MASD Assignment 4

MASD 2018
Department of Computer Science
University of Copenhagen

Casper Lisager Frandsen <fsn483@alumni.ku.dk>

Version 1
Due: 30 September 2018, 10:00

Contents

1. Exercise

(a)

For this, we start by rewriting the orthonormal basis b_1, b_2 to a matrix B:

$$B = \begin{bmatrix} \frac{-\sqrt{2}}{2} & \frac{-\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{-\sqrt{2}}{2} \end{bmatrix}$$

We can then calculate the product of the different vectors with this matrix and find their coordinates in the basis B. I will label the three vectors given as $a = (0,0), b = (1,0), c = (2,3)$ respectively.

$$Ba = (0,0)_B$$

$$Bb = (\frac{-\sqrt{2}}{2} + 0, \frac{\sqrt{2}}{2} + 0)_B = (\frac{-\sqrt{2}}{2}, \frac{\sqrt{2}}{2})_B$$

$$Bc = (2\frac{-\sqrt{2}}{2} + 3\frac{\sqrt{2}}{2}, 2\frac{\sqrt{2}}{2} + 3\frac{-\sqrt{2}}{2})_B = (-5\frac{\sqrt{2}}{2}, \frac{-\sqrt{2}}{2})_B$$

(b)

To prove this, we need to state a few things:
We know that it is possible to convert the basis a vector is shown in, by multiplying the vector onto the new basis $B = \{b_1, b_2\}$. We will do the following:

First we will show that $(x - y) = (x_b - y_b)$. We do this by making a combined vector of $(x - y)$:

$$(x - y) = \begin{pmatrix} x_1 - y_1 \\ x_2 - y_2 \end{pmatrix}$$

Now we can show the vector $(x - y)$ by showing $B(x - y)$:

$$B(x-y) = \begin{bmatrix} (b_1)_1 & (b_2)_1 \\ (b_1)_2 & (b_2)_2 \end{bmatrix} \begin{bmatrix} x_1 - y_1 \\ x_2 - y_2 \end{bmatrix} = \begin{bmatrix} (b_1)_1(x_1 - y_1) + (b_2)_1(x_2 - y_2) \\ (b_1)_2(x_1 - y_1) + (b_2)_2(x_2 - y_2) \end{bmatrix} = (x_b - y_b)$$

Since we have shown that $(x - y) = (x_b - y_b)$ we can say the following:

$$||x - y|| = \sqrt{x^2} - \sqrt{y^2} = \sqrt{x_b^2} - \sqrt{y_b^2} = ||x_b - y_b||$$

(c)

To determine whether a point is on the "green" or "blue" side of the subspace, we have to make observation that the normal vector is always perpendicular to the subspace. This means that if we calculate the dot product of the vector and any point, the result will be positive if it is on one side of the subspace, and negative on the other. Thus, we have an easy way to determine which side it is on.

2. Exercise

(a)

Since it is assumed in this case that the length of time between measurements is 1 unit, we can simply sum all of the elements. This has been implemented in the Jupyter Notebook.

(b)

$P(n)$, where $n \in N$ represents the integral of the function up until the point n, which can be seen as the upper limit of the integral.

(c)

There would need to be a few changes to the function for it to work with variable lengths:

First, we would have to find a way to calculate the length between the different inputs. $(x_i - x_{i-1})$. This would require a different type of input, perhaps an array of tuples, each with the value of the function, and the distance from the previous element.

Second, we would also have to multiply the value of each element by this length, to find the approximation of the area. Only then would we sum everything together as before.

3. Exercise

(a)

To find the discrete convolution of $f[x]$ we have to compute the following function:

$$\sum_{n=0}^{m} f[n] * g[m-n]$$

Where, m is the index of $f * g$. We will show the procedure only for the first and last few entries:

$$(f * g)[0] = f[0] * g[0-0] = 1 * 1 = 1$$

$$(f * g)[1] = f[0] * g[1-0] + f[1] * g[1-1] = 1 * 1 + 2 * 1 = 3$$

...

$(f * g)[11] = f[0] * g[11-0] + f[1] * g[10-1] + ... + f[10] * g[10-10] = 7 * 0 + 2 + 4 * 0 = 2$
Before $f[0]$ and after $f[10]$ the functions will simply evaluate to 0. This means

that $(f * g)$ is 11 units long, ignoring all edges where $(f * g) = 0$, because all indices in the function would be out of bounds in both arrays:

$$(f * g) = [1, 3, 2, -2, -2, 0, -2, -4, -1, 3, 2]$$

(b)

The result of the convolution of f with g is a longer signal, where each element consist of the sum of between 1 and 4 elements from f weighted by g

(c)

(d)

4. Exercise

This exercise has been completed in the attached Jupyter Notebook.
The end result of my code seems to have more noise than the guideline pictures. It seems that this noise comes from the algorithm used in exercise 4b, from which the output picture seems to be brighter than the guideline, which may lead to more noise being picked up by the later parts. It is very likely from this exercise that the errors in later exercises appear.

5. Appendix

(a)

```
# Importing packages -- feel free to add more, but recall that you should not use built-in
# functions for the task at hand. If in doubt, ask us by email.
from scipy import signal as sg
import numpy as np
import matplotlib.pyplot as plt
import skimage
from skimage import io
from skimage import color
from matplotlib import cm
import matplotlib
%pylab inline
pylab.rcParams['figure.figsize'] = (12, 6)
matplotlib.rcParams['figure.dpi']=150
```

(b)

```
data = np.loadtxt('data/precipitation.txt')
#print(data)
def evenIntegral(inArr):
```

```
    return sum(inArr)

print("Integral up to 182:\t\t", evenIntegral(data[:182]))
print("Integral of entire set:\t\t",evenIntegral(data))
```

(c)

```
# Read and view the coins.png image
coin = color.rgb2gray(io.imread('data/coins.png'))

# Uncomment if, alternatively, you want to test with the cameraman.png image
f = io.imread('data/cameraman.png')

# Uncomment if you want to view the image:
io.imshow(f, cmap=cm.Greys_r)
def convolve(image):
    sigma = 1.4
    X, Y = np.mgrid[-2:3, -2:3]
    g = np.exp(-(X**2 + Y**2)/(2*sigma**2))
    g /= g.sum()
    return sg.convolve2d(image, g, mode='same')

plt.imshow(convolve(f), cmap=cm.Greys_r)
plt.title('Cameraman.png convolved with a Gaussian filter')
plt.show()

plt.imshow(convolve(coin), cmap=cm.Greys_r)
plt.title('Coins.png convolved with a Gaussian filter')
```

(d)

```
gx = np.matrix([[1, 0, -1],
                [2, 0, -2],
                [1, 0, -1]])
gy = np.matrix([[-1,-2,-1,],
                [ 0, 0, 0],
                [ 1, 2, 1]])

def sobelD(image, matrix):
    return sg.convolve2d(image, matrix, mode='same')

def gradeMag(image):
    N, M = image.shape

    tempXDiff = sobelD(image, gx)
    tempYDiff = sobelD(image, gy)
    newImage = np.sqrt(tempXDiff**2 + tempYDiff**2)
```

```
    return newImage


plt.title("Gradient Magnitude")
plt.imshow(gradeMag(convolve(f)), cmap=cm.Greys_r)

plt.show()
plt.title("Gradient Magnitude")
plt.imshow(gradeMag(convolve(coin)), cmap=cm.Greys_r)
```

(e)

```
def nMSuppr(image):
    X,Y = image.shape
    #print(X)
    xImg = sobelD(image, gx)
    yImg = sobelD(image, gy)
    magImg = gradeMag(convolve(image))

    for i in range(X-1):
        for j in range(Y-1):
            if (j > 0 and i > 0):

                if (xImg[i,j] == 0):
                    angle = 0
                else:
                    angle = np.degrees(arctan(yImg[i,j]/xImg[i,j]))
                    angle = round(angle/45)*45

                if (angle == 45 and (magImg[i,j] <= magImg[i+1,j-1] or magImg[i,j] <= magI
                    magImg[i,j] = 0
                elif(angle == -45 and (magImg[i,j] <= magImg[i+1,j+1] or magImg[i,j] <= ma
                    magImg[i,j] = 0
                elif ((angle == 0 or angle == -0) and (magImg[i,j] <= magImg[i,j+1] or mag
                    magImg[i,j] = 0
                elif ((angle == 90 or angle == -90) and (magImg[i,j] <= magImg[i+1,j] or m
                    magImg[i,j] = 0
    return magImg

f = io.imread('data/cameraman.png')

plt.title("Non-maximum Suppression")
plt.imshow(nMSuppr(f), cmap=cm.Greys_r)

plt.show()
coin = color.rgb2gray(io.imread('data/coins.png'))

plt.title("Non-maximum Suppression")
```

```
plt.imshow(nMSuppr(coin), cmap=cm.Greys_r)
```

(f)

```
def dbThr(image, min_t, max_t):

    X,Y = convolve(image).shape
    supprImg = nMSuppr(image)
    maxVal = 0

    for i in range(X):
        for j in range(Y):
            if supprImg[i,j] > maxVal:
                maxVal = supprImg[i,j]

    for i in range(X):
        for j in range(Y):
            supprImg[i,j] = supprImg[i,j]/maxVal
            if (supprImg[i,j] > max_t):
                image[i,j] = 255
            elif (supprImg[i,j] < min_t):
                image[i,j] = 0
            else:
                image[i,j] = 100
    return image

f = io.imread('data/cameraman.png')

plt.title('Double Threshholding with cameraman.png')
plt.imshow(dbThr(f, 0.05, 0.3), cmap=cm.Greys_r)
plt.show()

coin = io.imread('data/coins.png')

plt.title('Double Threshholding with coin.png')
plt.imshow(dbThr(coin, 0.1, 0.25), cmap=cm.Greys_r)
```