Proposed solutions to the exercises in the first exercise set in Python

Here you can find the translated exercise set from Matlab to Python, along with some suggested solutions. Note that the functions availible in Python depends on which packages you want to use. This means that functions like imfilter, filter2 and conv2 as described in the set for Matlab does not necesarily hold for the package you have chosen to use in Python. However, what is important regarding Python , is to know which packages have the desired implementations. There might be implementations in some packages written in a more desirable manner, and knwoing which to choose from can save one a lot of time. In the suggestions, you might see packages like scipy, scikit-image and openCV being used.

Translation of the exercise set from Matlab to Python

In Matlab, there exists some test images by default, e. g. football.jpg and coins.png. Unfortunately, these test images does not exist in the OpenCV package. To be able to load images using the proposed translatation, two images with the name image1.jpg and image2.jpg must be in the same folder as the downloaded program.

```
import numpy as np # For numeric computing, setting up matrices and performing computati
import matplotlib.pyplot as plt # For plotting

# Originally written by Ole-MariusHoel Rindal for image analysis in 2015.
# Translated into Python by Kristine B. Hein for the same course held in 2018.


# The matrix | 1 0 0 |
#            | 0 1 0 |
#            | 0 0 1 |
# may be created like this (note that the backslash is not necessary,
# but is used such that a newline is allowed in the script for readability):
someMatrix = np.array([[1,0,0],\
                       [0,1,0],\
                       [0,0,1]])
# Actually, a matrix in Python is represented as a list of lists.
# To be able to perform matrix operatons from Numpy between matrices, lists that are
# converted into numpy arrays must be used.

# e.g. [ [row1] , [row2] , ... , [rowM] ], where a row can be defined as
# [number1, number2, ..., numberN]

# Numpy comes built-in with function for many common matrix operations,
# for the case above we could have typed:
someMatrix = np.diag([1,1,1])

# If you wonder how a function/operator works in Numpy, the documentation can be
# found here (along with documentation for scipy):
# https://docs.scipy.org/doc/

# To constract a matrix, memory must be allocated. It is not possible to
# dynamic allocate memory to a matrix in Python.

# E.g. allocate memory for 100 x 100 matrix and fill it with ones.
# Note that most of the dimensions of the matrices in Numpy
# are defined as tuples/lists: (number-of-rows, number-of-columns)
big_1 = np.ones((100,100))

# Or, allocate a 50 x 40 x 3 matrix and fill it with zeros:
big_0 = np.zeros((50,40,3))

# An 1D array can be created using the '<from>:<to>:<step>' syntax using np.arange.
# Note that the end-value of the array will end at the integer closest to the value <to>
# but never be equal to <to> or higher than <to>.
steg1 = np.arange(0,50,2)

# The <step> is optional, and so is the <from>.
# If omitted the step defaults to 1 and the from value defaults to zero.
steg2 = np.arange(50)

# The standard plotting function is PLOT, e.g.:
plt.plot(steg1)
plt.title('Demonstration of plot()');
# To show the plot, remeber to use plt.show()!
#plt.show()

# Extracting a value, or a series of values, from a matrix is easily
# achieved like this:
mat = np.array([[1,7,4] , [3,4,5]])
print(mat[1,0]) # retrives the '3'. NOTE: The first index is 0 in Python!
```

```
print(mat[1][0]) # An altenative syntax to retrieve the same matrix element as above.

# A range of elements may retrieved using the <from>:<to> syntax, i.e slicing:
print(mat[0:2,0])     # First column
print(mat[0:-1,0])   # The same column where -1 means the last element
print(mat[:,0])       # The same, : is here 'all rows'

# We can use the same syntax to set a range of elements:
mat[0:2,0:2] = 0

# Note that a matrix can be stored in various formats, e.g. UINT8, INT8 or
# DOUBLE. They all have their conversion functions, see e.g. 'help uint8'.
# If your result looks fishy, and you have a hard time figuring out why,
# a type conversion (of lack of one) may be the reason!

## 2. Matrix operations
# Here we'll demonstrate how some simple matrix operations is done in
# MATLAB

# We'll start
mat1 = np.array([[2,3], [5,7]]);
mat2 = np.array([[4,5], [1,9]]);

# Elementwise addition and subtraction.
print(mat1 + mat2)
print(mat1 - mat2)

# Transpose.
print(np.transpose(mat1))

# Complex conjugate transpose (same as transpose when the matrix is real).
print(np.conjugate(mat1))

# But notice the difference for a complex matrix, note at 1j
# is the imaginary unit, i.e 1j = sqrt(-1).
complexMatrix = np.array([[0,1j-2], [1j+2,0]])

# Transpose for complex, see the function transpose
print(np.transpose(complexMatrix))

# Complex conjugate transpose, see the function ctranspose
print(np.conjugate(complexMatrix))

# Multiplication, division and power.
print(mat1.dot(mat2)) # The * denotes elementwise product, the Hadamard product
print(np.divide(mat1,mat2))
print(np.linalg.matrix_power(mat1,2))

# Elementwise multiplication, division and power
print(mat1 * mat2)
print(mat1 / mat2)
print(np.power(mat1,mat2))

## 3. Control structures

# While and for loops.
# Can often be used instead of matrix operations, but tend to lead to a
# slightly inferior perfomance.
#
# The reason for introducing performance-tips already is because we will be
# storing images as matrices, and these tend to be large. If we also
```

```
# consider that the algorithms we use are computationally intensive,
# writing somewhat optimised code is often worth the extra effort.


# E.g. can this simple loop:
A = np.zeros(100)
for i in range(100):
    A[i] = 2

# be replaced by a single and more efficient command:
A = np.ones(100) * 2

stor_1 = np.zeros((100,100))
# And:
for i in range(100):
    for j in range(100):
        stor_1[i,j] = i^2

# could be constructed by:
big_1_temp = np.tile((np.arange(100)**2).reshape(-1,1), (1,100))
# Note that .reshape(-1,1) makes the 1D array from arange into a column vector.
# The -1 is a flag to tell that 'any number' of rows is okay, as long as the resulting v
# The final vector has the same number of elements than before the rehaping.


# Note:
# If you are not used to 'doing everything with vectors', you'll likely
# want to use loops far more often than you need to. When you get the hang
# of it, typing vector-code is faster and often less error-prone than the
# loop-equivalents, but loops offer better flexibility.

# As always, add lots of COMMENTS, especially for complex one-liners!

# Logical structures: if-elseif-else. Which number is displayed?
if -1:
    print(1)
elif -2:
    print(2)
else:
    print(3)

# A very handy method, is to use boolean matrices.

# Example: Find elements which are equal to 4.
M = np.array([[1,2,3],\
              [4,4,5],\
              [0,0,2]])
print(M)

M_bool = M == 4            # a boolean matrix
print(M_bool)

M[M == 4] = 3          # change them to 3
print(M)

M[M_bool] = 4               # and back to 4
print(M)

# 5. Images
#Now, we'll finally look at some images, as most of our programs will have to deal with
```

```
# Many packages for image analysis, computer vision etc. exists.
# Some of the packages that could be used, are openCV for Python or scikit-image.
# Especially openCV has a alot of functions used in image analysis, with many of the fun
# that will be discussed throughout this course implemented.
import cv2 # Import openCV.

# Open a couple of images.
img1 = cv2.imread('image1.jpg')
img2 = cv2.imread('image2.jpg')

# If we choose to work with images having color, i.e having three channels,
# have in mind that the images read are in BGR, and not RGB!
# Plotting with matplotlib will therefore give images having different colors than usual
# To overcome this, the image can be converted into RGB before plotting.
# If any processing of the image on each of the channels are performed, especially
# by using openCV's implementation, be careful to do the conversion after the processing

img1_rgb = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2_rgb = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)

# Draw the first image (plt.show is used at the bottom to avoid hindrering other windows
plt.figure()
plt.imshow(img1)
plt.title('The first image in BGR')

plt.imshow(img1_rgb)
plt.title('The first image in RGB')

# Display the second image in a new figure.
plt.figure()
plt.imshow(img2_rgb)
plt.title('The second image in RGB')

# Histogram of the image channel 0, that being the layer of the image
# that described how much blue it is in the image
H = cv2.calcHist([img2],[0],None,[256],[0,256])

plt.figure()
plt.plot(np.arange(256),H.ravel())
plt.title('The histogram for the second image at the blue channel');

# Converting a colour image with 3 colour channels to a greyscale image
img3 = cv2.imread('image1.jpg', 0).astype('float') # 0 is used to indicate a gray-level
plt.figure()
plt.imshow(img3,cmap='gray')
plt.title('Grayscale image of image 1');

## 6. Some basic filtering in the image domain.
# Now we have jumped into the curriculum of INF2310.
# Don't hesitate to ask or read up on last courses materials
# available at http://www.uio.no/studier/emner/matnat/ifi/INF2310/v17/
# under 'Undervisningplan'

# 2D correlation: FILTER2
# 2D convolution: CONV2
# With IMFILTER you can choose either 2D correlation (default) and 2D
# convolution, and it also has some boundary options.

# Filter IMG3 using a 5x5 mean filter.
h1 = np.ones((5,5)) / 25
```

```
img4 = cv2.filter2D(img3,-1,h1) # symmetric filter => correlation = convolution

plt.figure()
plt.imshow(img3,cmap='gray')
plt.title('Original image');

plt.figure()
plt.imshow(img4,cmap='gray')
plt.title('Filtered image')

# Find the gradient magnitude of IMG3.
h2x = np.array([[-1,-2,-1],\
                [0,0,0],\
                [1,2,1]])

h2y = np.array([[-1,0,1],\
                [-2,0,2],\
                [-1,0,1]])

resX = cv2.filter2D(img3,-1, h2x)
resY = cv2.filter2D(img3,-1, h2y)
resXY = np.sqrt(resX**2 + resY**2)

# Display the gradient magnitude:
plt.figure()
plt.imshow(resXY,cmap='gray')
plt.title('Gradient magnitude')

# Show the images
plt.show()


#################
## EXERCISE 1
#################

# Above, we loaded the image 'image1.jpg', converted it to a greyscale
# image and applied a 5x5 mean filter, by using the commands:

img3 = cv2.imread('image1.jpg', 0).astype('float')
h1 = np.ones((5,5)) / 25;
img4 = cv2.filter2D(img3,-1,h1)

plt.figure()
plt.imshow(img3,cmap='gray')
plt.title('Original image');

plt.figure()
plt.imshow(img4,cmap='gray')
plt.title('Filtered image')

# The filtered image have a two pixel wide black frame.
# a) Use the indexing techniques described above to remove these.
# b) Use scipy.signal.convolve2d with the option mode='valid' to remove the borders.

#################
## EXERCISE 2
#################

# Make a function that returns the same as np.histogram
#
```

```
# Although it is allowed to use loops, try to avoid using them where
# it is possible. One loop should suffice.


################
## EXERCISE 3
################

# Above, we loaded the image 'image2.jpg' using the command:
img2 = cv2.imread('image2.jpg', 0).astype('float')

# a)
# Use the operators >, <, >=, <= to threshold IMG2 using an arbitrary
# threshold.
#
# b)
# Use an implemented method to compute the threshold using Otsu's method,
# e. g. from scipy or openCV
#
# c)
# Compare the binary image resulting from part a with the one from part b
# by displaying the images. Do you notice any differences?
# Display also the difference between the images.

################
## EXERCISE 4
################

# Normalize resXY such that np.max(resXY) = 255 and np.min(resXY) = 0.
# Threshold the result with T = 100.
#
# What would you do if you wanted to obtain an image containing
# only the seam, and the entire seam, of the the ball?
```

The program is available here (https://github.com/krisbhei/IN5520-IN9520/blob/master/1/exerciseset_python.py) (right click and press "save link as").

Suggested solution

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

### Exercise 1

img3 = cv2.imread('image1.jpg', 0).astype('float')
h1 = np.ones((5,5)) / 25;
img4 = cv2.filter2D(img3,-1,h1)

# a)
img4_indexed = img4[2:-2,2:-2]

plt.figure()
plt.imshow(img4_indexed,cmap='gray')
plt.title('Filtered img3 without borders, indexed')

# b)

from scipy.signal import convolve2d

img4_conv2 = convolve2d(img3.astype('float'),h1,mode='valid')

plt.figure()
plt.imshow(img4_conv2,cmap='gray')
plt.title('Filtered img3 without borders, convolve2d')

#plt.show()

# Just to check if a) and b) gives the same answer:
print("Max absolute error between index and filtered using valid: %g"\
      %np.max(np.abs(img4_conv2 - img4_indexed)))

### Exercise 2

hist_np,bins = np.histogram(img3.ravel(), bins=256,range=(0,256))

def iimhist(img):
    h = np.zeros(256)
    for i in range(256):
        h[i] = np.sum(img == i)
        # img == i makes a boolean matrix. By summing over the matrix,
        # all entries being False evaluates to 0, and all entries being True
        # to 1.
    return h

hist_ = iimhist(img3)

# Too check is the implementation is correct compared to np.histogram
print("Max absolute difference between np.histogram and iimhist: %g"%np.max(np.abs(hist_

plt.figure()
plt.bar(bins[:-1], hist_np) #take the whole bin array except for the last element,
                            #this is equal to 256.
plt.bar(bins[:-1], hist_)
plt.title('Histogram using numpy')
plt.legend(['numpy', 'iimhist'])

#plt.show()
```

```
### Exercise 3

# a)
img2 = cv2.imread('image2.jpg', 0)

img2_size = img2.shape

thresholded1 = np.zeros(img2_size) # Make a matrix of the same size as img2
thresholded1[img2 > 100] = 1

thresholded2 = np.zeros(img2_size)
thresholded2[img2 < 100] = 1

thresholded3 = np.zeros(img2_size)
thresholded3[img2 >= 120] = 1

thresholded4 = np.zeros(img2_size)
thresholded4[img2 <= 120] = 1

# Make a figure with several plots iwhtin a window
plt.figure()

plt.subplot(2,2,1)
plt.imshow(thresholded1,cmap='gray')
plt.title("Pixel values from img2 greater than 100")

plt.subplot(2,2,2)
plt.imshow(thresholded2,cmap='gray')
plt.title("Pixel values from img2 less than 100")

plt.subplot(2,2,3)
plt.imshow(thresholded3,cmap='gray')
plt.title("Pixel values from img2 greater or equal to 120")

plt.subplot(2,2,4)
plt.imshow(thresholded4,cmap='gray')
plt.title("Pixel values from img2 less or equal to 120")

#plt.show()

# b)
thr_otsu,img2_otsu = cv2.threshold(img2,0,1,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

plt.figure()
plt.imshow(img2_otsu,cmap='gray')
plt.title('img2 thresholded with Otsu`s method')

#plt.show()

# c)
plt.figure()

plt.subplot(2,2,1)
plt.imshow(np.abs(thresholded1-img2_otsu),cmap='gray')
plt.title("Difference between thresholded1 and \n thresholded image using Otsu`s method"

plt.subplot(2,2,2)
plt.imshow(np.abs(thresholded2-img2_otsu),cmap='gray')
plt.title("Difference between thresholded2 and \n thresholded image using Otsu`s method"
```

```
plt.subplot(2,2,3)
plt.imshow(np.abs(thresholded3-img2_otsu),cmap='gray')
plt.title("Difference between thresholded3 and \n thresholded image using Otsu`s method"

plt.subplot(2,2,4)
plt.imshow(np.abs(thresholded4-img2_otsu),cmap='gray')
plt.title("Difference between thresholded4 and \n thresholded image using Otsu`s method"

#plt.show()


### Exercise 4

h2x = np.array([[-1,-2,-1],\
                [0,0,0],\
                [1,2,1]])

h2y = np.array([[-1,0,1],\
                [-2,0,2],\
                [-1,0,1]])

resX = cv2.filter2D(img3,-1, h2x)
resY = cv2.filter2D(img3,-1, h2y)
resXY = np.sqrt(resX**2 + resY**2)

resXY_norm = cv2.normalize(resXY, None, 0, 255, cv2.NORM_MINMAX)

print("max(resXY_norm) = %g"%np.max(resXY_norm))
print("min(resXY_norm) = %g"%np.min(resXY_norm))

# One could also do the normalization 'manually'.
# The normalization could be seen as a linear transform of the pixel
# intensities: normalized_img = a*img + b
# To determine the coefficients a and b, the following equations can
# be solved:
#
# 0 = a*min(img) + b
# 255 = a*max(img) + b
#
# which yields
# a = 255/( max(img) - min(img) )
# b = -a*min(img)
#   = -( 255 * min(img) )/( max(img) - min(img) )
#
# and gives the following expression for the normalization
#
# normalized_img = 255/( max(img) - min(img) )*( img -  min(img) )

resXY_norm2 = 255/( np.max(resXY) - np.min(resXY) )*( resXY -  np.min(resXY) )

print("Max absolute difference between the normalizations: %g"%np.max(np.abs(resXY_norm
```

The program is available here (https://github.com/krisbhei/IN5520-IN9520/blob/master/1/proposed_solutions.py) (right click and press "save link as").

Made with DocOnce (https://github.com/hplgit/doconce)