In solving the questions in this assignment, I worked together with my classmate Shaoxuan Zheng 1005251503. I confirm that I have written the solutions / code / report in my own words

Part I Q1 :

$x(n)$ can be written as the sum of $\delta(n)$

$$x(n) = \sum_{i=-\infty}^{\infty} x(i) \, \delta(n-i)$$

validation : Let j be an arbitrary input

$$\text{LHS} = x(j)$$

$$\text{RHS} = \sum_{i=-\infty}^{\infty} x(i) \, \delta(j-i)$$

Since $i \neq j \Rightarrow \delta(j-i) = 0$

We are left with only 1 term where $i = j$

$$\text{RHS} = x(j) \, \delta(j-j) = x(j) \, \delta(0) = x(j) = \text{LHS}$$

So $T[x(n)] = T\left[\sum_{i=-\infty}^{\infty} x(i) \, \delta(n-i)\right]$

$$= \sum_{i=-\infty}^{\infty} T[x(i) \, \delta(n-i)] \quad \text{(since T is linear)}$$

$$= \sum_{i=-\infty}^{\infty} x(i) \, T[\delta(n-i)]$$

$$= \sum_{i=-\infty}^{\infty} x(i) \, h(n-i)$$

$$= x * h \quad \text{(by definition of convolution)}$$

$$= h(n) * x(n) \quad \text{(law of commulative)}$$

□

Part I Q2 :

Let $P(x)$, $Q(x)$ be two polynomials with degree n, m

Let u be the vector representation of P with length $n+1$

Let v be the vector representation of Q with length $m+1$

Assume proper zero-padding and full-size convolution

By convention, $u[i] = \begin{cases} u[i] & 0 \leq i \leq n \\ 0 & i < 0 \text{ or } i > n \end{cases}$ $v[i] = \begin{cases} v[i] & 0 \leq i \leq m \\ 0 & i < 0 \text{ or } i > m \end{cases}$

The reason that it becomes 0 when index is going

out of the bond is because we are doing zero-padding

Define $G(i) = [u * v](i)$

$$= \sum_{j=-\infty}^{\infty} u[j]\, v[i-j]$$

$$= \sum_{j=0}^{i} u[j]\, v[i-j] \quad \text{(by our convention)}$$

Since degrees of P and Q are n, m

$$P(x)\, Q(x) = \sum_{i=0}^{m+n} C_i\, x^i$$

where $C_i = \sum_{j=0}^{i} P_j\, q_{i-j}$

$$= \sum_{j=0}^{i} u[j]\, v[i-j]$$

$$= G(i)$$

Thus, polynomial multiplication is equivalent to convolve

their vector representation

Part I Q3:

By definition of Laplacian, we know that

$$L_i = I_i - Up(I_{i+1}) \quad \text{(where } Up() \text{ is upsampling method)}$$

$$\Rightarrow I_i = L_i + Up(I_{i+1})$$

Since we have all the L_i's, the minimum information

we need to calculate I_0 is I_n

Listing first few terms.

$$I_{n-1} = L_{n-1} + Up(I_n)$$

$$I_{n-2} = L_{n-2} + Up(L_{n-1} + Up(I_n))$$

$$= L_{n-2} + Up(L_{n-1}) + Up^2(I_n)$$

$$I_{n-3} = L_{n-3} + Up(L_{n-2} + Up(L_{n-1}) + Up^2(I_n))$$

$$= L_{n-3} + Up(L_{n-2}) + Up^2(L_{n-1}) + Up^3(I_n)$$

Claim that $I_{n-k} = \sum_{i=0}^{k} Up^i(L_{n-k+i})$ for simplicity, assume $I_n = L_n$

Base case: when $k=0$

then $I_n = \sum_{i=0}^{0} Up^i(L_{n+i}) = L_n$ ✓

Inductive step: assume $I_{n-k+1} = \sum_{i=0}^{k-1} Up^i(L_{n-k+1+i})$ (I.H)

WTP: $I_{n-k} = \sum_{i=0}^{k} Up^i(L_{n-k+i})$

$$I_{n-k} = L_{n-k} + Up(I_{n-k+1})$$

$$= L_{n-k} + Up\left(\sum_{i=0}^{k-1} Up^i(L_{n-k+1+i})\right) \quad (\text{by I.H})$$

$$= L_{n-k} + \sum_{i=0}^{k-1} Up^{i+1}(L_{n-k+1+i})$$

$$= L_{n-k} + \sum_{i=1}^{k} Up^i(L_{n-k+i})$$

$$= \sum_{i=0}^{k} Up^i(L_{n-k+i}) \qquad \square$$

So, when $k=n \Rightarrow I_0 = \sum_{i=0}^{n} Up^i(L_i)$ when $L_n = I_n$

So the minimimum information we need from Gaussian pyramid

is I_n

Q1. part 4: Need to show

$$I_{xx} + I_{yy} = I_{rr} + I_{r'r'}$$

or $\dfrac{\partial^2 f}{\partial x^2} + \dfrac{\partial^2 f}{\partial y^2} = \dfrac{\partial^2 f}{\partial r^2} + \dfrac{\partial^2 f}{\partial r'^2}$

Suppose (r, r') is rotated from (x, y) with a radius θ

Based on the rotation formula

$$\begin{cases} r = x\cos\theta - y\sin\theta \\ r' = x\sin\theta + y\cos\theta \end{cases}$$

Starting with $\dfrac{\partial f}{\partial x}$

From MAT 237, we know

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial r} \cdot \frac{\partial r}{\partial x} + \frac{\partial f}{\partial r'} \cdot \frac{\partial r'}{\partial x}$$

$$= \cos\theta \, \frac{\partial f}{\partial r} + \sin\theta \, \frac{\partial f}{\partial r'}$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial r} \cdot \frac{\partial r}{\partial y} + \frac{\partial f}{\partial r'} \cdot \frac{\partial r'}{\partial y}$$

$$= -\sin\theta \, \frac{\partial f}{\partial r} + \cos\theta \cdot \frac{\partial f}{\partial r'}$$

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial x} \right)$$

$$= \frac{\partial}{\partial x}\left(\cos\theta \, \frac{\partial f}{\partial r} + \sin\theta \, \frac{\partial f}{\partial r'} \right)$$

$$= \frac{\partial r}{\partial x} \cdot \frac{\partial}{\partial r}\left(\cos\theta \, \frac{\partial f}{\partial r} + \sin\theta \, \frac{\partial f}{\partial r'} \right) + \frac{\partial r'}{\partial x} \cdot \frac{\partial}{\partial r'}\left(\cos\theta \, \frac{\partial f}{\partial r} + \sin\theta \, \frac{\partial f}{\partial r'} \right)$$

$$= \cos\theta \cdot \cos\theta \cdot \frac{\partial^2 f}{\partial r^2} + \sin\theta \cos\theta \cdot \frac{\partial^2 f}{\partial r r'} + \sin\theta \cos\theta \, \frac{\partial^2 f}{\partial r' r} + \sin^2\theta \, \frac{\partial^2 f}{\partial r'^2}$$

$$= \cos^2\theta \, \frac{\partial^2 f}{\partial r^2} + 2\sin\theta\cos\theta \, \frac{\partial^2 f}{\partial r r'} + \sin^2\theta \, \frac{\partial^2 f}{\partial r'^2}$$

$$\frac{\partial^2 f}{\partial y^2} = \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial y}\right)$$

$$= \frac{\partial}{\partial y}\left(-\sin\theta \frac{\partial f}{\partial r} + \cos\theta \cdot \frac{\partial f}{\partial r'}\right)$$

$$= \frac{\partial r}{\partial y} \cdot \frac{\partial}{\partial r}\left(-\sin\theta \frac{\partial f}{\partial r} + \cos\theta \cdot \frac{\partial f}{\partial r'}\right) + \frac{\partial r'}{\partial y} \cdot \frac{\partial}{\partial r'}\left(-\sin\theta \frac{\partial f}{\partial r} + \cos\theta \cdot \frac{\partial f}{\partial r'}\right)$$

$$= \sin^2\theta \frac{\partial^2 f}{\partial r^2} - \sin\theta\cos\theta \frac{\partial f}{\partial rr'} - \sin\theta\cos\theta \frac{\partial^2 f}{\partial rr'} + \cos^2\theta \frac{\partial^2 f}{\partial r'^2}$$

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = (\sin^2\theta + \cos^2\theta)\frac{\partial^2 f}{\partial r^2} + (\sin^2\theta + \cos^2\theta)\frac{\partial^2 f}{\partial r'^2}$$

$$= \frac{\partial^2 f}{\partial r^2} + \frac{\partial^2 f}{\partial r'^2}$$

Report for part 2

Step 1: The first row of the figure is showing the result after calling gaussian filter function (see step 4). Parameters are size = 10, sigma = 3 and size = 25, sigma = 5. The first one is the gaussian filter I use to blur the image. This is the function that I implemented for gaussian. It takes size and sigma as inputs and outputs a gaussian filter matrix. Details are commented in a1_code.py file.

```python
def gaussian_filter(size: int, sig: float):
    # Calculate the value for the left most and right most position and
    # using linspace to create the coresponding vector
    # for ex, 5 -> [-2, -1, 0, 1, 2]
    # 6 -> [-2.5, -1.5, -0.5, 0.5, 1.5, 2.5]
    right_end = (size - 1) / 2.0
    left_end = -right_end
    # The values in x are values that are going to be sub in gaussian formula
    x = np.linspace(left_end, right_end, size)

    # applying gaussian formula to all entries in x
    gaussian = np.exp(-0.5 * np.square(x) / np.square(sig))

    # taking product of 1d gaussian vector will give us 2D gaussian filter matrix
    gaussian_2d = np.outer(gaussian, gaussian)

    # normalize the entries
    gaussian_2d /= gaussian_2d.sum()
    return gaussian_2d
```

The result of images after taking a gaussian blur is listed in the second column in the figure (see step 4).

Step 2: Below is the function that I use for convolution. It uses valid option here so the result image will be smaller than the original one.

```python
def convolution(image, filter):
    assert filter.shape[0] == filter.shape[1], 'filter has to be k by k'
    # flip the fiter first
    filter = np.flip(filter)
    k = filter.shape[0]
    m, n = image.shape
    # Since I'm doing valid option here, so the result img size will be smaller
    m = m - k + 1
    n = n - k + 1
    res = np.zeros((m, n))
    # loop through each entry to calculate the result
    for i in range(m):
        for j in range(n):
            res[i][j] = np.sum(image[i:i + k, j:j + k] * filter)
    return res
```

This function only works for the filters in k x k size, so it takes an 2D image and a filter as an input and returns the result image after applying the filter.

Below is the function of taking gradient magnitude. It takes image as input, convolve it with sober matrixes and calculates the magnitude for each entry.

```python
def gradient_magnitude(image):
    # input image is already in gray scale
    # define sober filter
    sober_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sober_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
    # using covolution with sober matrix to get derivatives.
    g_x = convolution(image, sober_x)
    g_y = convolution(image, sober_y)
    # Calculate gradient matrix
    res = np.sqrt(g_x ** 2 + g_y ** 2)
    return res
```

The result images are listed in the third column (see step 4).

Step 3: Below is the function that I use to apply the threshold.

```python
58  ∨ def threshold(image, epsilon):
59          # calcutate the matrix mean
60          t = image.mean()
61          # initial value of t_{i - 1} should be inf so the first iteration will start
62          t_last = float('inf')
63  ∨      while abs(t - t_last) > epsilon:
64              t_last = t
65              # calcutate low average and high average with last t value
66              low_avg = image.mean(where=image < t_last)
67              high_avg = image.mean(where=image >= t_last)
68              t = (low_avg + high_avg) / 2
69          # map every entry to 0 or 255 based on the condition
70          mapping_func = np.vectorize(lambda x: 255 if x >= t else 0)
71          res = mapping_func(image)
72          return res
73
74
```

It basically follows the logic described in assignment. The epsilon that I used

in the figure above is 0.0001. These parameters can be altered in line 76 –

80 in a1_code.py

Step 4: testing
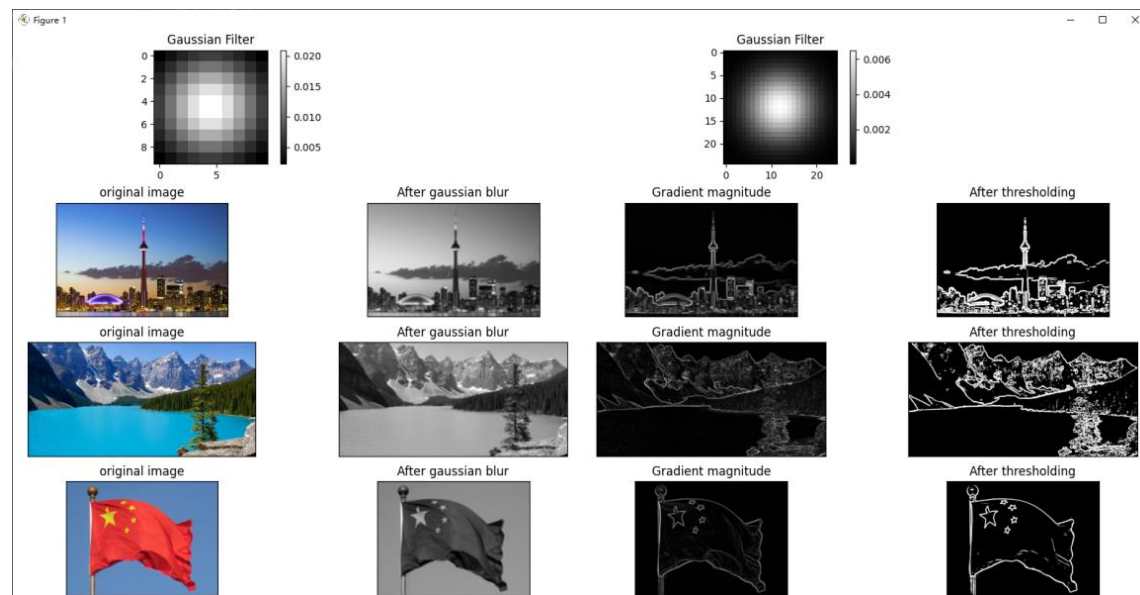
This is the result after running the program. The second and the third rows are images provided on quercus and the fourth row is the image that I chose.

There are 4 columns:

   i) The original image.

   ii) The image after taking gaussian blur.

   iii) The image after taking gradient magnitude.

   ix) The image after applying threshold.

Also shape of images in each step are printed out.


```
The shape of original image 1 is  (650, 980, 3)
The shape of image 1 after applying gaussian blur is  (641, 971)
The shape of image 1 after taking gradient magnitude is  (639, 969)
The shape of image 1 after applying threshold is  (639, 969)
The shape of original image 2 is  (900, 1800, 3)
The shape of image 2 after applying gaussian blur is  (891, 1791)
The shape of image 2 after taking gradient magnitude is  (889, 1789)
The shape of image 2 after applying threshold is  (889, 1789)
The shape of original image 3 is  (960, 1280, 3)
The shape of image 3 after applying gaussian blur is  (951, 1271)
The shape of image 3 after taking gradient magnitude is  (949, 1269)
The shape of image 3 after applying threshold is  (949, 1269)
```

After thresholding

After thresholding
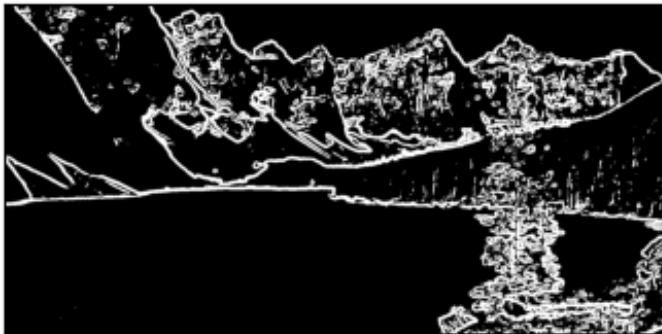
Strength: Most of the information is preserved. By just looking at the result image, we could image what it is in original image. For example, I know that the first image is a picture of city with a lot of buildings and a tower that is much taller than other building. Also, there are clouds in the background. For the second image, I know that it should be picture of a tree and mountains.

Weakness: Some information is lost as well. For example,

After thresholding

The part circled in red, this edge doesn't survive the threshold because it is too weak. However, it should be preserved because we know that tower is not part of the cloud. Also,

After thresholding

This part is a bit noisy. If we increase the sigma value, that will risk of losing important information. The values of parameters are hard to control.