

## ✓ Coursework 1: Image filtering

In this coursework you will practice techniques for image filtering. The coursework includes coding questions and written questions. Please read both the text and the code in this notebook to get an idea what you are expected to implement.

### What to do?

- Complete and run the code using `jupyter-lab` or `jupyter-notebook` to get the results.
- Export (File | Save and Export Notebook As...) the notebook as a PDF file, which contains your code, results and answers, and upload the PDF file onto [Scientia](#).
- Instead of clicking the Export button, you can also run the following command instead: `jupyter nbconvert coursework_01_solution.ipynb --to pdf`
- If Jupyter complains about some problems in exporting, it is likely that pandoc (<https://pandoc.org/installing.html>) or latex is not installed, or their paths have not been included. You can install the relevant libraries and retry. Alternatively, use the Print function of your browser to export the PDF file.
- If Jupyter-lab does not work for you at the end (we hope not), you can use Google Colab to write the code and export the PDF file.

### Dependencies:

You need to install Jupyter-Lab ([https://jupyterlab.readthedocs.io/en/stable/getting\\_started/installation.html](https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html)) and other libraries used in this coursework, such as by running the command: `pip3 install [package_name]`

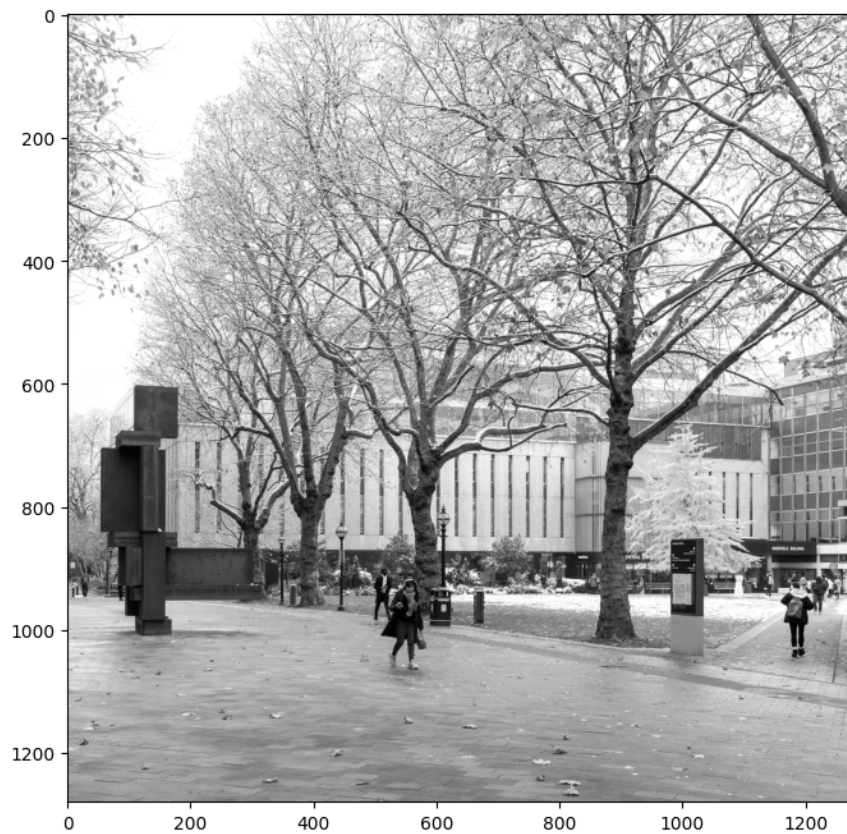
```
# Import libraries (provided)
import imageio.v3 as imageio
import numpy as np
import matplotlib.pyplot as plt
import noise
import scipy
import scipy.signal
import math
import time
```

### ✓ 1. Moving average filter (20 points).

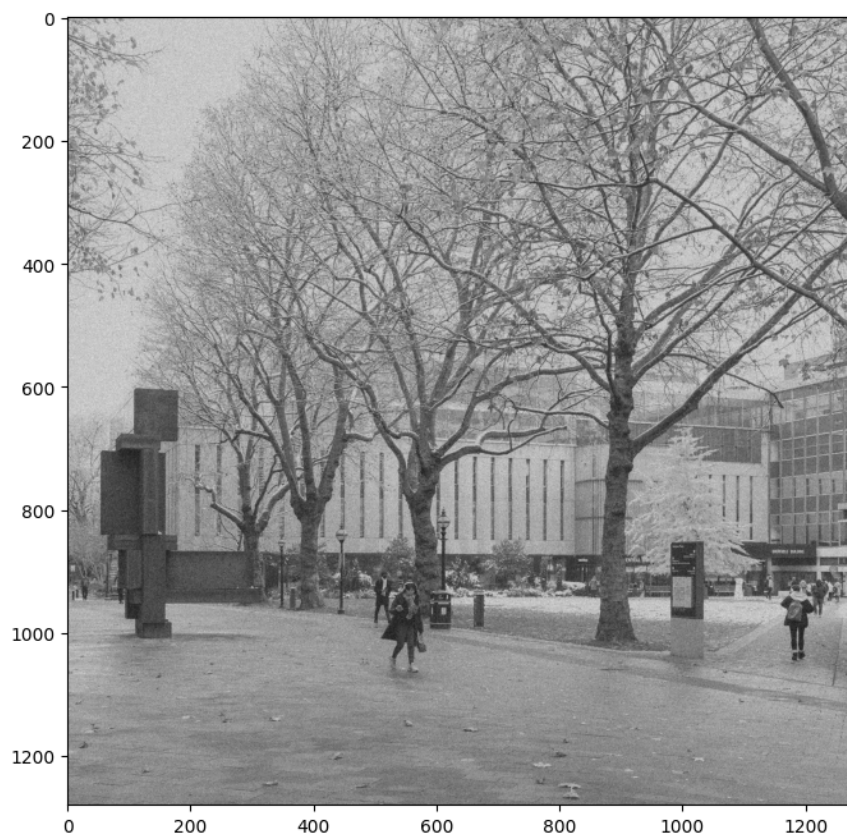
Read the provided input image, add noise to the image and design a moving average filter for denoising.

You are expected to design the kernel of the filter and then perform 2D image filtering using the function `scipy.signal.convolve2d()`.

```
# Read the image (provided)
image = imageio.imread('campus_snow.jpg')
plt.imshow(image, cmap='gray')
plt.gcf().set_size_inches(8, 8)
```



```
# Corrupt the image with Gaussian noise (provided)
image_noisy = noise.add_noise(image, 'gaussian')
plt.imshow(image_noisy, cmap='gray')
plt.gcf().set_size_inches(8, 8)
```



✓ Note: from now on, please use the noisy image as the input for the filters.

## 1.1 Filter the noisy image with a 3x3 moving average filter. Show the filtering results.

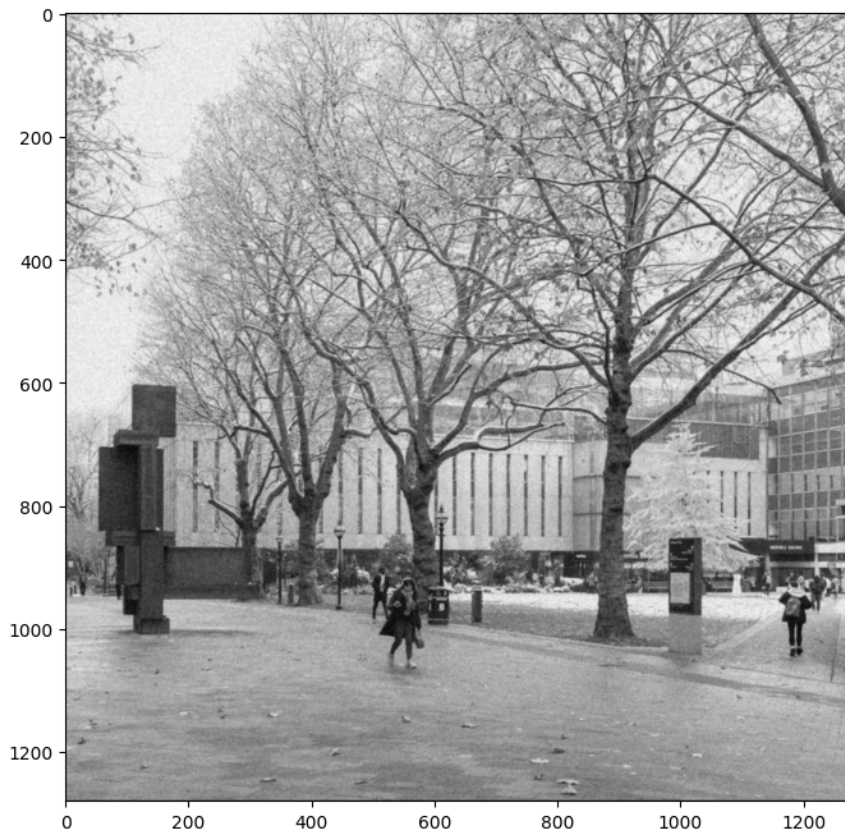
```
# Design the filter h
### Insert your code ###
h = np.ones((3, 3), dtype=float) / 9

# Convolve the corrupted image with h using scipy.signal.convolve2d function
### Insert your code ###
image_filtered = scipy.signal.convolve2d(image_noisy, h, mode='same', boundary='fill', fillvalue=0)

# Print the filter (provided)
print('Filter h:')
print(h)

# Display the filtering result (provided)
plt.imshow(image_filtered, cmap='gray')
plt.gcf().set_size_inches(8, 8)
```

```
Filter h:
[[0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]]
```



## 1.2 Filter the noisy image with a 11x11 moving average filter.

```
# Design the filter h
### Insert your code ###
h = np.ones((11, 11), dtype=float) / 121

# Convolve the corrupted image with h using scipy.signal.convolve2d function
### Insert your code ###
image_filtered = scipy.signal.convolve2d(image_noisy, h, mode='same', boundary='fill', fillvalue=0)

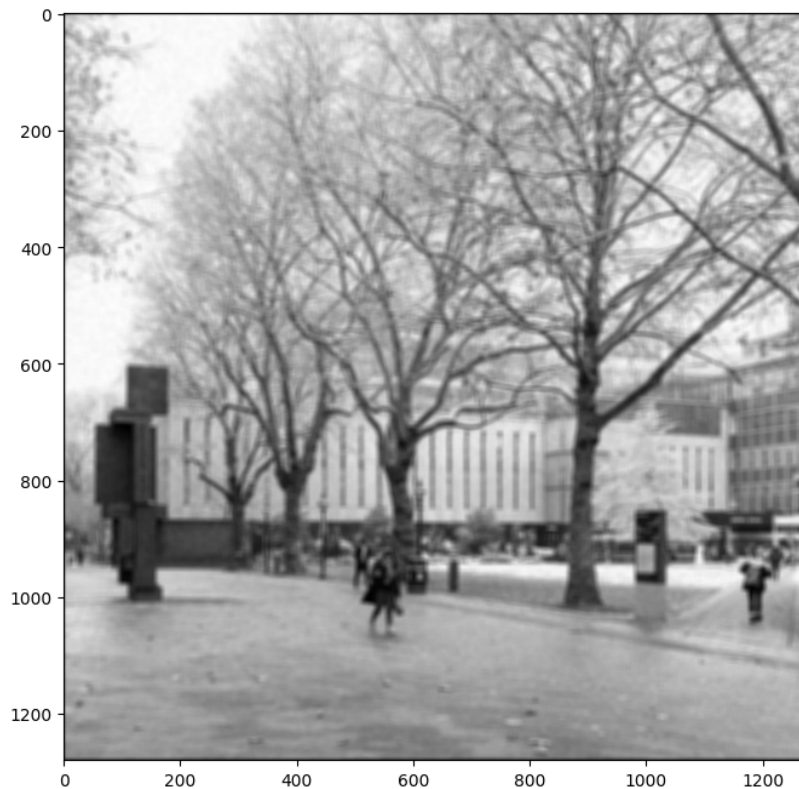
# Print the filter (provided)
print('Filter h:')
print(h)

# Display the filtering result (provided)
plt.imshow(image_filtered, cmap='gray')
plt.gcf().set_size_inches(8, 8)
```

```

Filter h:
[[0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]]

```



1.3 Comment on the filtering results. How do different kernel sizes influence the filtering results?

✓ Insert your answer

With the 3x3 moving average filter, the noisy image is blurred out slightly but the image still retains sharpness to distinguish small objects. This is because the kernel moves between a relatively small group of pixels (9 pixels as compared to the image width of 1280). In the 11x11 moving average filter, there is a much greater smoothing affect as we calculate the average between a larger cluster of pixels, and this leads to more noise removal, but also loss in detail which makes it look blurrier.

✓ 2. Edge detection (56 points).

Perform edge detection using Sobel filtering, as well as Gaussian + Sobel filtering.

✓ 2.1 Implement 3x3 Sobel filters and convolve with the noisy image.

```

# Design the filters
### Insert your code ###
sobel_x = [[1, 0, -1],
            [2, 0, -2],
            [1, 0, -1]]
sobel_y = [[1, 2, 1],
            [0, 0, 0],
            [-1, -2, -1]]

# Image filtering
image_sobel_x = scipy.signal.convolve2d(image_noisy, sobel_x, mode='same', boundary='fill', fillvalue=0)
image_sobel_y = scipy.signal.convolve2d(image_noisy, sobel_y, mode='same', boundary='fill', fillvalue=0)

# Calculate the gradient magnitude
### Insert your code ###
grad_mag = np.sqrt(image_sobel_x**2 + image_sobel_y**2)

# Print the filters (provided)
print('sobel_x:')
print(sobel_x)
print('sobel_y:')
print(sobel_y)

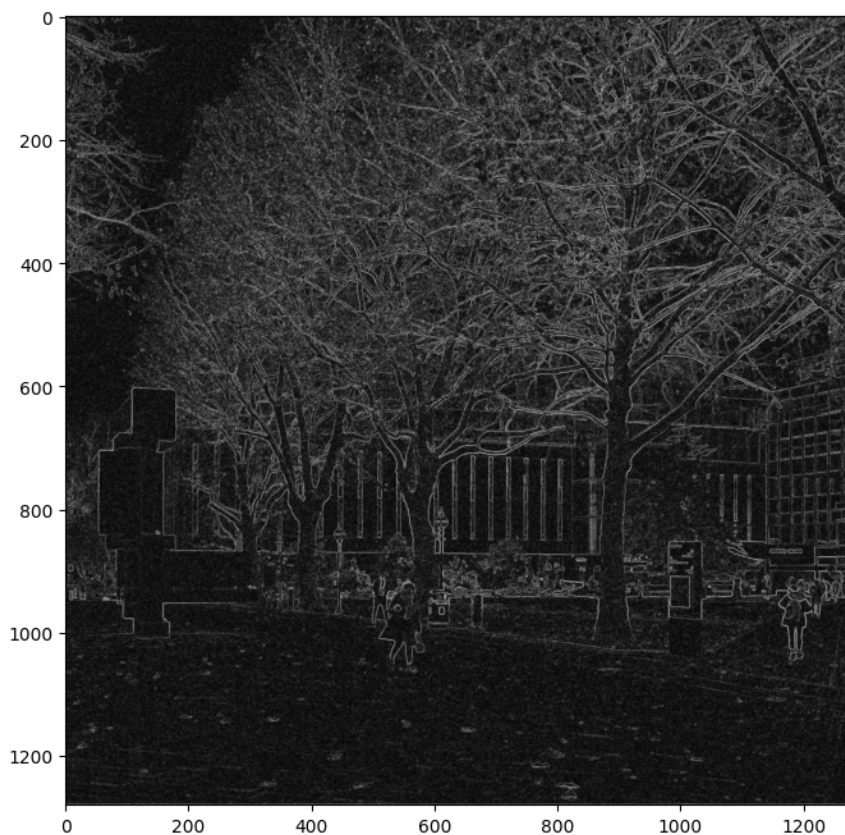
# Display the magnitude map (provided)
plt.imshow(grad_mag, cmap='gray')
plt.gcf().set_size_inches(8, 8)

```

```

sobel_x:
[[1, 0, -1], [2, 0, -2], [1, 0, -1]]
sobel_y:
[[1, 2, 1], [0, 0, 0], [-1, -2, -1]]

```



✓ 2.2 Implement a function that generates a 2D Gaussian filter given the parameter  $\sigma$ .

```
# Design the Gaussian filter
```

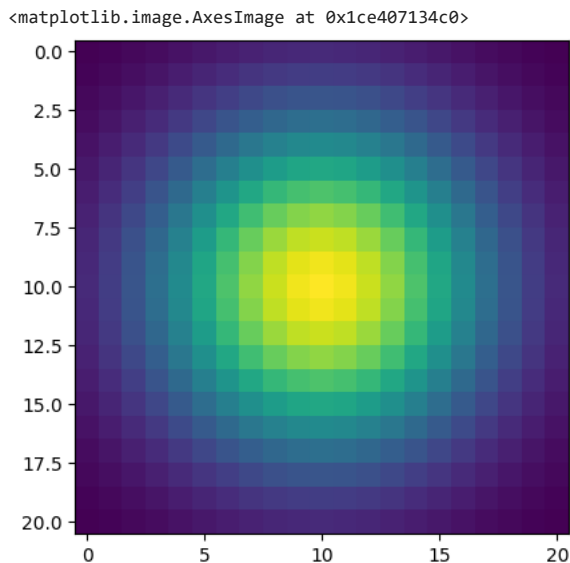
```
def gaussian_filter_2d(sigma):
    # sigma: the parameter sigma in the Gaussian kernel (unit: pixel)
    #
    # return: a 2D array for the Gaussian kernel

    ### Insert your code ###
    radius = 2*sigma
    size = int(2*radius + 1)

    h = np.fromfunction(
        lambda x, y: (1/(2*np.pi*sigma**2)) * np.exp(-((x-(size-1)/2)**2 + (y-(size-1)/2)**2)/(2*sigma**2)),
        (size, size)
    )
    return h / np.sum(h)
```

```
# Visualise the Gaussian filter when sigma = 5 pixel (provided)
```

```
sigma = 5
h = gaussian_filter_2d(sigma)
plt.imshow(h)
```



2.3 Perform Gaussian smoothing ( $\sigma = 5$  pixels) and evaluate the computational time for Gaussian smoothing.

After that, perform Sobel filtering and show the gradient magnitude map.

```
# Construct the Gaussian filter
```

```
### Insert your code ###
```

```
sigma = 5
h = gaussian_filter_2d(sigma)
```

```
# Perform Gaussian smoothing and count time
```

```
### Insert your code ###
```

```
start = time.time()
image_gaussian = scipy.signal.convolve2d(image_noisy, h, mode='full', boundary='fill', fillvalue=0)
finish = time.time()
print(f"Gaussian Smoothing Time Taken: {finish - start} seconds")
```

```
# Image filtering
```

```
### Insert your code ###
```

```
#sobel_x = scipy.signal.convolve2d(image_gaussian, sobel_x, mode='same', boundary='fill', fillvalue=0)
#sobel_y = scipy.signal.convolve2d(image_gaussian, sobel_x, mode='same', boundary='fill', fillvalue=0)
```

```
sobel_x = np.gradient(image_gaussian, axis=1)
sobel_y = np.gradient(image_gaussian, axis=0)
```

```
# Calculate the gradient magnitude
```

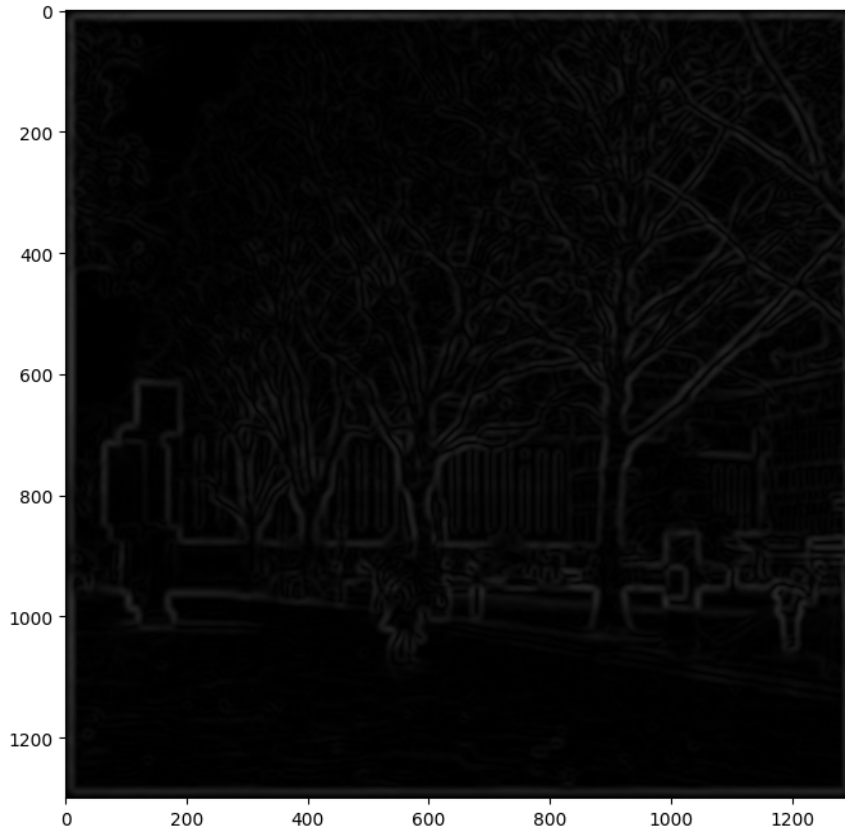
```
### Insert your code ###
```

```
grad_mag = np.sqrt(sobel_x**2 + sobel_y**2)
```

```
# Display the gradient magnitude map (provided)
```

```
plt.imshow(grad_mag, cmap='gray', vmin=0, vmax=100)
plt.gcf().set_size_inches(8, 8)
```

Gaussian Smoothing Time Taken: 1.7251667976379395 seconds



2.4 Implement a function that generates a 1D Gaussian filter given the parameter  $\sigma$ . Generate 1D Gaussian filters along x-axis and y-axis respectively.

```
# Design the Gaussian filter
def gaussian_filter_1d(sigma):
    # sigma: the parameter sigma in the Gaussian kernel (unit: pixel)
    #
    # return: a 1D array for the Gaussian kernel


    ### Insert your code ###
    size = int(4 * sigma + 1)
    x = np.arange(size) - (size-1) / 2
    h = 1 / (np.sqrt(2 * np.pi) * sigma) * (np.exp(-x**2 / (2 * sigma**2)))
    return (h / np.sum(h))

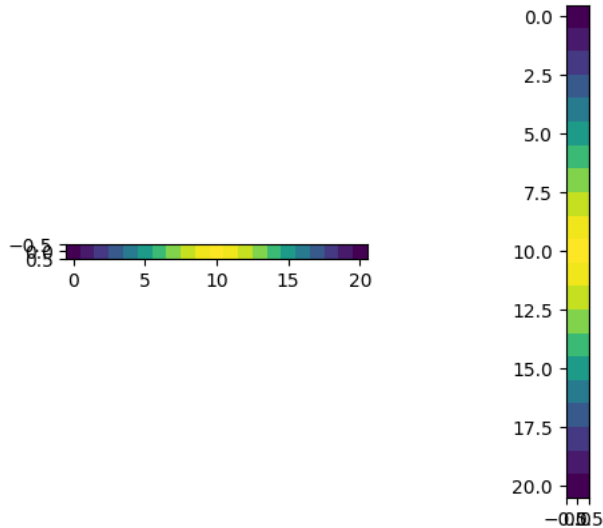
# sigma = 5 pixel (provided)
sigma = 5

# The Gaussian filter along x-axis. Its shape is (1, sz).
### Insert your code ###
h_x = gaussian_filter_1d(sigma)

# The Gaussian filter along y-axis. Its shape is (sz, 1).
### Insert your code ###
h_y = h_x.reshape((-1, 1))

# Visualise the filters (provided)
plt.subplot(1, 2, 1)
plt.imshow(h_x.reshape((1, -1)))
plt.subplot(1, 2, 2)
plt.imshow(h_y)
```

 <matplotlib.image.AxesImage at 0x1ce40aa3850>



2.6 Perform Gaussian smoothing ( $\sigma = 5$  pixels) using two separable filters and evaluate the computational time

- ✓ for separable Gaussian filtering. After that, perform Sobel filtering, show the gradient magnitude map and check whether it is the same as the previous one without separable filtering.

```
# Perform separable Gaussian smoothing and count time
### Insert your code ###
sigma = 5

start = time.time()
gaussian_horizontal = scipy.signal.convolve2d(image_noisy, h_x.reshape(1, -1), mode='full', boundary='fill', fillvalue=0)
gaussian_full = scipy.signal.convolve2d(gaussian_horizontal, h_y.reshape(-1, 1), mode='full', boundary='fill', fillvalue=0)
finish = time.time()

print(f"Gaussian Smoothing Time Taken: {finish - start} seconds")

# Image filtering
### Insert your code ###
sobel_x = np.gradient(gaussian_full, axis=1)
sobel_y = np.gradient(gaussian_full, axis=0)

# Calculate the gradient magnitude
### Insert your code ###
grad_mag2 = np.sqrt(sobel_x**2 + sobel_y**2)

# Display the gradient magnitude map (provided)
plt.imshow(grad_mag2, cmap='gray', vmin=0, vmax=100)
plt.gcf().set_size_inches(8, 8)

# Check the difference between the current gradient magnitude map
# and the previous one produced without separable filtering. You
# can report the mean difference between the two.
### Insert your code ###
mean_difference = np.mean(grad_mag2 - grad_mag)
print(f"Mean difference: {mean_difference}")
```



Gaussian Smoothing Time Taken: 0.3050558567047119 seconds  
 Mean difference: 2.5300623783222767e-16



2.7 Comment on the Gaussian + Sobel filtering results and the computational time.

✓ Insert your answer

As observed from the 1D and 2D Gaussian smoothing calculation results, it takes the 2D calculation on average, 1.3s longer or 400% more time to complete. The reason for this improvement can be explained by the size of the kernels. The 1D kernels used in separable smoothing are smaller than the 2D Gaussian kernel. For a Gaussian kernel of size  $N \times N$ , the separable approach involves two 1D kernels of size  $N \times 1$  and  $1 \times N$ . The total number of operations is proportional to  $N$  (size of the kernel) in each dimension, rather than  $N^2$  for a 2D convolution. Furthermore, algorithms for 1D convolution are often much more optimised than 2D convolution, which allows algorithms such as the Fast Fourier Transform to be employed in the 1D case. The visual results of both images appears to be the same, as we still smooth over the same number of pixels, and the convolution result is identical, but in the 1D case, it is much faster and less computationally taxing.

✓ 3. Challenge: Implement 2D image filters using Pytorch (24 points).

[Pytorch](#) is a machine learning framework that supports filtering and convolution.

The [Conv2D](#) operator takes an input array of dimension  $N \times C_1 \times X \times Y$ , applies the filter and outputs an array of dimension  $N \times C_2 \times X \times Y$ . Here, since we only have one image with one colour channel, we will set  $N=1$ ,  $C_1=1$  and  $C_2=1$ . You can read the documentation of Conv2D for more detail.

```
# Import libraries (provided)
import torch
```

✓ 3.1 Expand the dimension of the noisy image into  $1 \times 1 \times X \times Y$  and convert it to a Pytorch tensor.

```
# Expand the dimension of the numpy array
### Insert your code ###
image_expanded = np.expand_dims(image_noisy, axis=(0, 1)) # Adds new axis along the dimensions specified

# Convert to a Pytorch tensor using torch.from_numpy
### Insert your code ###
torch_image = torch.from_numpy(image_expanded) # Converts NumPy array to PyTorch tensor
print(torch_image.shape)

torch.Size([1, 1, 1280, 1280])
```

✓ 3.2 Create a Pytorch Conv2D filter set its kernel to be a 2D Gaussian filter and perform filtering