Coursework 1: Image filtering

The coursework includes coding questions and/or written questions. Please read both the text and the code in this notebook to get an idea what you are expected to implement.

What is expected?

- Complete and run the code using `jupyter-lab`.
- 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.ipynb --to pdf`
- If Jupyter complains issues during exporting, it is likely that pandoc 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, alternatively, you can use Google Colab to write the code and export the PDF file.

Dependencies:

You may need to install Jupyter-Lab and other libraries used in this coursework, such as by running the command: `pip3 install [package_name]`

In [1]:

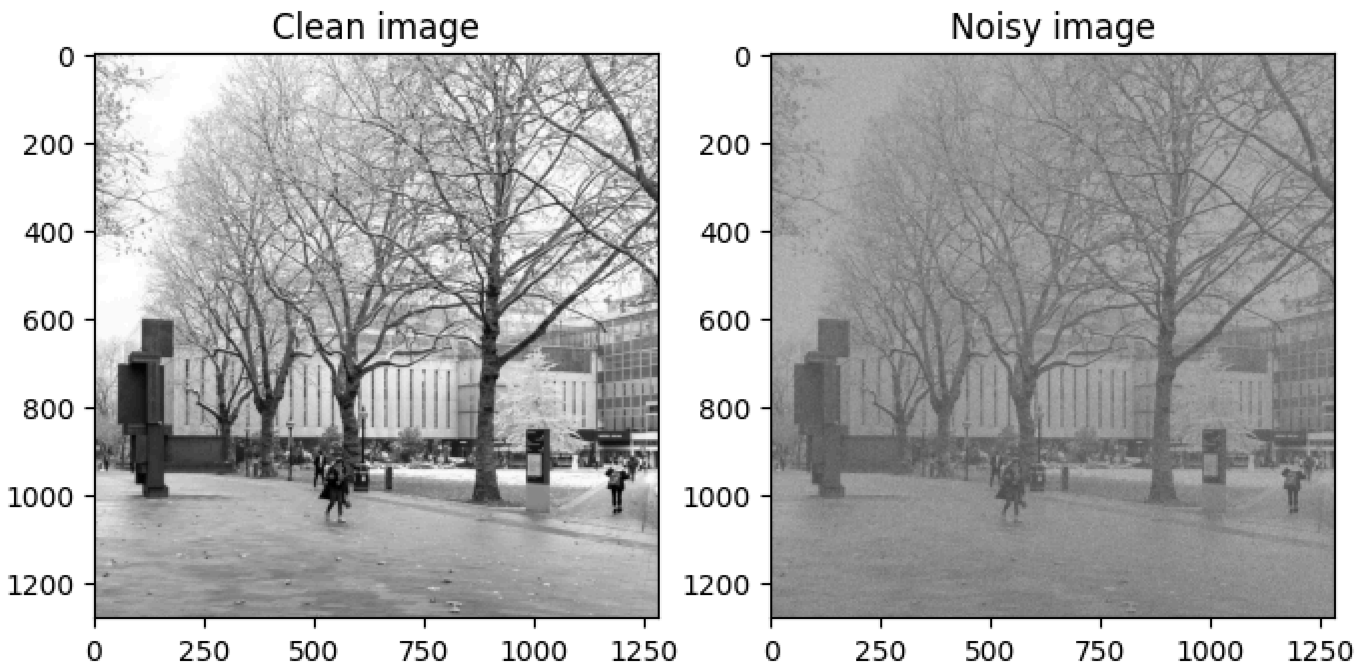
```python
# Import libaries (provided)
import imageio.v3 as imageio
import numpy as np
import matplotlib.pyplot as plt
import scipy
import scipy.signal
import math
import time
```

Q1. Moving average filtering (20 points).

Read the provided clean 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()`.

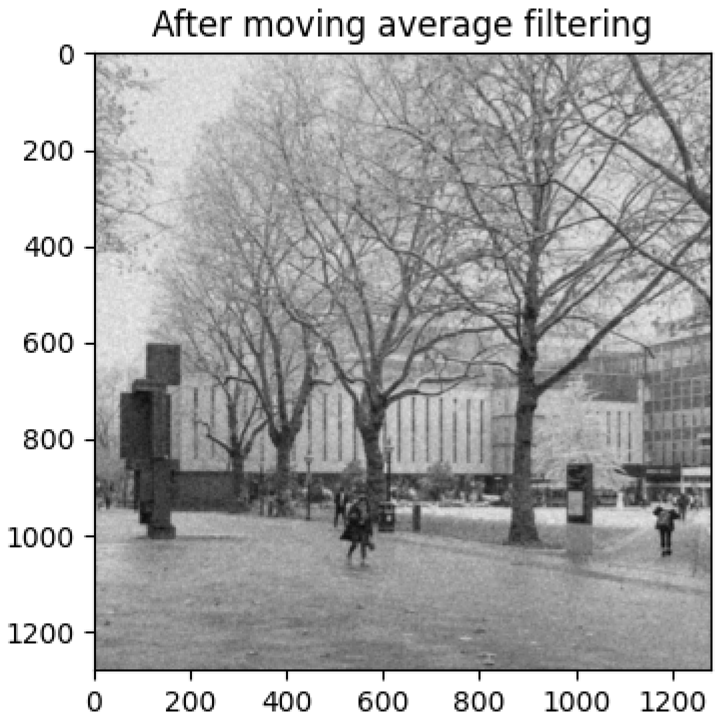
In [2]:
```
# Read the image (provided)
image = imageio.imread('campus_snow.jpg')

# Corrupt the image with Gaussian noise (provided)
noise_mu = 0
noise_sigma = 50
noise = np.random.normal(noise_mu, noise_sigma, image.shape)
image_noisy = image + noise

# Visualise the images (provided)
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Clean image')

plt.subplot(1, 2, 2)
plt.imshow(image_noisy, cmap='gray')
plt.title('Noisy image')
plt.gcf().set_size_inches(8, 4)
```



Q1.1 Filter the noisy image using a 5x5 moving average filter. Display the filtered image.

In [3]:
```
# Design the filter h
### Insert your code ###
# Creating 5x5 matrix, with each cell = 1/25
h = np.full((5,5), 1/25)

# Convolve the noisy image with h using scipy.signal.convolve2d function
### Insert your code ###
# Symmetrically reflecting the pixel values at the boundary when filtering
image_filtered = scipy.signal.convolve2d(image_noisy, h, mode='same', boundary='symm')

# Print the filter (provided)
print('Filter h = {0}'.format(h))

# Display the filtering result (provided)
plt.imshow(image_filtered, cmap='gray')
plt.title('After moving average filtering')
plt.gcf().set_size_inches(4, 4)
```

```
Filter h = [[0.04 0.04 0.04 0.04 0.04]
 [0.04 0.04 0.04 0.04 0.04]
 [0.04 0.04 0.04 0.04 0.04]
 [0.04 0.04 0.04 0.04 0.04]
 [0.04 0.04 0.04 0.04 0.04]]
```



Q1.2 Assess the quality of the filtered image using a quantitative metric, the peak signal-to-noise ratio (PSNR).

For this case, the pixel intensity of the image is represented using the uint8 format, with the peak value to be 255. The PSNR is defined as,

PSNR = $10 \cdot log_{10} \frac{255^2}{\frac{1}{N} \sum_x [J(x) - I(x)]^2}$

where x denotes the pixel index, N denotes the total number of pixels in the image, J denotes the filtered i.e. denoised image and I denotes the ground truth clean image. The denominator of the term within the logarithm operator is the mean squared error between I and J.

You can find more detail about PSNR here.

In [4]:
```python
# Implement the PSNR function
def eval_psnr(I, J):
    # I: the ground truth clean image (peak value: 255 for uint8 data format)
    # J: the denoised image
    #
    # return: the PSNR metric (unit: dB)

    ### Insert your code ###
    mse = np.sum((I - J)**2) / I.size
    psnr = 10 * math.log10(255**2 / mse)
    return psnr

# Evaluate the PSNR for the filtered image (provided)
psnr = eval_psnr(image, image_filtered)


# Print the PSNR (provided)
print('PSNR = {0:.2f} dB'.format(psnr))
```

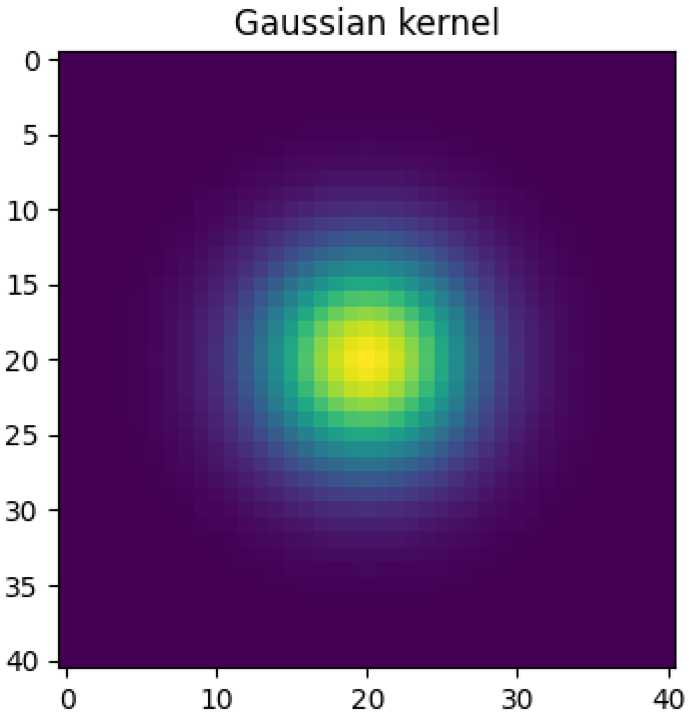
```
PSNR = 18.49 dB
```

Q2. Gaussian filtering (70 points).

Q2.1 Implement a function that constructs a 2D Gaussian filter given the parameter σ.

In [5]:
```python
# Implement the Gaussian filter
def gaussian_filter_2d(sigma):
    # sigma: the parameter sigma for the Gaussian kernel (unit: pixel)
    #
    # return: a 2D array for the Gaussian kernel

    # The filter radius is 4 times sigma (provided)
    rad = int(math.ceil(4 * sigma))
    sz = 2 * rad + 1

    # Calculate the filter weights
    ### Insert your code ###
    # Creating 2D array centred on (0,0)
    range_1d = np.arange(-rad, rad + 1)
    x, y = np.meshgrid(range_1d, range_1d)
```

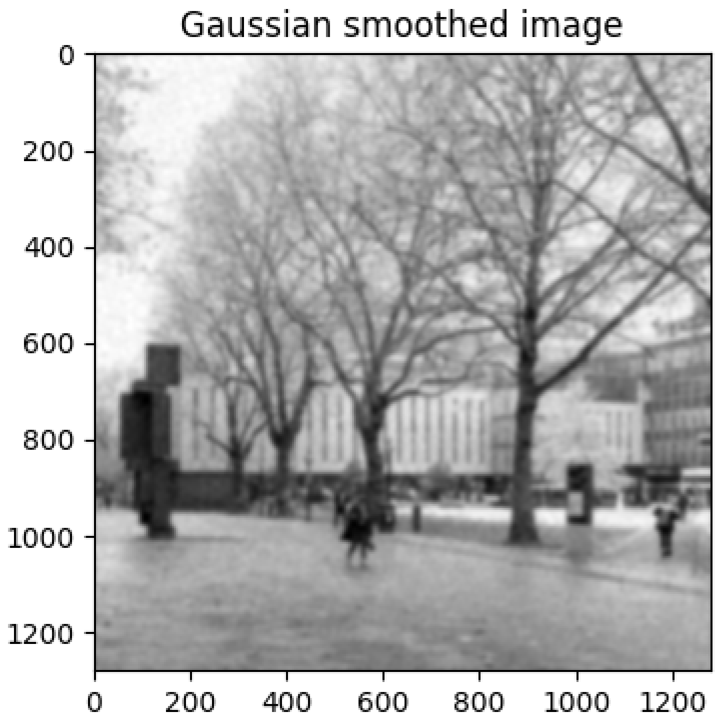
```python
    # Applying Gaussian formula
    h = np.exp(-(x**2 + y**2) / (2 * sigma**2))

    # Normalise by the sum (rather than the analytic 2 * pi * sigma^2)
    h /= h.sum()
    return h

# Visualise the Gaussian filter when sigma = 5 pixel (provided)
sigma = 5
h = gaussian_filter_2d(sigma)
plt.imshow(h)
plt.title('Gaussian kernel')
plt.gcf().set_size_inches(4, 4)
```



Q2.2 Perform Gaussian filtering (σ = 5 pixels) for the noisy image, evaluate the computational time for Gaussian filtering and display the filtered image.

In [6]:
```
# Construct the Gaussian filter (provided)
sigma = 5
h = gaussian_filter_2d(sigma)

# Perform Gaussian filtering and count time
### Insert your code ###
start = time.time()
# Using symmetric pixel values for filtering at the edge of the image
image_smoothed_2d = scipy.signal.convolve2d(image_noisy, h, mode='same', boundary='symm')

elapsed = time.time() - start
print("Time taken: {0:.4f} seconds".format(elapsed))

# Visualise the filtered image (provided)
plt.imshow(image_smoothed_2d, cmap='gray')
plt.title('Gaussian smoothed image')
plt.gcf().set_size_inches(4, 4)
```

```
Time taken: 1.5237 seconds
```

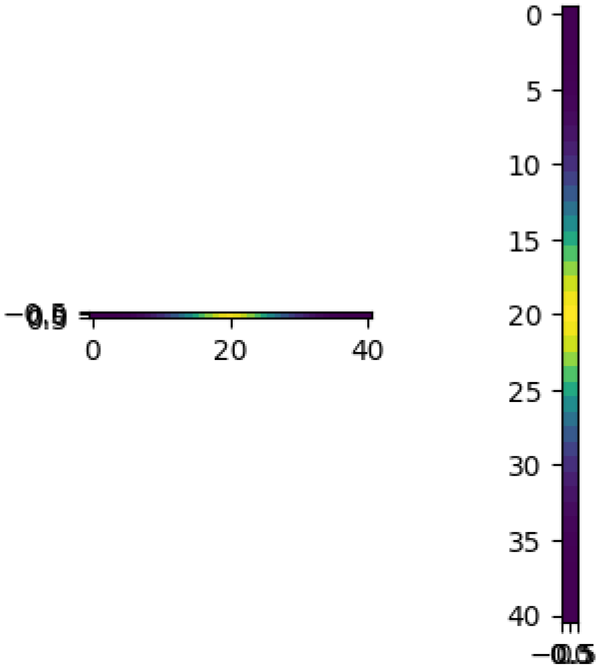


Q2.3 Implement a function that generates a 1D Gaussian filter given the parameter σ. Construct 1D Gaussian filters along x-axis and y-axis respectively.

In [7]:

```
# Implement the 1D 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

    # The filter radius is 4 times sigma (provided)
    rad = int(math.ceil(4 * sigma))
    sz = 2 * rad + 1

    # Calculate the filter weights
    ### Insert your code ###
    # First, generating an array of numbers of length sz centred on 0
    array = np.arange(-rad, rad+1)

    # Create 1D array using Gaussian formula
    h = np.exp(-array**2 / (2 * sigma**2))

    # Normalise by the sum (rather than the analytic sqrt(2 * pi) * sigma)
    h /= h.sum()
    return h

# sigma = 5 pixel (provided)
sigma = 5

# Construct the Gaussian filter along x-axis. Its shape is (1, sz).
### Insert your code ###
h = gaussian_filter_1d(sigma)
h_x = h.reshape(1, -1) # -1 inferred to be sz

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

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

```python
plt.subplot(1, 2, 2)
plt.imshow(h_y)
plt.gcf().set_size_inches(4, 4)
```



Q2.4 Perform Gaussian filtering (σ = 5 pixels) using two separable filters and evaluate the computational time for separable Gaussian filtering. Compare the smoothed image using separable filtering to the smoothed image using a single 2D Gaussian filter.

In [8]:

```python
# Perform separable Gaussian smoothing and count time
### Insert your code ###
start = time.time()
# First, filtering along the x-axis
# You could also use scipy.ndimage.convolve1d if you used h directly as a 1D array
# Rather than the reshaped h_x and h_y arrays.
image_smoothed = scipy.signal.convolve2d(image_noisy, h_x, mode='same', boundary='symm')
# Now along y-axis
image_smoothed = scipy.signal.convolve2d(image_smoothed, h_y, mode='same', boundary='symm')

elapsed = time.time() - start
print("Time taken: {0:.4f} seconds".format(elapsed))
```

```python
# Report the difference between the separably filtered image and the image filtered by a single 2D Gaussian filter (provided)
diff = image_smoothed - image_smoothed_2d
print('Mean absolute difference = {:.6f}'.format(np.mean(np.abs(diff))))
```

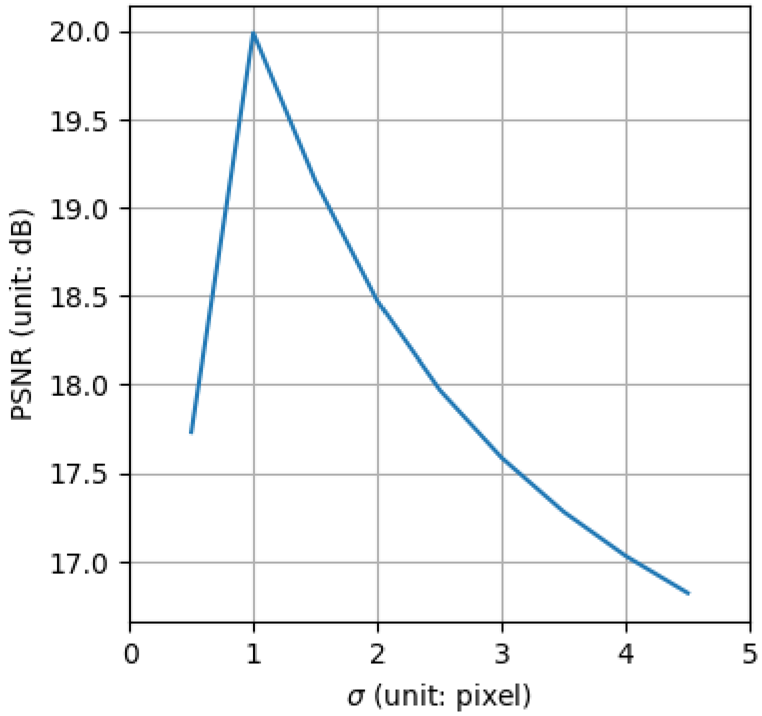
```
Time taken: 0.2007 seconds
Mean absolute difference = 0.000000
```

Q2.5 Perform Gaussian smoothing for the same noisy image, assess the quality of the Gaussian smoothed image using PSNR, when different sigma values are used.

In [9]:

```python
# A list of sigma values (provided)
list_sigma = np.arange(0.5, 5, 0.5)

# Perform Gaussian smoothing with different sigma values and record the PSNR values
### Insert your code ###
list_psnr = []
for sigma in list_sigma:
        h = gaussian_filter_1d(sigma)
        h_x = h.reshape(1, -1)
        h_y = h.reshape(-1, 1)

        # Separable filtering
        image_smoothed = scipy.signal.convolve2d(image_noisy, h_x, mode='same', boundary='symm')
        image_smoothed = scipy.signal.convolve2d(image_smoothed, h_y, mode='same', boundary='symm')

        list_psnr.append(eval_psnr(image, image_smoothed))


# Plot the PSNR metric against sigma (provided)
plt.plot(list_sigma, list_psnr)
plt.xlim([0, 5])
plt.xlabel('$\sigma$ (unit: pixel)')
plt.ylabel('PSNR (unit: dB)')
plt.grid()
plt.gcf().set_size_inches(4, 4)
```

```
<>:22: SyntaxWarning: "\s" is an invalid escape sequence. Such sequences will not work in the future. Did you mean "\\s"? A raw string is also an option.
<>:22: SyntaxWarning: "\s" is an invalid escape sequence. Such sequences will not work in the future. Did you mean "\\s"? A raw string is also an option.
/var/folders/pt/5mg57l7j27b2bbpf3wt5bm540000gn/T/ipykernel_79377/1282904532.py:22: SyntaxWarning: "\s" is an invalid escape sequence. Such sequences will not work in the future. Did you mean "\\s"? A raw string is also an option.
  plt.xlabel('$\sigma$ (unit: pixel)')
```

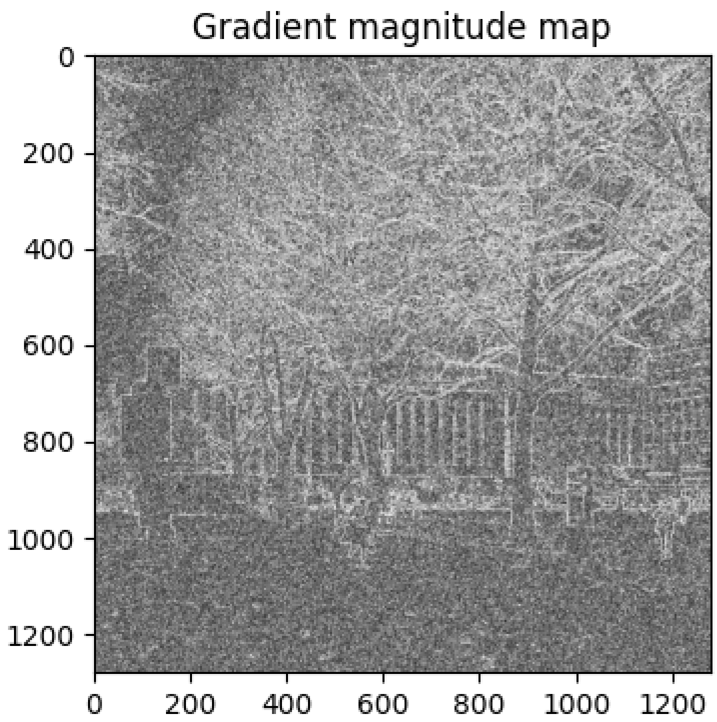


Q2.6 Implement 3x3 Sobel filters, perform Sobel filtering for the noisy image, and display the gradient magnitude map.

In [10]:
```
# Construct the Sobel filters
### Insert your code ###
sobel_x = np.array([[1,0,-1],
                    [2,0,-2],
                    [1,0,-1]])
sobel_y = sobel_x.T

# Print the filters (provided)
```

```python
print('Sobel_x = {0}'.format(sobel_x))
print('Sobel_y = {0}'.format(sobel_y))

# Sobel filtering for the noisy image
def sobel_filter(image: np.ndarray):
    # Note: I separate the x- and y-axis Sobel filters into 1D vectors to speed up the convolutions
    # Initialising the separable row vectors (with shape (1,3))
    weight_vector = np.array([[1,2,1]])
    diff_vector = np.array([[1,0,-1]])

    # First I apply the Sobel filter along the x axis (i.e. getting the gradients with respect to the x-axis)
    g_x = scipy.signal.convolve2d(image, weight_vector.T, mode='same', boundary='symm')
    g_x = scipy.signal.convolve2d(g_x, diff_vector, mode='same', boundary='symm')

    # Now filtering along the y axis (note the vectors are different)
    g_y = scipy.signal.convolve2d(image, diff_vector.T, mode='same', boundary='symm')
    g_y = scipy.signal.convolve2d(g_y, weight_vector, mode='same', boundary='symm')

    return g_x, g_y

g_x, g_y = sobel_filter(image_noisy)

# Calculate the gradient magnitude
### Insert your code ###
grad_mag_noisy = np.sqrt(g_x**2 + g_y**2)

# Display the magnitude map (provided)
plt.imshow(grad_mag_noisy, cmap='gray', vmin=0, vmax=500)
plt.title('Gradient magnitude map')
plt.gcf().set_size_inches(4, 4)
```

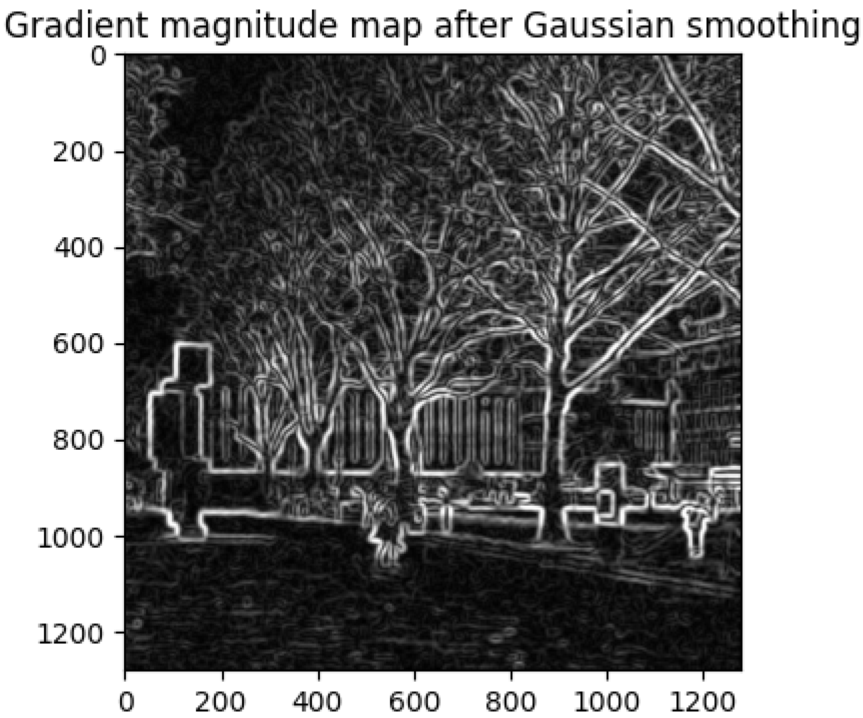
```
Sobel_x = [[ 1  0 -1]
 [ 2  0 -2]
 [ 1  0 -1]]
Sobel_y = [[ 1  2  1]
 [ 0  0  0]
 [-1 -2 -1]]
```



Q2.7 Perform Gaussian smoothing for the noisy image, followed by Sobel filtering and display the gradient magnitude map.

In [11]:
```
# Parameter for the Gaussian filter (provided)
sigma = 5

# Gaussian smoothing
### Insert your code ###

# Copied this code from Q2.4 (using 2 separable filters)
# Smoothing along x-axis
image_smoothed = scipy.signal.convolve2d(image_noisy, h_x, mode='same', boundary='symm')
# Now along y-axis
image_smoothed = scipy.signal.convolve2d(image_smoothed, h_y, mode='same', boundary='symm')


# Sobel filtering
### Insert your code ###
```

```
smooth_g_x, smooth_g_y = sobel_filter(image_smoothed)

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

# Display the magnitude map (provided)
plt.imshow(grad_mag, cmap='gray', vmin=0, vmax=100)
plt.title('Gradient magnitude map after Gaussian smoothing')
plt.gcf().set_size_inches(4, 4)
```



Q3. Implement image filters using Pytorch (10 points).

Pytorch is a machine learning framework that supports filtering and convolution.

The Conv2D operator takes an input array of dimension NxC1xXxY, applies the filter and outputs an array of dimension NxC2xXxY. Here, since we only have one image with one colour channel, we will set N=1, C1=1 and C2=1. You can read the documentation of Conv2D for more detail.

In [12]:

```python
# Import libaries (provided)
import torch
```

Q3.1 Expand the dimension of the noisy image into 1x1xXxY and convert it to a Pytorch tensor.

In [13]:

```python
# Expand the dimension of the numpy array
### Insert your code ###
expanded_image = np.expand_dims(image_noisy, axis=(0, 1))

# Convert to a Pytorch tensor using torch.from_numpy
### Insert your code ###

# I convert it to float32 to speed up computational time (it took ages on my computer for some reason)
tensor_noisy = torch.from_numpy(expanded_image).float()
```

Q3.2 Create a Pytorch Conv2D filter, set its kernel to be a 2D Gaussian filter, perform filtering, report computational time and display the result.

In [14]:

```python
# A 2D Gaussian filter when sigma = 5 pixel (provided)
sigma = 5
h = gaussian_filter_2d(sigma)

# Construct the Conv2D filter
### Insert your code ###
# Converting to float32 as in Q3.1
filter = torch.from_numpy(np.expand_dims(h, axis=(0, 1))).float()

# Filtering and assess computational time
### Insert your code ###
start = time.time()

# Zero-padding so the resulting image is the same size as the original
image_filtered = torch.nn.functional.conv2d(tensor_noisy, filter, padding='same')

elapsed = time.time() - start
print("Time taken: {0:.4f} seconds".format(elapsed))

# squeezing tensor back to 2D
image_filtered = image_filtered.squeeze().numpy()
```

```
# Display the filtering result (provided)
plt.imshow(image_filtered, cmap='gray')
plt.title('Gaussian smoothed image using PyTorch')
plt.gcf().set_size_inches(4, 4)
```

```
Time taken: 37.4893 seconds
```