, Simpson's Rule, and the Riemann Integral, respectively. All inputs and outputs are scalars of class `double`. Do not use the built-in MATLAB function `trapz` (though you can use it to check your answer). Hint: Make sure you check out the "Try it!" examples in Chapter 18 of the book before you do this problem!
Test Cases:

```
>> [L_T1, L_S1, L_R1] = roofSheetLength(72, 1.5, 2*pi, 50)
L_T1 =
    102.8949
L_S1 =
    102.5351
L_R1 =
    102.9242

>> [L_T2, L_S2, L_R2] = roofSheetLength(108, 2, 5, 20)
L_T2 =
    214.5902
L_S2 =
    214.7156
L_R2 =
    215.7755
```

3 Simpson's Rule and Lagrange Polynomials

Read §18.4 in your textbook to get some background on how Simpson's rule is formulated. As you've read, to perform Simpson's Rule, we need to calculate the Lagrange interpolation polynomial of three given points. (With 3 points, this just means you are fitting a quadratic polynomial to the 3 points - the Lagrange polynomial is a convenient way to find this unique quadratic polynomial that passes through these points.) Given a set of points (x_i, y_i), where $i = 1, 2, ..., n$, the Lagrange interpolation polynomial P of degree $n-1$ is such that $P(x_i) = y_i$. Note that there is only one unique Lagrange polynomial passing through a set of points. We also note that $P = a_0 + a_1X + ... + a_{n-1}X^{n-1}$. See §14.4 for background on Lagrange polynomials.

3.1 Lagrange polynomial

Write a function with header

```
function [P] = LagrangePolynomial(x,y)
```

where x and y are two vectors (class `double`) of length n containing the coordinates of the points (x_i, y_i). Your function should return the Lagrange interpolation polynomial in the form of a column vector (class `double`) of its coefficients, that is $P = [a_0, a_1, ..., a_{n-1}]^T$.
Test cases:

```
>> x=[1;2]; y=[2;0];
```

```
>> P = LagrangePolynomial(x,y)
P =
     4
    -2
```

```
>> x = [1;2;5;7;8];y=[2;0;3;-4;5];
>> P = LagrangePolynomial(x,y)
P =
   19.4444
  -29.5548
   14.6242
   -2.6738
    0.1599

% The following codes plots the points and function
% and checks that it passes through the points
>> plot(x,y,'o')
>> hold on
>> X=0:0.1:10;
>> Y=P'*[X.^0; X.^1; X.^2; X.^3; X.^4];
>> plot(X,Y)
```

3.2 Trinomial integration

On paper, show that

$$\int_a^b (a_0 + a_1 x + a_2 x^2)dx = a_0(b-a) + a_1\frac{b^2-a^2}{2} + a_2\frac{b^3-a^3}{3}$$

Write a function with header

```
function [I]= TrinomialIntegral(a,b,P)
```

which returns the **exact** value of $\int_a^b P(x)dx$ where P is a polynomial of degree 2 represented by the column vector of its coefficients: $P = [a_0, a_1, a_2]^T$. `I`, `a`, and `b` are scalars of class `double`. Here `P` is a 3×1 array of class `double`.
Test cases:

```
>> I1 = TrinomialIntegral(0,3,[1;2;1])
I1 =
    21

>> I2 = TrinomialIntegral(1,5,[3;0;3])
I2 =
    136
```

3.3 Simpson's Rule

Now that you can access the interpolation polynomial given a set of points and compute the exact integral of a polynomial of degree 2, you can implement Simpson's Rule where you only need to interpolate on sets of three points. Write a function with header

```
function [I,e]=SimpsonIntegral(f,a,b,n)
```

which calculates the integral `I` (scalar `double`) of the function handle `f` using Simpson's Rule and subdividing the interval $[a, b]$ in $2n$ equal parts. (You can use the functions you wrote above to do this.) Your function should also return the scalar `double e`, which represents the absolute value of the error between your estimate of the integral and Matlab's estimate which you can obtain using the function `integral`.
Test cases:

```
>> f = @(x) sin(x);
>> [I,e] = SimpsonIntegral(f,0,pi/2,10)
I =
    1.0000
e =
   2.1155e-07
```

```
>> f = @(x) x.^2 .* cos(x);
>> [I,e] = SimpsonIntegral(f,0,5,15)
I =
   -19.2188
e =
    1.1485e-04
```

Submission Instructions

Your submission should include the following following function files in a single zip file named Lab11.zip.

- `speedFD.m`
- `myGradient.m`
- `GaussIntegral.m`
- `roofSheetLength.m`
- `LagrangePolynomial.m`
- `TrinomialIntegral.m`
- `SimpsonIntegral.m`