Student Names and IDs:

- Pablo Ortiz, 0686443
- Jongwan Park, 0848815

Homework 2

Part 1: Exam-Style Questions

Problem 1.1

Write the "full" correlation $J = I * I$ of the following 2×2 image with itself, using zero padding. Do *not* normalize.

$$I = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}$$

Answer

If

$$J = \begin{bmatrix} J(0,0) & J(0,1) & J(0,2) \\ J(1,0) & J(1,1) & J(1,2) \\ J(2,0) & J(2,1) & J(2,2) \end{bmatrix},$$

then

$$\begin{aligned} J(0,0) &= 1 \times 0 + 2 \times 0 + 0 \times 0 + 3 \times 1 = 3, \\ J(0,1) &= 1 \times 0 + 2 \times 0 + 0 \times 1 + 3 \times 2 = 6, \\ J(0,2) &= 1 \times 0 + 2 \times 0 + 0 \times 2 + 3 \times 0 = 3, \\ J(1,0) &= 1 \times 0 + 2 \times 1 + 0 \times 0 + 3 \times 0 = 2, \\ J(1,1) &= 1 \times 1 + 2 \times 2 + 0 \times 0 + 3 \times 3 = 14, \\ J(1,2) &= 1 \times 2 + 2 \times 0 + 0 \times 3 + 3 \times 0 = 2, \\ J(2,0) &= 1 \times 0 + 2 \times 0 + 0 \times 0 + 3 \times 0 = 0, \\ J(2,1) &= 1 \times 0 + 2 \times 3 + 0 \times 0 + 3 \times 0 = 6, \\ J(2,2) &= 1 \times 3 + 2 \times 0 + 0 \times 0 + 3 \times 0 = 3. \end{aligned}$$

Therefore,

$$J = I * I = \begin{bmatrix} 3 & 6 & 3 \\ 2 & 14 & 2 \\ 0 & 6 & 3 \end{bmatrix}.$$

Problem 1.2

Is the following convolution kernel separable? If so, separate it. If not, prove that it is not.

$$H = \begin{bmatrix} 2 & 0 & 6 \\ 0 & 1 & 0 \\ 1 & 0 & 3 \end{bmatrix}$$

Answer

A kernel H is separable if it can be expressed as the outer product of two vectors:

$$H = h * l,$$

where

$$h = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}, \; l = \begin{bmatrix} l_1 & l_2 & l_3 \end{bmatrix}.$$

Thus,

$$H = \begin{bmatrix} h_1 l_1 = 2 & h_1 l_2 = 0 & h_1 l_3 = 6 \\ h_2 l_1 = 0 & h_2 l_2 = 1 & h_2 l_3 = 0 \\ h_3 l_1 = 1 & h_3 l_2 = 0 & h_3 l_3 = 3 \end{bmatrix}.$$

Since

$$h_1 l_1 = 2, h_1 l_3 = 6, h_2 l_2 = 1, h_3 l_1 = 1, h_3 l_3 = 3,$$

it is true that

$$h_1, h_2, h_3, l_1, l_2, l_3 \neq 0.$$

However, that can't be true because

$$h_1 l_2 = 0, h_2 l_1 = 0, h_2 l_3 = 0, h_3 l_2 = 0.$$

Therefore, H is not separable.

Problem 1.3

Give expressions for the summation endpoints s and e in the formula for c_i given above. Your expression should be in terms of n, i, and the $\max$ and/or $\min$ operators (maximum and minimum between two numbers).

Answer

For $-n \leq i \leq 0$,

$$i + s = 0, \;\; e = n.$$

And for $0 \leq i \leq n$,

$$i + s = i, \;\; e = n - i.$$

These conditions can be met if:

$$s = \max(-i, 0), \;\; e = \min(n, n - i).$$

Problem 1.4

Show that the "full" correlation $a \star b$ as defined in the previous problem does not commute.

Answer

Say $c = a \star b$ and $d = b \star a$.

Then, with $n = 2$,

$$c_{-2} = a_0 b_2 ,$$

whereas

$$d_{-2} = b_0 a_2 .$$

Thus, $c \neq d$ and the "full" correlation does not commute.

Problem 1.5

Let

$$c = a \star b \quad \text{and} \quad d = b \star a$$

be the two "full" correlations between arbitrary, non-empty sequences a, b of equal length $n + 1$, as defined above.

Give a general expression for the relationship between c and d. Specifically, how does the generic element c_i of c relate to the generic element d_j of d?

No proof is necessary.

Answer

For c,

$$c_i = \sum_{j=-i}^{n} a_{i+j} b_j \quad \text{for} \quad -n \leq i \leq 0,$$

$$c_i = \sum_{j=0}^{n-i} a_{i+j} b_j \quad \text{for} \quad 0 \leq i \leq n.$$

Substitute $-i$ for i:

$$c_{-i} = \sum_{j=i}^{n} a_{-i+j} b_j \quad \text{for} \quad 0 \leq i \leq n,$$

$$c_{-i} = \sum_{j=0}^{n+i} a_{-i+j} b_j \quad \text{for} \quad -n \leq i \leq 0.$$

Substitute $j + i$ for j:

$$c_{-i} = \sum_{j=0}^{n-i} a_j b_{i+j} \quad \text{for} \quad 0 \leq i \leq n,$$

$$c_{-i} = \sum_{j=-i}^{n} a_j b_{i+j} \quad \text{for} \quad -n \leq i \leq 0.$$

Since d,

$$d_i = \sum_{j=-i}^{n} b_{i+j} a_j \quad \text{for} \quad -n \leq i \leq 0,$$

$$d_i = \sum_{j=0}^{n-i} b_{i+j} a_j \quad \text{for} \quad 0 \leq i \leq n,$$

$$c_{-i} = d_i$$

Part 2: Convolution

Problem 2.1

Write a function with header

```
def convolve(a, b, ctype='same'):
```

that takes two one-dimensional `numpy` arrays `a` and `b` and an optional convolution type specification `ctype` and returns the convolution of the two arrays as a `numpy` array. Assume that sequence `a` is no shorter than sequence `b`.

The possible values for `ctype` are `'full'`, `'same'` (the default), and `'valid'`. For the last two types, `a` is the signal and `b` is the kernel. For instance, `'same'` means that the output has the same size as `a`.

Show your code and the result of running the given test cases, which compare your implementation to `numpy.convolve`. It is OK if the `dtype` of your output differs from that of `numpy.convolve`.

Answer

In [1]:
```python
def convolve(a, b, ctype='same'):
    if ctype == 'same':
        if np.mod(len(b), 2) == 1:
            pada = np.append(np.zeros([1, int((len(b) - 1) / 2)]), a)
            pada = np.append(pada, np.zeros([1, int((len(b) - 1) / 2)]))
        else:
            pada = np.append(np.zeros([1, int(len(b) / 2)]), a)
            pada = np.append(pada, np.zeros([1, int(len(b) / 2)]))
            b = np.append([0], b)
        c = np.zeros((1, len(a)))
    elif ctype == 'full':
        pada = np.append(np.zeros([1, int(len(b) - 1)]), a)
        pada = np.append(pada, np.zeros([1, int(len(b) - 1)]))
        c = np.zeros((1, len(a) + len(b) - 1))
    elif ctype == 'valid':
        pada = a
        c = np.zeros((1, len(a) - len(b) + 1))
    else:
        raise ValueError('invalid convolution type')

    for i in range(len(pada) - len(b) + 1):
        for j in range(len(b)):
            c[0, i] = c[0, i] + b[-j - 1] * pada[i + j]
    return c
```

In [2]:
```python
import numpy as np

signal = np.array([2, 1, -3, 0, 5])
kernel = np.array([2, 0, -1, -2])

ctypes = ('full', 'same', 'valid')

try:
    print('convolve:')
    for ctype in ctypes:
        print(ctype, ': ', convolve(signal, kernel, ctype), sep='')
    print()

    print('numpy.convolve:')
    for ctype in ctypes:
        print(ctype, ': ', np.convolve(signal, kernel, ctype), sep='')
    print()
except NameError:
    pass
```

```
convolve:
full: [[  4.   2.  -8.  -5.  11.   6.  -5. -10.]]
same: [[ 2. -8. -5. 11.  6.]]
valid: [[-5. 11.]]

numpy.convolve:
full: [  4   2  -8  -5  11   6  -5 -10]
same: [ 2 -8 -5 11  6]
valid: [-5 11]
```

Problem 2.2

Write a function with header

```
def convolve2d(I, H):
```

that takes a two-dimensional numerical `numpy` array `I` and a two-dimensional numerical `numpy` array `H`, both non-empty, *and uses* `np.convolve` (the `numpy` one-dimensional convolution function) to implement the (two-dimensional) convolution of `I` and `H` of style `same`. Do *not* use any other convolution operators.

Show your code and the result of running the given test cases, which compare your implementation to `scipy.signal.convolve2d` when called with the `'same'` option. It is OK if the `dtype` of your output differs from that of `scipy.signal.convolve2d`.

Answer

```
In [3]: def convolve2d(I, H):
            J = np.zeros_like(I, dtype=float)
            for c, _ in enumerate(I.T):
                for r, _ in enumerate(I):
                    for u, _ in enumerate(H):
                        nu = u - int(np.ceil(H.shape[0] / 2 - 1))
                        if r - nu < I.shape[0] and r >= nu:
                            J[r, c] += np.convolve(I[r - nu, :], H[u, :], 'same')[c]
            return J
```

In [4]:
```
from scipy import signal as ss
import numpy as np

a = np.array([[10, 11, 12],
              [20, 21, 22],
              [30, 31, 32]])
hList = (np. array([[1]]), np.array([[1, -2]]), np.array([[2], [-1]]),
         np.array([[2, 3, 5], [4, 0, 1]]), np.array([[2, 3, 5], [4, 0, 1], [1, -1, 2]]))

try:
    for h in hList:
        c = convolve2d(a, h)
        s = ss.convolve2d(a, h, 'same')

        print('----')
        print('convolve2d:', c, sep='\n', end='\n\n')
        print('scipy.signal.convolve2d:', s, sep='\n', end='\n\n')
except NameError:
    pass
```

```
----
convolve2d:
[[10. 11. 12.]
 [20. 21. 22.]
 [30. 31. 32.]]

scipy.signal.convolve2d:
[[10 11 12]
 [20 21 22]
 [30 31 32]]

----
convolve2d:
[[ 10.  -9. -10.]
 [ 20. -19. -20.]
 [ 30. -29. -30.]]

scipy.signal.convolve2d:
[[ 10  -9 -10]
 [ 20 -19 -20]
 [ 30 -29 -30]]

----
convolve2d:
[[20. 22. 24.]
 [30. 31. 32.]
 [40. 41. 42.]]

scipy.signal.convolve2d:
[[20 22 24]
 [30 31 32]
 [40 41 42]]

----
convolve2d:
[[ 52. 107.  91.]
 [146. 265. 182.]
 [236. 415. 272.]]

scipy.signal.convolve2d:
[[ 52 107  91]
 [146 265 182]
 [236 415 272]]

----
convolve2d:
[[146. 265. 182.]
 [237. 436. 282.]
 [125. 199.  51.]]

scipy.signal.convolve2d:
[[146 265 182]
 [237 436 282]
 [125 199  51]]
```

Part 3: Image Differentiation

Problem 3.1

Implement two functions with headers

```
def Gaussian(sigma):
```

and

```
def dGaussian(sigma):
```

that return a one-dimensional truncated discrete Gaussian convolution kernel and its derivative (the two kernels $g(x)$ and $d(x)$ from the class notes). Both kernels take the parameter σ of the Gaussian as argument. The output of each function should be a `numpy` array (like the one that `scipy.signal.gaussian` returns) that contains the values of the function at integer points of their domain in the interval $[-a, a]$ where $a = \text{ceil}(3.5\sigma)$ (use `math.ceil`). The output kernels should be properly normalized, as explained in the class notes.

Show your code and a plot of each function with $\sigma = 2.3$. No labels are needed, but plot the Gaussian first. Use the `'.-'` line style option.

Remember to use

```
%matplotlib inline
```

as you did in homework 1.

Answer

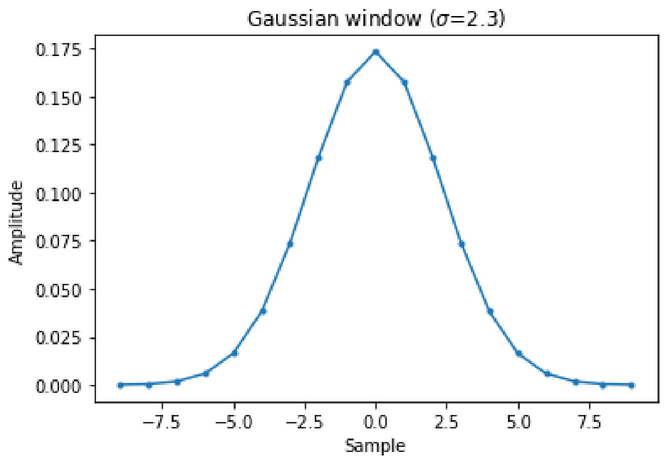
In [5]:
```
from matplotlib import pyplot as plt
from scipy import signal as ss
import math
import numpy as np
from skimage import io, color

%matplotlib inline
```

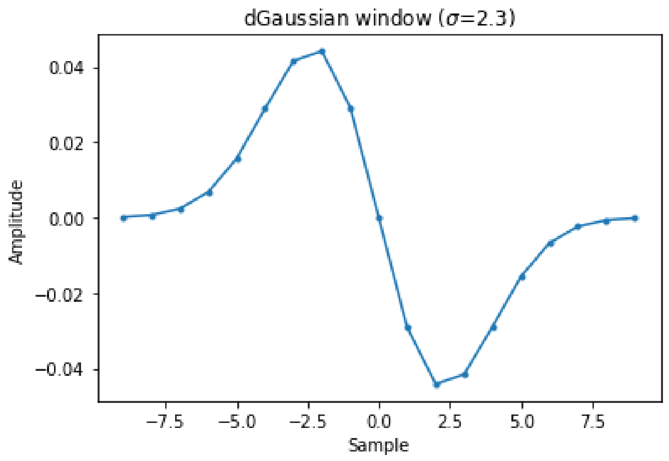
In [6]:
```
def Gaussian(sigma):
    a = math.ceil(3.5*sigma)
    gaussKernel1D = ss.gaussian(2*a+1,std=sigma)
    return gaussKernel1D/np.sum(gaussKernel1D)

def dGaussian(sigma):
    a = math.ceil(3.5*sigma)
    gaussKernel1D = ss.gaussian(2*a+2,std=sigma)
    dGaussKernel1D = np.diff(gaussKernel1D/np.sum(gaussKernel1D))
    # dKernel1D = np.array([-1, 0, 1])
    # dGaussKernel1D = np.transpose(ss.convolve2d(gaussKernel1D[np.newaxis], dKernel1D[n
p.newaxis], 'same'))
    return dGaussKernel1D

print(np.sum(dGaussian(2.3)))
print(np.sum(Gaussian(2.3)))
print(dGaussian(2.3).shape)
print(Gaussian(2.3).shape)
```

```
9.622294280808852e-18
0.9999999999999998
(19,)
(19,)
```

In [7]:
```
sigma = 2.3
a = math.ceil(3.5*sigma)
plt.plot(np.arange(-a,a+1),Gaussian(sigma), '.-')
plt.title("Gaussian window ($\sigma$=2.3)")
plt.ylabel("Amplitude")
plt.xlabel("Sample")


plt.figure()
plt.plot(np.arange(-a,a+1),dGaussian(sigma), '.-')
plt.title("dGaussian window ($\sigma$=2.3)")
plt.ylabel("Amplitude")
plt.xlabel("Sample")
```

Out[7]:
```
Text(0.5, 0, 'Sample')
```





Problem 3.2

Write a function with header

```
def gradient(img, sigma=2.5):
```

that computes the gradient of the black-and-white image `img` (a two-dimensional `numpy` array). Your implementation should use the function `scipy.signal.convolve2d` and the kernel functions you wrote for the previous problem. The result should be a pair (that is, a Python tuple of length 2) of `numpy` arrays, each of the same size as `img`. The first array in the pair should be the derivative in the horizontal direction.

The positive coordinate axes are horizontal to the right and vertical down. Make sure you get the signs of the derivatives right, given this convention.

The function `scipy.signal.convolve2d` takes a 2-dimensional kernel, but you *must* exploit separability of the differentiation kernels for full credit, so you will be calling this function with kernels that are either $k \times 1$ or $1 \times k$ for some k.

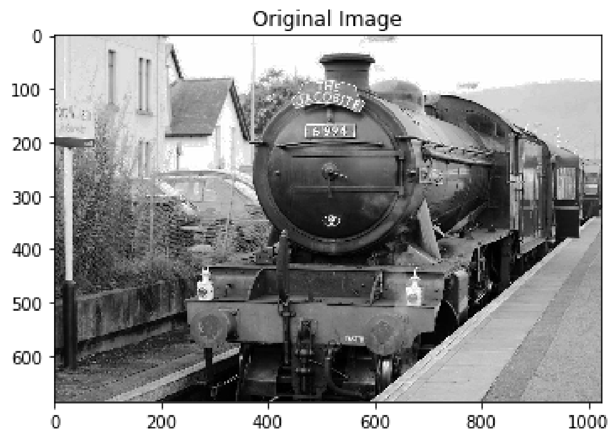
Show your code, and display the `locomotive.jpg` image (provided) and the two gradient images for it. Use the default value of `sigma`, and use `matplotlib.pyplot.imshow` for display.

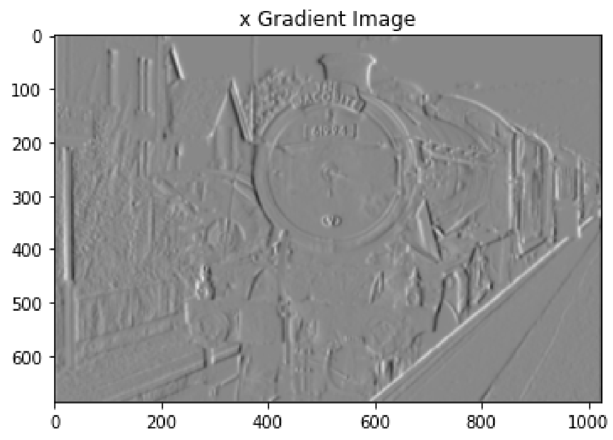
Answer

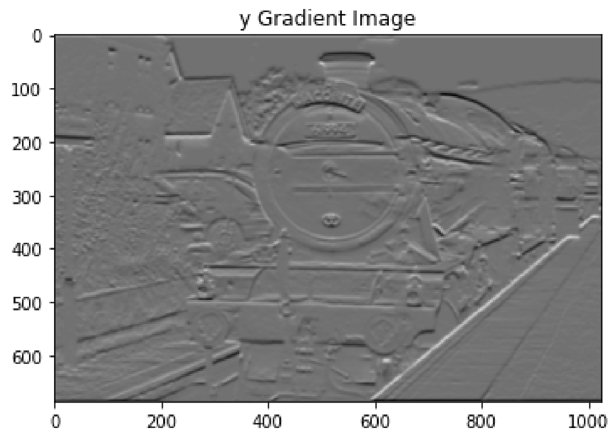
In [8]:
```
def gradient(img, sigma=2.5):
    a = math.ceil(3.5*sigma)
    xGaussKernel1D = Gaussian(sigma)[np.newaxis]
    yGaussKernel1D = np.transpose(xGaussKernel1D)
    dxGaussKernel1D = dGaussian(sigma)[np.newaxis]
    dyGaussKernel1D = np.transpose(dxGaussKernel1D)
    xGradientImg = ss.convolve2d(ss.convolve2d(img, yGaussKernel1D, 'same'), dxGaussKerne
l1D, 'same')
    yGradientImg = ss.convolve2d(ss.convolve2d(img, xGaussKernel1D, 'same'), dyGaussKerne
l1D, 'same')

    return [xGradientImg, yGradientImg]

# xGaussKernel1D = Gaussian(2.5)[np.newaxis]
# yGaussKernel1D = np.transpose(xGaussKernel1D)
# dxGaussKernel1D = dGaussian(2.5)[np.newaxis]
# dyGaussKernel1D = np.transpose(dxGaussKernel1D)
# print(xGaussKernel1D.shape)
# print(yGaussKernel1D.shape)
# print(dxGaussKernel1D.shape)
# print(dyGaussKernel1D.shape)
# print(ss.convolve2d(img, dxGaussKernel1D,'same').shape)
```

In [9]:
```
img = color.rgb2gray(io.imread('locomotive.jpg'))

plt.figure()
plt.imshow(img, cmap = 'gray')
plt.title("Original Image")

[xGradientImg, yGradientImg] = gradient(img, sigma=2.5)

plt.figure()
plt.imshow(xGradientImg, cmap = 'gray')
plt.title("x Gradient Image")

plt.figure()
plt.imshow(yGradientImg, cmap = 'gray')
plt.title("y Gradient Image")
```

Out[9]:
```
Text(0.5, 1.0, 'y Gradient Image')
```







Problem 3.3

Compute and display the magnitude of the gradient for `locomotive.jpg` . Show your code and the output.

Answer

```
In [10]: plt.figure()
         plt.imshow(np.sqrt(np.square(xGradientImg) + np.square(yGradientImg)), cmap = 'gray')
         plt.title("Magnitude of Gradient Image")
```

```
Out[10]: Text(0.5, 1.0, 'Magnitude of Gradient Image')
```