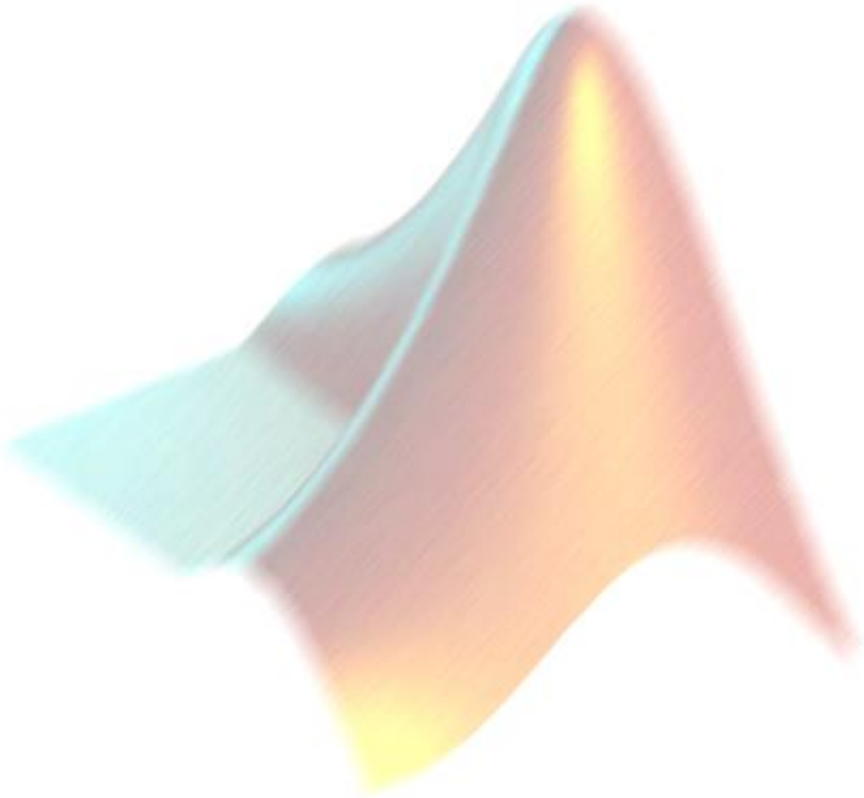
ENGGEN 131
Engineering Computation and
Software Development

Laboratory 4: Graphics and Image Processing



The Department of Engineering Science
The University of Auckland

Laboratory 4

TASK 4.0

Download the file "Lab_4.zip" from Canvas and extract the contents to an appropriate file directory. This contains files you may need for later tasks.

Then read the post "Lab 4 Notes" on the class Piazza discussion forum.

PLOTTING BASICS

Remember that all plots should include a title and labels (with units if appropriate). You can easily add a title and label your axes using the appropriate functions:

```
t = linspace(0,10,100);
plot(t,t.^2);
title('Graph of rocket path');
xlabel('time (seconds)');
ylabel('distance (metres)');
```

TIP ***Extra Figure Windows***

If you want to save the figure you have created and start work on a new figure just use the **`figure`** command. This creates a new figure window for drawing on. You can choose to draw on existing figure window *Figure n* by using the command:

```
>> figure(n)
```

For more information look up **`figure`** in the MATLAB *Help* i.e.,

```
>> doc figure
```

PLOTTING MULTIPLE DATA SETS

When plotting multiple data on the same set of axes you should use different colours/line styles and include a legend. You should refer back to your lecture notes to make sure you know how to modify line colours and styles.

SIGNAL PROCESSING – BACKGROUND FOR TASKS 4.1, 4.2, 4.3

Electrocardiography (ECG) is a technique used to record the electrical activity of the heart and determine the rate and regularity of a test subject's heartbeat. The data is measured using electrodes attached to the skin's surface, which detect electrical impulses generated by the heart and display them as a waveform.

Often, the signal provided by these electrodes suffers from electrical noise. Signal processing techniques can be used to reduce this noise, and provide data that suffers from less uncertainty. In Task 4.1, you are required to create a function to smooth the signal once, reducing the noise of the signal. Then in Task 4.2, you are required to written a function that will smooth the signal multiple times. Finally in Task 4.3 we plot the original unprocessed signal and smoothed versions.

THREEE POINT SMOOTHING

One way to smooth a signal is to use a three point smoothing algorithm. The basic idea is that you can create a smoother version of a signal by creating a new array, where each element (i.e. point) in the new array is found by taking a weighted average of the corresponding element and the two elements either side of it.

With a little thought it should be obvious that the first and last elements in the array can't be smoothed using this idea, as they only have an element on one side of them (so their values will simply remain the same).

All other elements can be smoothed by finding a weighted average of the corresponding element from the original array (along with the elements either side of it).

The weighted average of three elements (i.e. points) in a row e_1, e_2, e_3 is found using the following formula

$$\frac{e_1 + 2e_2 + e_3}{4}$$

e.g. if we have three values, 4, 1, 12 then the weighted average of these values will be

$$\frac{4 + 2 \times 1 + 12}{4} = \frac{18}{4} = 4.5$$

You have been provided with a function called `ThreePointSmooth` that will take as an input a 1D array containing three points in a row and calculate this weighted average for you.

Now consider smoothing a small array once using this idea:

$$[4 \quad 1 \quad 12 \quad 3 \quad 8]$$

The first and last values stay the same, so we can fill them in:

$$[4 \quad ? \quad ? \quad ? \quad 8]$$

Element 2 in the smoothed array will be the weighted average of elements 1, 2 and 3 from the original array, i.e. the weighted average of 4, 1 and 12:

$$\frac{\mathbf{4} + 2 \times \mathbf{1} + \mathbf{12}}{4} = \frac{18}{4} = 4.5$$

This gives us:

[4 4.5 ? ? 8]

Element 3 in the smoothed array will be the weighted average of elements 2, 3 and 4 from the original array, i.e. the weighted average of 1, 12 and 3:

$$\frac{\mathbf{1} + 2 \times \mathbf{12} + \mathbf{3}}{4} = \frac{28}{4} = 7$$

This gives us:

[4 4.5 7 ? 8]

Element 4 in the smoothed array will be the weighted average of elements 3, 4 and 5 from the original array, i.e. the weighted average of 12, 3 and 8:

$$\frac{\mathbf{12} + 2 \times \mathbf{3} + 8}{4} = \frac{26}{4} = 6.5$$

This gives us:

[4 4.5 7 6.5 8]

We have successfully completed ONE pass of three point smoothing the array.

Now we can repeat the process on our smoothed array [4 4.5 7 6.5 8] to get a slightly smoother signal. This would be a second pass. We can then smooth **that** signal, which would be a third pass. Repeated passes should produce smoother and smoother versions of the signal.

Try smoothing a signal for several passes by completing the table overleaf.

TIP ***Working an algorithm by hand helps your understanding***

Don't be tempted to skip filling in the table overleaf. Working through a problem by hand is often the best way to get to grips with what is going on and doing so will make it much easier for you to code up a solution.

Number of Passes	Signal				
0	4	1	12	3	8
1	4	4.5	7	6.5	8
2					
3					
4					
5					

TASK 4.1 SMOOTH SIGNAL

Write a function called `SmoothSignalPass` that applies the function `ThreePointSmooth` to smooth the signal ONCE (i.e. it performs one pass of the three point smooth algorithm). Your function will return the smoothed signal.

Input:
an array containing the unsmoothed signal.

Output:
an array containing the smoothed signal.

Your function should call the function `ThreePointSmooth` multiple times (think about how you are going to loop over all the elements in the unsmoothed signal).

Be careful with the end points of your input array (remember they get treated a little differently).

Make sure you read the documentation included in the header of the supplied function `ThreePointSmooth` so that you understand how this function works and the required input.

It is recommended that you create a test script to test the function `SmoothSignalPass`, possibly with some of the data from the table you have completed.

Here is an example call:

```
s = SmoothSignalPass([4 1 12 3 8]);
```

This should return the array `[4 4.5 7 6.5 8]`.

Submit the completed function `SmoothSignalPass` to MATLAB Grader.

TASK 4.2 SMOOTH SIGNAL N TIMES

Write a function called `SmoothSignal` that will smooth a singal n times (i.e. it does n passes) and returns the resulting smoothed signal.

Inputs:

an array containing the unsmoothed signal.
n, the number of times to smooth the signal

Output:

an array containing the smoothed signal.

Your function should call your `SmoothSignalPass` function n times..

Remember that from the second pass onwards you should be working on the smoothing the values found from the **previous** pass (not the very original unsmoothed signal).

It is recommended that you create a test script to test the function `SmoothSignal`, possibly with data from the table you have completed

Submit the completed function `SmoothSignal` to MATLAB Grader.

TASK 4.3 PLOTTING PROCESSING SIGNALS

A signal has been provided to you in `signal.mat`

For Task 4.3, write a **single** script file which produces **TWO** separate figures:

Figure 1: a single set of axes showing the unprocessed signal (in green) and a processed signal (in dashed red) obtained after 100 passes of smoothing the original signal.

Figure 2: 3 sets of axis showing the unprocessed signal, the signal after ONE pass of smoothing, and the signal after 100 passes of smoothing. Use solid black lines for each of these plots.

With the file `signal.mat` in your working directory, use the command:

```
load signal.mat
```

in your script to load two variables, `y` and `t`, into the workspace. `y` is an array of values giving the measurements taken by the electrodes (the signal), and `t` is an array of the corresponding times at which they were taken.

The figures that you should expect to obtain are shown on the next page.

Show the completed code and output of your script (the two figures) to a Teaching Assistant.

Figure 1 should look like this:

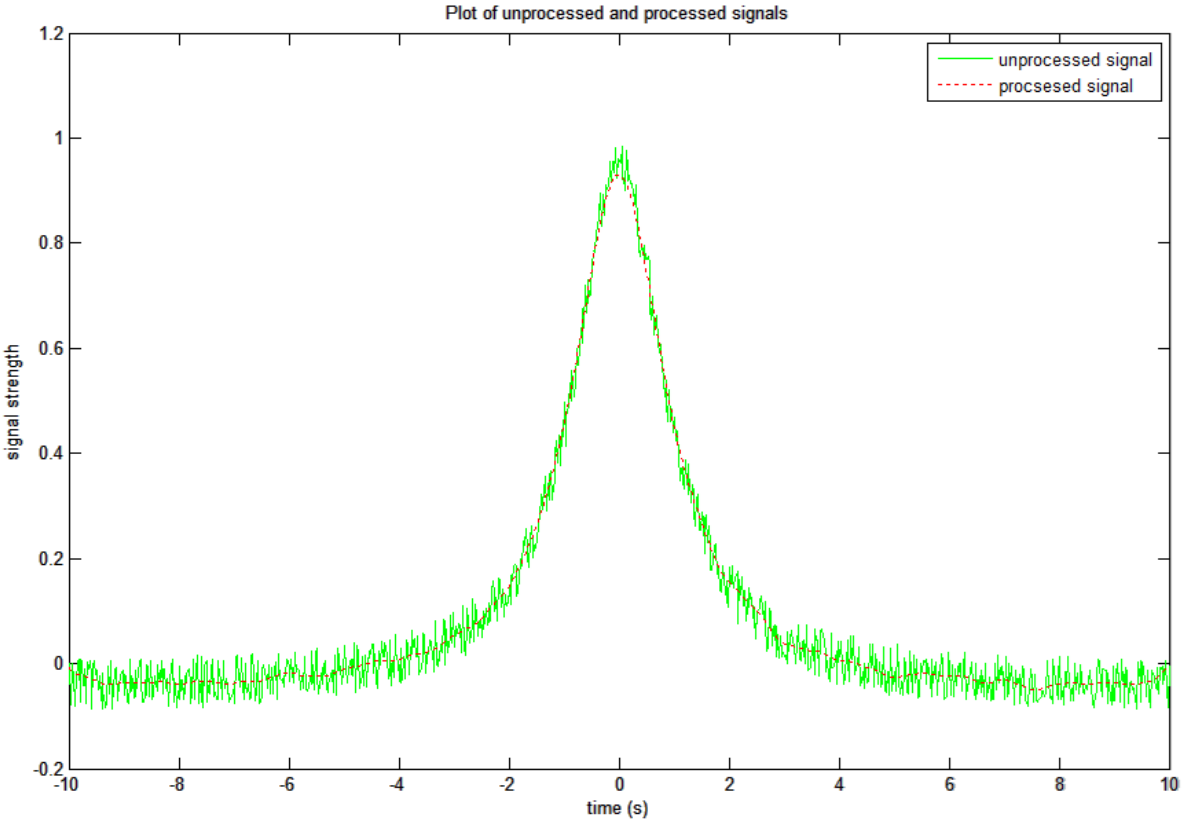
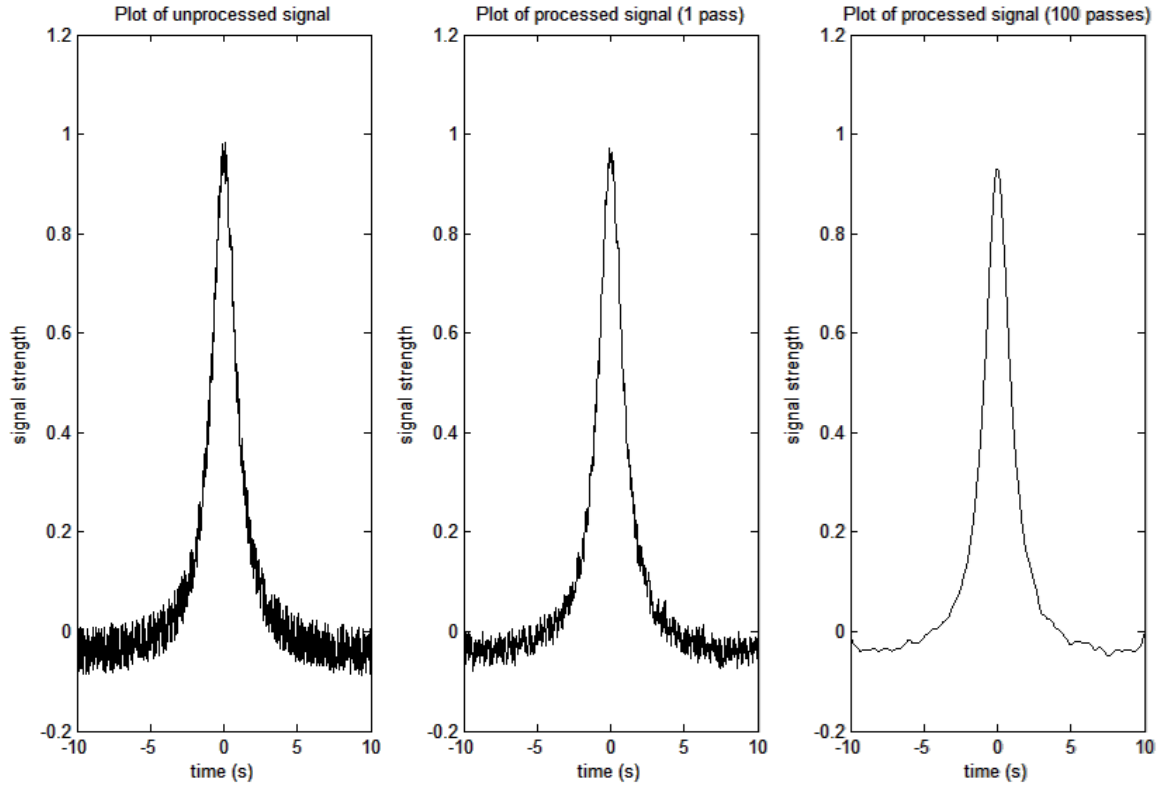


Figure 2 should look like this:



Stop ***Make sure you wrote a single script file and that it generates two figures***

Remember to follow instructions. If we request a single script file, we want a single file, not two.

IMAGE PROCESSING

MATLAB allows you to display many different types of image files using the `imread` and `image` functions. For example, you can read the supplied `xray.jpg` file (available for download from Canvas) using the command

```
>> xray_RGB = imread('xray.jpg', 'JPG');
```

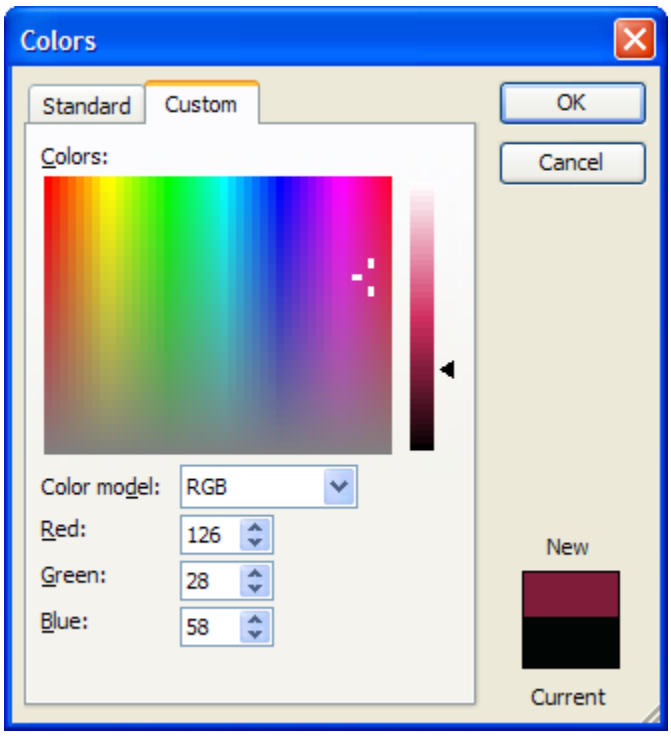
The variable `xray_RGB` is a 3-dimensional variable of *unsigned 8-bit integers* (integers that take values between 0 and 255). Unsigned 8-bit integers can not store negative numbers, fractions or values larger than 255. Attempting to assign values that cannot be represented as unsigned 8-bit integers to an array of unsigned 8-bit integers can produce unexpected results. Depending on what you want to do with the contents of the array, you may first need to create an array of standard numerical values (doubles), using the `double` function. If needed you could always convert back to unsigned 8-bit images using the `uint8` function.

The first two dimensions represent the location of pixels in the image and the third dimension identifies whether we are dealing with the amount or red, blue or green. MATLAB represents images using the RGB colouring for the pixels.

```
>> size(xray_RGB)

ans =
   1024    745      3
```

Thus, pixel (i, j) has red content `xray_RGB(i, j, 1)`, green content `xray_RGB(i, j, 2)` and blue content `xray_RGB(i, j, 3)`. As an example of RGB colours, consider the following *Colors Dialog Box* for choosing colours in Microsoft Word:



To display an image represented by an $m \times n \times 3$ matrix of unsigned 8-bit integers you simply input the matrix into the `image` function. To ensure each pixel is square set the axis to be equal.

```
>> image(xray_RGB)
>> axis equal
```

If you have read in the supplied image, Try typing the above commands to see what the image looks like.

Note that this image is used under the licensing terms CC BY 2.5 and can be found here: https://en.wikipedia.org/wiki/X-ray_generator#/media/File:Xray-verkehrshaus.jpg

IMPORTANT: remember that to draw an image, you need to make sure the array passed into the `image` function contains data stored as unsigned 8-bit integers. If need be you can create an array of unsigned 8-bit integers from a standard array of numerical values by using the `uint8` function.

TASK 4.4 Write a function to test if a pixel is blue

Airport security staff use x-ray imaging to scan luggage and detect dangerous objects such as weapons. Metal objects are inorganic and typically show up as blue on airport x-rays. This means that being able to detect blue pixels is important.

Your friend told you that a pixel with RGB value (r, g, b) is considered to be blue if

$$r < 128 \text{ and } g < 128 \text{ and } b \geq 128$$

Write a function called `IsPixelBlue` that recognizes if a pixel is blue by examining its red, green and blue values.

Input:
- red intensity
- green intensity
- blue intensity

Output:
- Boolean indicating whether the pixel is blue or not

It is recommended that you create a test script to test the function `IsPixelBlue`, your tests should include trying the following calls:

```
IsPixelBlue(10,20,200)          (should return 1, ie true)
IsPixelBlue(120,180,0)          (should return 0, ie false)
```

Submit the completed function `IsPixelBlue` to MATLAB Grader.

TASK 4.5 Create a filtered image

For Task 4.4, you need to create a filtered image to aid airport security staff, replacing all pixels **EXCEPT** the blue pixels with the colour white.

The RGB values for white are (255, 255, 255).

Write a function called `FilterImage` that filters the x-ray image to make the inorganic material more noticable.

Input:

a 3D image array (of `uint8` values)

Output:

a filtered 3D image array (of `uint8` values)

It should examine each pixel in turn and replace it with a white one if it is NOT blue. You should use the function `IsPixelBlue` that you wrote in Task 4.4 to determine if pixels are blue or not.

It is recommended that you create a test script to test the function `FilterImage`, be sure to read and display the image before filtering it. Also, be sure to display the final filtered image in a separate figure.

You may use the provided image files: `test.png`, `logo.png` and `xray.png` to test your function.

Does your filter work completely? If not why not?

Submit the completed function `FilterImage` to MATLAB Grader.

DRAWING SURFACES

The `meshgrid` function

Use the MATLAB *Help* to find out more about **`meshgrid`**. What is the output of the following commands?

```
>> x = [0.1 0.2 0.3];
>> y = [1.5 2.0 2.5 3.0];
>>[X, Y] = meshgrid(x, y)
```

Why is this useful for drawing surfaces?

Stop ***Meshgrid***

If you're not sure how **`meshgrid`** works or why it is important make sure you ask a teaching assistant to clarify things.

TASK 4.6 Drawing Surfaces

The script `PlotSurfaces.m` uses two different methods to draw surfaces over the domain: $-1 \leq x \leq 1$ and $-2 \leq y \leq 2$.

Method 1: Using a nested `for` loop and the dot operator to create the surface :

$$f(x,y) = \frac{1}{e^{(5x)^2(5y)^2}}$$

Method 2: Using `meshgrid()` and the dot operator to create the surface:

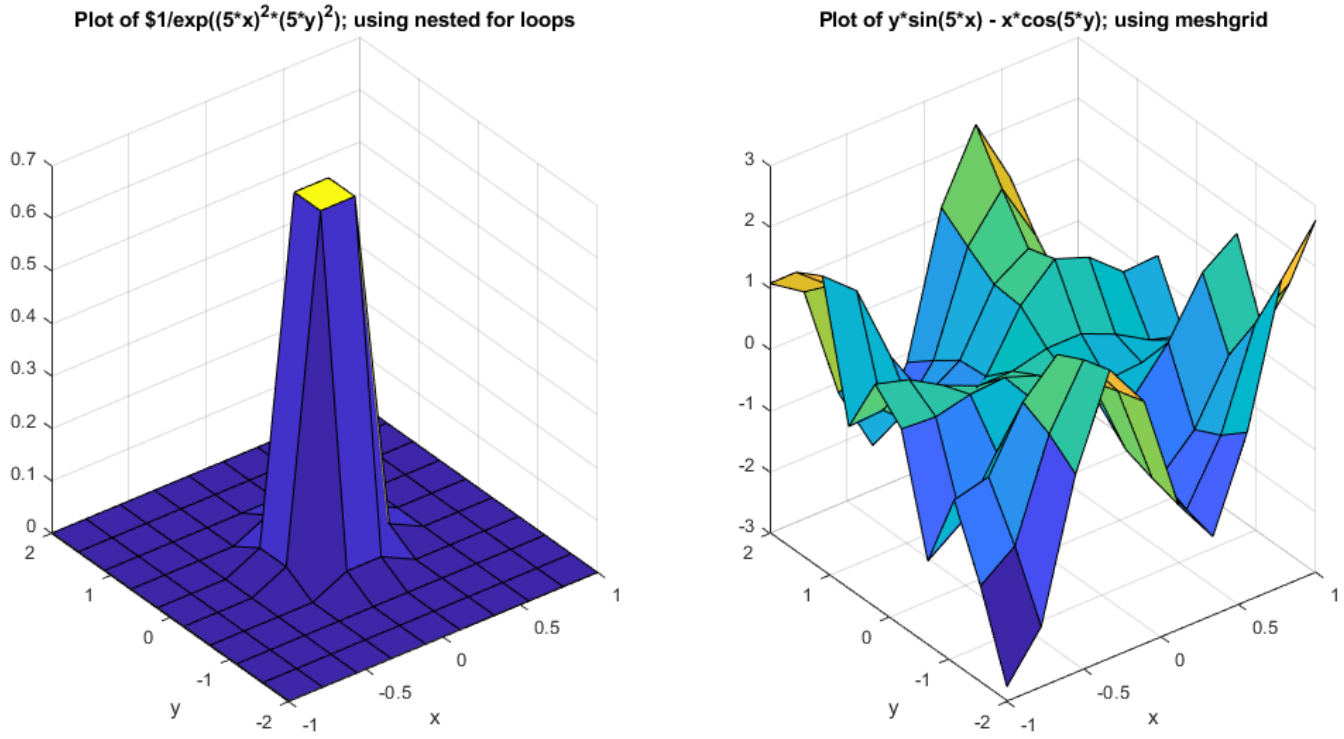
$$f(x,y) = y\sin(5x) - x\cos(5y)$$

When the script `PlotSurfaces.m` is run, it should display two plots, one subplot for each method, side by side in two columns on a single figure.

However, the script `PlotSurfaces.m` has bugs. Read through the script to better understand the scripts intentions and then fix any bugs in the script.

Remember, you can use the debugger to step through line by line, so that you can see how each line of code changes the variables in the workspace.

The expected figure from the script `PlotSurfaces.m` with 10 grid points equally spaced along each axis should look like as follows.



Make sure that both your surfaces look the same as the figure above. Try increasing the number of points to 100 in each direction and see what happens to your surfaces.

Show the output of the script `PlotSurfaces.m` with 100 grid points equally spaced in each direction to a Teaching Assistant.

LAB 4 Lab Task Submission Summary

□ Task 4.1
Submit the completed function `SmoothSignalPass` to MATLAB Grader.

□ Task 4.2
Submit the completed function `SmoothSignal` to MATLAB Grader.

□ Task 4.3
Show the completed code and output of the script to a Teaching Assistant.

□ Task 4.4
Submit the completed function `IsPixelBlue` to MATLAB Grader.

□ Task 4.5
Submit the completed function `FilterImage` to MATLAB Grader.

□ Task 4.6
Show the output of the debugged script `PlotSurfaces.m` to a Teaching Assistant.

COMPULSORY LAB TASKS END HERE

EXAM PRACTICE

To convert a colour RGB image into a grayscale image, the values of red, green and blue for each pixel must be replaced by a value representing the luminance for that pixel.

The formula for calculating the luminance value for a given r,g,b pixel is

$$lum = 0.3r + 0.59g + 0.11b$$

Setting the r, g and b values all to be *lum* will result in a shade of grey.

Write a function called `luminance`() that will calculate the luminance value for a given colour. Your function should take as inputs three values (representing the amount of red, green and blue) and return the luminance value.

Write a second function called `ConvertToGrayscale()` which will convert a colour RGB image into a grayscale image by using the function `luminance()`. The function `ConvertToGrayscale()` should take as input a 3D array describing a colour image and then return a 3D array describing the equivalent grayscale image.

IMPORTANT: Remember to comment your function files.

EXAM PRACTICE

Write a script file called halveImage that scales the size of an image down by a factor of 2 and then draws it to the screen.

Your script should work by replacing each 2x2 square of pixels in the old image by a single pixel in the new image. The RGB values for the new pixel will be the average of the RGB values for the 4 pixels from the 2x2 square.

You may assume that your image is an even number of pixels wide and high.

IMPORTANT: Remember to comment your file.

HINT: You are likely to need to use the `double` function and the `uint8` function.