SCC.366 Media Coding and Processing
Course Work 1

This CW comprises **THREE** distinct tasks:

- **Task 1:** Piece-wise Linear Transformation for Image Enhancement
- **Task 2:** Histogram Equalization for Image Enhancement
- **Task 3:** Gaussian Filtering for blurring and noise reduction in images

The submission deadline for submission is **Friday, 17 November 2023, 4:00 PM GMT**

Submission Requirements: For each of the three tasks, you are expected to submit a single script containing the function implementation, along with a few lines of code that demonstrate how to call the function, using default input parameters. This submission should consist of three MATLAB scripts, named "Task1.m," "Task2.m," and "Task3.m," all placed within a single folder. Additionally, you may include sample images that serve as default parameters for your functions.

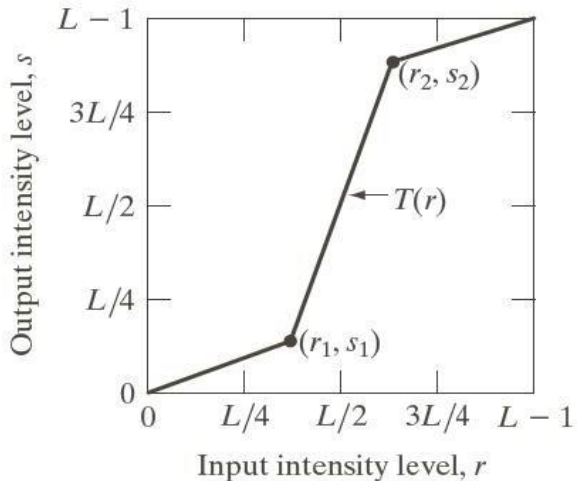
Task 1: Piece-wise Linear Transformation for Image Enhancement

Introduction:
Piece-wise Linear Transformation is a grayscale transformation technique used for image enhancement in the spatial domain. It allows the manipulation of an image to make it more suitable for specific applications. This method is particularly useful for enhancing contrast.

Description:
The illustration below depicts a typical Piece-wise Linear Transformation used for contrast stretching. The locations of points (r_1, s_1) and (r_2, s_2) govern the shape of the transformation function. Various scenarios can arise based on these points. For instance:

- If $r_1 = s_1$ and $r_2 = s_2$, the transformation is linear, resulting in no change in intensity.
- If $r_1 = r_2$, $s_1 = 0$ and $s_2 = L - 1$, the transformation becomes a thresholding function that creates a binary image.
- Intermediate values of (r_1, s_1) and (r_2, s_2) yield different degrees of intensity level spread in the output image, impacting its contrast.

In general, $r_1 \leq r_2$ and $s_1 \leq s_2$ is assumed so that the function is single valued and monotonically increasing. This preserves the order of intensity levels, thus preventing the creation of intensity artifacts.



Task Requirements:

You are tasked with creating a MATLAB function for a more general Piece-wise Linear Transformation, allowing the user to specify the number of points used for creating the transformation. The function should accept the following inputs:

- An integer, denoted as *numPoints*, representing the number of points for creating the transformation.
- A gray-scale image denoted as *inputImage*
- An array of points to control the shape of the transformation function. This array should have a size of *numPoints*× 2 and be denoted as *pointsArray*.

MATLAB Function:

```
function transformedImage = piecewiseLinearTransform(numPoints, inputImage, pointsArray)
    % Your code here
end
```

Function Outputs:
Your MATLAB function should return the transformed image as output. Additionally, it should display both the original and transformed images in a single window as shown below:

Input Validation:
The function should validate the inputs for correct range and format. If the inputs are invalid, the function should provide informative error messages to users.
Note:

You must implement your own code using basic MATLAB built-in functions, such as *imshow*, *subplot*, *zeros*, *im2double.* Avoid using MATLAB built-in functions designed for generating line parameters in your Piece-wise Linear Transformation function. Additionally, minimize the use of loops to improve code efficiency.

Marks allocation for Task 1:
Code efficiency (e.g., avoid loops if possible, fewer lines)
Code commenting (descriptive comments for each line of the code)
Code correctness and results

Task 2: Histogram Equalization for Image Enhancement

Introduction:
Histogram equalization is a fundamental image processing technique used to enhance the contrast of an image. It works by redistributing the intensity levels in an image, which can help in revealing more details and making the image more visually appealing. In this assignment, you will be required to implement histogram equalization from scratch using MATLAB.

Description:
Histogram equalization involves modifying the pixel intensities of an image so that the histogram of the resulting image is as uniform as possible. It enhances the overall contrast by stretching the intensity range of the image. Your task is to create a MATLAB function to perform histogram equalization without using built-in functions for this purpose.

Task Requirements:
You are tasked with creating a MATLAB function, *histogramEqualization*, for histogram equalization that takes a grayscale image (*inputImage*) as input.
Avoid using built-in MATLAB functions designed for histogram equalization.
Ensure the function returns the histogram-equalized image.

Additionally, it should display both the original and transformed images in a single window.

The function should also generate two subplots. In the first subplot, plot the histogram of the original image, and in the second subplot, plot the histogram of the histogram-equalized image.

MATLAB Function:

```
function histogramEqualizedImage = histogramEqualization (inputImage)
   % Your code here
end
```

Input Validation:
The function should validate the input for correct format. If the input is an RGB image, the function should transfer the image to YCbCr color space and performs histogram equalisation.

Note:
You are required to implement your own code using basic MATLAB built-in functions such as *imshow*, *imread*, *subplot*, *zeros*, *im2double*. Avoid using MATLAB built-in functions explicitly designed for generating or manipulating histograms. Furthermore, minimize the use of loops to improve code efficiency.

Marks allocation for Task 2:
Code efficiency (e.g., avoid loops if possible, fewer lines)
Code commenting (descriptive comments for each line of the code)
Code correctness and results

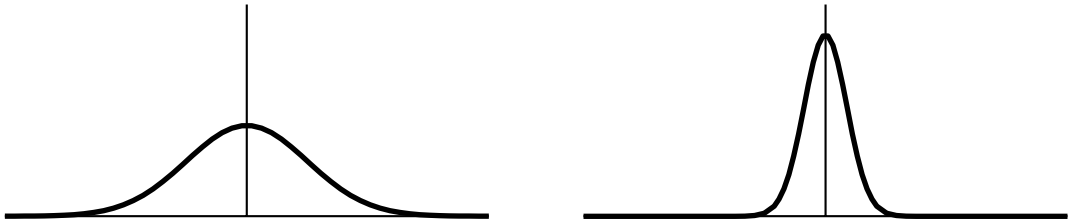
Task 3: Gaussian Filtering for blurring and noise reduction in images

Introduction:
Gaussian filtering is a widely used image processing technique that is crucial for both image blurring and noise reduction. It operates by applying a Gaussian kernel to an image, which effectively blurs the image or reduces noise while preserving important image features. In this task, you will embark on the implementation of Gaussian filtering from scratch using MATLAB. This will allow you to grasp the inner workings of this valuable tool for image enhancement..

Description:
Gaussian filters are a class of low-pass filters, all based on the Gaussian probability distribution function:

$$f(x) = e^{-\frac{x^2}{2\sigma^2}}$$

where σ is the standard deviation: a large value of σ produces a flatter curve, and a small value leads to a "pointier" curve. The figure below shows examples of such one-dimensional Gaussian.



Large value of σ Small value of σ

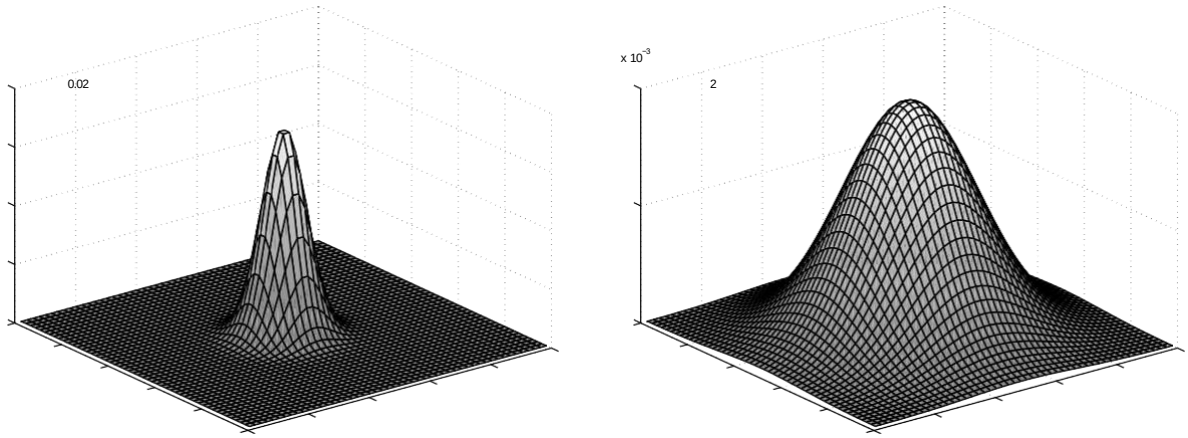
A two-dimensional Gaussian function is given by

$$f(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The command *fspecial('gaussian')* produces a discrete version of this function. We can draw pictures of this with the surf function, and to ensure a nice smooth result, we shall create a large filter (size 50×50) with different standard deviations.

```
>> a=50;s=3;
>> g=fspecial('gaussian',[a a],s);
>>  surf(1:a,1:a,g)
>> s=9;
>> g2=fspecial('gaussian',[a a],s);
>>    figure,surf(1:a,1:a,g2)
```

The surfaces are shown in the figure below:



Gaussian filters have a blurring effect which looks very similar to that produced by neighbour- hood averaging. Let's experiment with some different gaussian filters.

```
>>    g1=fspecial('gaussian',[5,5]);
>>    g1=fspecial('gaussian',[5,5],2);
>>    g1=fspecial('gaussian',[11,11],1);
>>    g1=fspecial('gaussian',[11,11],5);
```

The final parameter is the standard deviation, which if not given defaults to 0.5. The second parameter (which is also optional), gives the size of the filter; the default is 3×3. If the filter is to be square, as in all the above examples, we can just give a single number in each case.

If we let the standard deviation grows large without bound, we obtain the averaging filters as limiting values. For example, the command *fspecial('gaussian',3,100)* produces the 3×3 averaging filter.

Task Requirements:

You are tasked with creating a MATLAB function, *gaussianFiltering*, that takes the size of filter *N* and standard deviation *sigma* of a Gaussian distribution as input and outputs a normalised $N \times N$ Gaussian filter. A few notes:

- Ensure the function display the generated filter with the *surf* function.
- Avoid using built-in MATLAB functions designed for filtering such as *fspecial* and *imfilter* etc.
- (To validate your implementation) If you pass *N=3* and *sigma=1* to your function, you should get the same filter as the one generated by the following command: *fspecial('gaussian',3,1)*

MATLAB Function:

```
function gaussianFilter = gaussianFiltering (N, sigma)
   % Your code here
end
```

Input Validation:
The function should validate the inputs for correct format/range.

Note:
You are required to implement your own code using basic MATLAB built-in functions such as *meshgrid*, *imshow*, *imread*, *subplot*, *exp*, *zeros*, *im2double*. Minimize the use of loops to improve code efficiency.

Marks allocation for Task 3:
Code efficiency (e.g., avoid loops if possible, fewer lines)
Code commenting (descriptive comments for each line of the code)
Code correctness and results