

Homework Assignment 0

This assignment is a teaser/refresher on calculus, linear algebra, and includes exercises on lenses, image gradients, point processing, and harris corners. It also introduces you to jupyter notebook environment and python. Notebook environment allows you to combine cells with python code and cells with text (markdown cells). Text cells can include "latex" mathematical formulas. Such formulas can be written in the inline mode, for example, $(x + y)^2 = x^2 + 2xy + y^2$. Important or longer formulas may look better in a show mode, e.g.

$$1 = \sum_{n=1}^{\infty} \left(\frac{1}{2}\right)^n.$$

Latex is commonly used for scientific writing and you should use it for the written parts of your assignments. You should use text cells (markdown cells) to answer written questions or to present your explanations/comments in notebook reports with code. A list of common mathematical symbols in latex can be easily found online (e.g.

https://oeis.org/wiki/List_of_LaTeX_mathematical_symbols). You can also find many online resources explaining latex for mathematical equations, e.g.

https://en.wikibooks.org/wiki/LaTeX/Advanced_Mathematics.

In [1]:

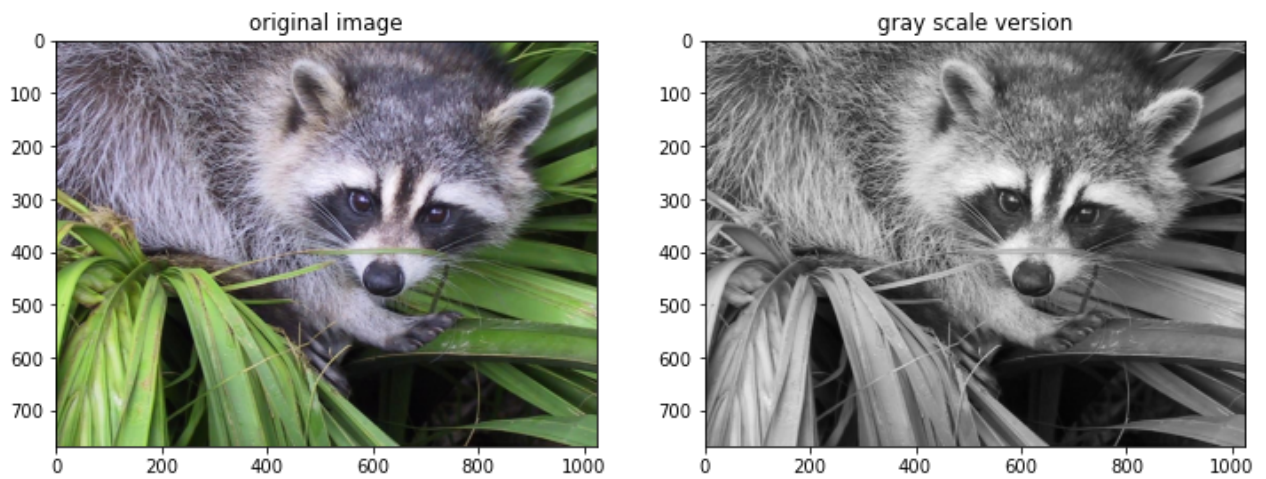
```
# This cell loads some libraries and a test image you can use. Feel free to load your own
# but you must save them in "images" subdirectory before creating .zip for your submission

%matplotlib inline
# NOTE: all "magic" options for backend plotting are: inline, notebook
# see http://ipython.readthedocs.io/en/stable/interactive/plotting.html

import numpy as np
import matplotlib
import matplotlib.image as image
import matplotlib.pyplot as plt
from scipy import misc
from skimage.color import rgb2gray

im = misc.face() # a sample image in misc library
#im=image.imread("../images/IMG_3306.jpg") # another image (loaded from your file), uncomment

plt.figure(1, figsize = (12, 8))
plt.subplot(121)
plt.imshow(im)
plt.title("original image")
plt.subplot(122)
plt.imshow(rgb2gray(im), cmap="gray")
plt.title("gray scale version")
plt.show()
```



Problem 1

Use the following three cells to write your own python functions that take an arbitrary RGB image and outputs its greyscale version. The functions' input should be an RGB image. The computed greyscale image should be a 2D array of the same size as the input image. You should write your own code for converting colored images to greyscale images without using any standard functions like `rgb2gray` from "skimage" in the cell above, or any other image library for python. Treat greyscale value as an *average* of the corresponding R G and B values. You should write three versions A, B, and C, as detailed in each cell below.

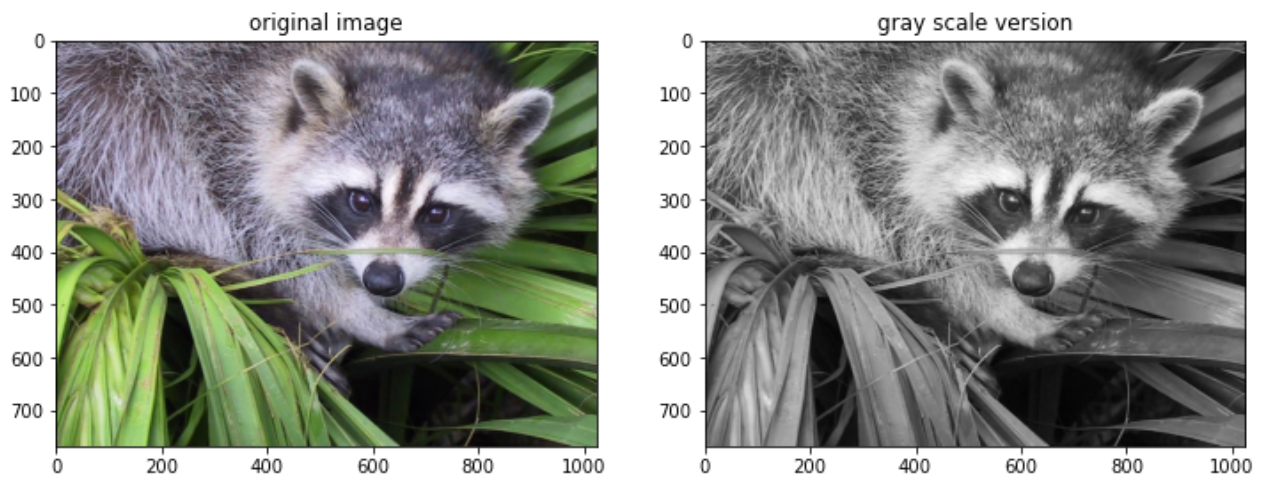
```
In [2]: # Solution A: (for-loops)
# In this version you should explicitly use two nested for-loops traversing individual pixels
# of the input image, computing the average of R, G, and B values for each pixel, and copying
# to the corresponding element of the output matrix (gray-scale image).
def toGrayScale_A(color_image):
    w = color_image.shape[0]
    h = color_image.shape[1]
    newImg = np.zeros([w, h, 3], dtype=np. uint8)
    for x in range(w):
        for y in range(h):
            r = int(color_image[x][y][0])
            g = int(color_image[x][y][1])
            b = int(color_image[x][y][2])
            avg = int((r + g + b) / 3)
            newImg[x][y][0] = avg
            newImg[x][y][1] = avg
            newImg[x][y][2] = avg
            # print(avg)
    return newImg
```

```
In [3]: # Solution B: (basic numpy operators for matrix operations)
# In the next two versions you can't use for-loops (or other loops) explicitly traversing
# In B below you should first separate image colors into individual 2D arrays (matrices)
# using "slicing" or "reshaping" (e.g. see Filtering.ipynb in Code/Samples - course web page)
# and then compute the average of these matrices  $0.3333 \cdot (A+B+C)$  directly using numpy operators
# for adding and scaling matrices. HINT: your code can look like linear algebraic expressions
def toGrayScale_B(color_image):
    R = color_image[:, :, 0]
    G = color_image[:, :, 1]
    B = color_image[:, :, 2]
```

```
NewImg = 0.3333*(R+G+B)
return NewImg
```

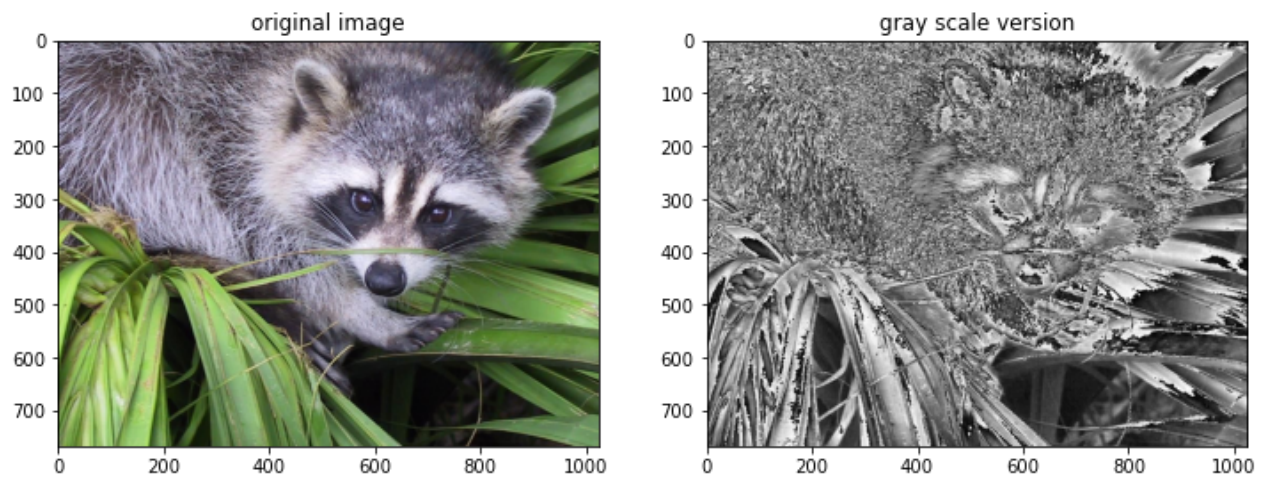
```
In [4]: # Solution C: (vectorized functions)
# In this version you should use numpy function 'dot' applying it
# directly to colored image (3d array) and vector [0.33,0.33,0.33] defining weights
# for each color component.
def toGrayScale_C(color_image):
    newImg = np.dot(color_image, [0.33,0.33,0.33])
    return newImg
```

```
In [5]: %%time
# Test your code for version A in this cell.
plt.figure(2,figsize = (12, 8))
plt.subplot(121)
plt.imshow(im)
plt.title("original image")
plt.subplot(122)
plt.imshow(toGrayScale_A(im), cmap="gray")
plt.title("gray scale version")
plt.show()
```



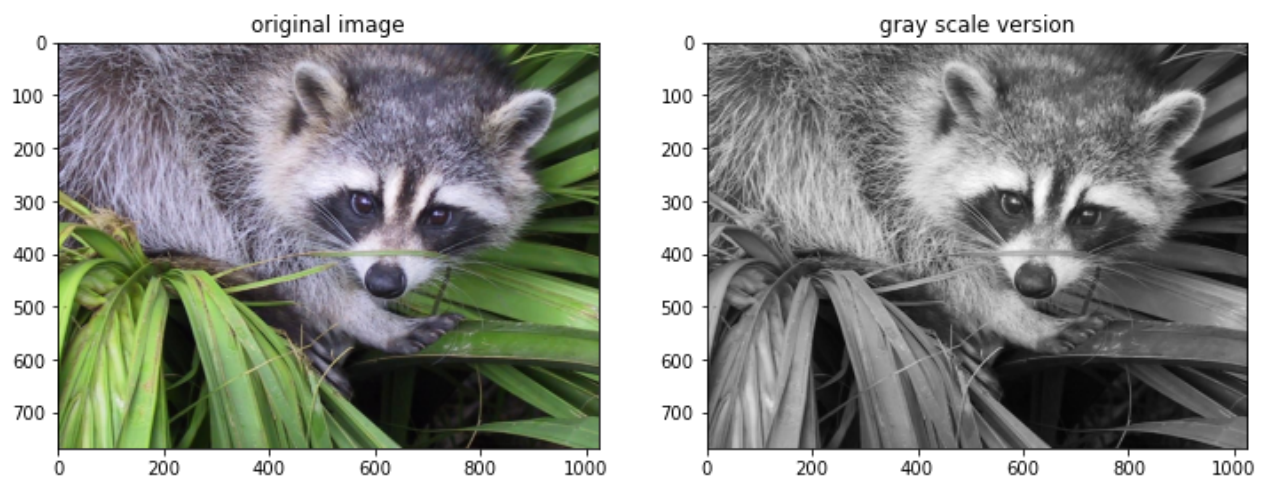
Wall time: 2.31 s

```
In [6]: %%time
# Test your code for version B in this cell.
plt.figure(3,figsize = (12, 8))
plt.subplot(121)
plt.imshow(im)
plt.title("original image")
plt.subplot(122)
plt.imshow(toGrayScale_B(im), cmap="gray")
plt.title("gray scale version")
plt.show()
```



Wall time: 362 ms

```
In [7]: %%time
plt.figure(4,figsize = (12, 8))
plt.subplot(121)
plt.imshow(im)
plt.title("original image")
plt.subplot(122)
plt.imshow(toGrayScale_C(im), cmap="gray")
plt.title("gray scale version")
plt.show()
```



Wall time: 385 ms

ATTENTION: problem 1 should teach you NEVER to use for-loops (or other explicit loops) when coding operations over image pixels or, more generally, matrix elements! Compare the running times reported in the last three cells and decide for yourself what would happen if the image was much larger and you had to run your code on many images. **Later in this course, your marks will be significantly reduced if your code explicitly traverses matrix elements. Numpy was designed to avoid that type of code.** You should always

use basic 'numpy' operators for matrices that make your code both efficient and simple. In many cases, numpy's code should look exactly like linear algebraic equations. When basic linear algebraic operators are not enough, you should look for appropriate "vectorized" functions (e.g. like 'dot'). We will often provide hints on specific vectorized functions that can be helpful. Learning how to use vectorized functions is significant for proper numpy coding. You should also get comfortable with "slicing" and "reshaping" operations as soon as possible - they are ubiquitous in numpy and critical for effective code.

Problem 2

Write code that randomly shuffles all image pixels using function `numpy.random.shuffle`. Show some test case (original image and result). Display the histograms of both the original and the "shuffled" images.

```
In [8]: # Solution: write your code in this cell. Show one image and a result of shuffling.

plt.figure(1,figsize = (12, 8))
plt.subplot(121)
plt.imshow(im)
plt.title("original image")
R = np.reshape(im[:, :, 0], im.shape[0]*im.shape[1]) # generating a sequence of intensitie
G = np.reshape(im[:, :, 1], im.shape[0]*im.shape[1]) # generating a sequence of intensitie
B = np.reshape(im[:, :, 2], im.shape[0]*im.shape[1]) # generating a sequence of intensitie
rawpixels = np.reshape(im, (im.shape[0]*im.shape[1], im.shape[2]))
pixels = rawpixels.copy()

pixels.setflags(write = 1)
np.random.shuffle(pixels)
newImg = np.reshape(pixels, im.shape)

plt.figure(3,figsize = (12, 3))
plt.subplot(131)
plt.hist(R, 50, density=True, facecolor='r') # matplotlib version (plot)
plt.title("R")
plt.subplot(132)
plt.hist(G, bins=50, density=True, facecolor='g') # matplotlib version (plot)
plt.title("G")
plt.subplot(133)
plt.hist(B, bins=50, density=True, facecolor='b') # matplotlib version (plot)
plt.title("B")
plt.show()
plt.figure(1,figsize = (12, 8))
plt.subplot(121)
plt.imshow(newImg)
plt.title("shuffled image")
R2 = np.reshape(newImg[:, :, 0], newImg.shape[0]*newImg.shape[1]) # generating a sequence
G2 = np.reshape(newImg[:, :, 1], newImg.shape[0]*newImg.shape[1]) # generating a sequence
```

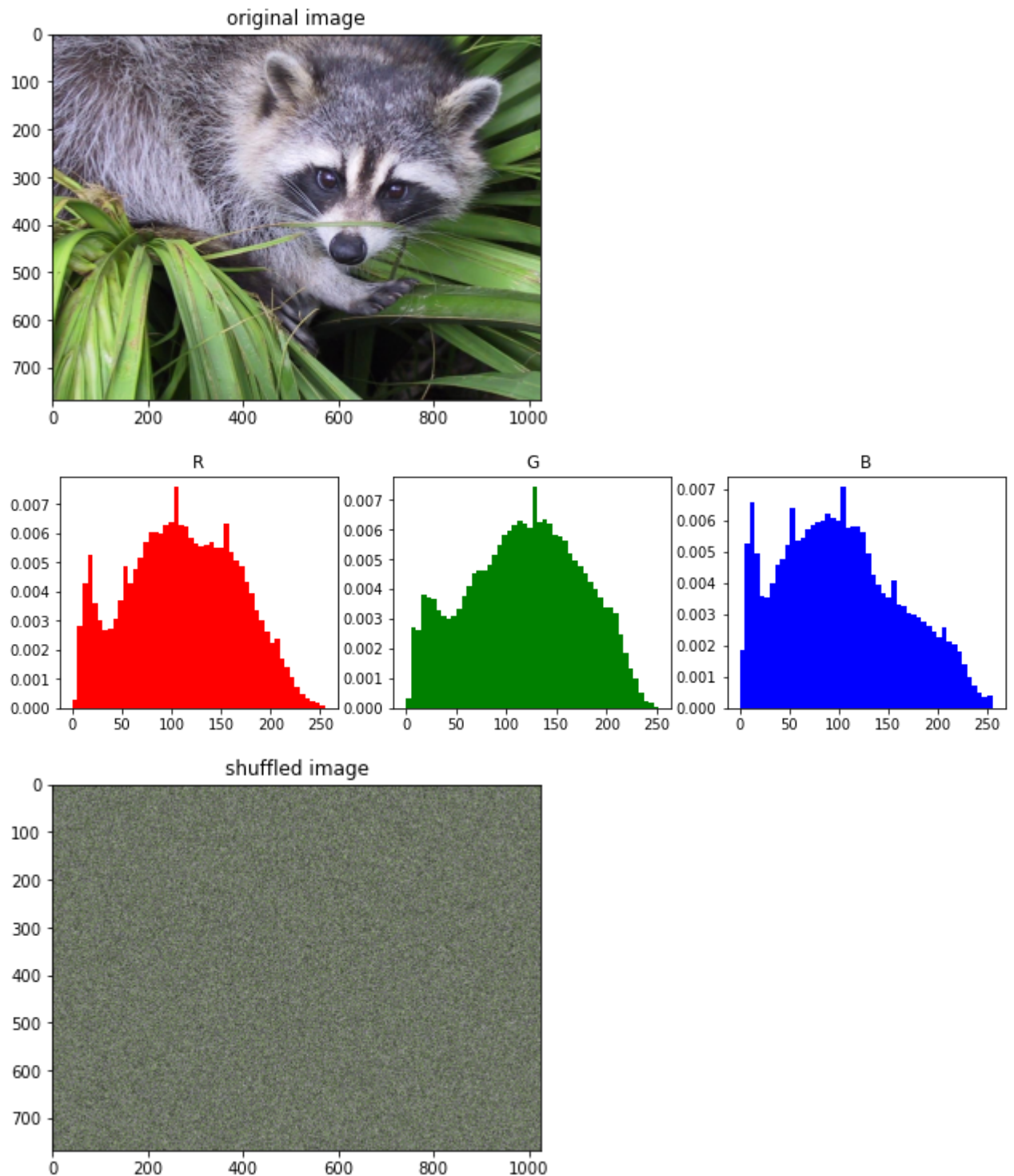


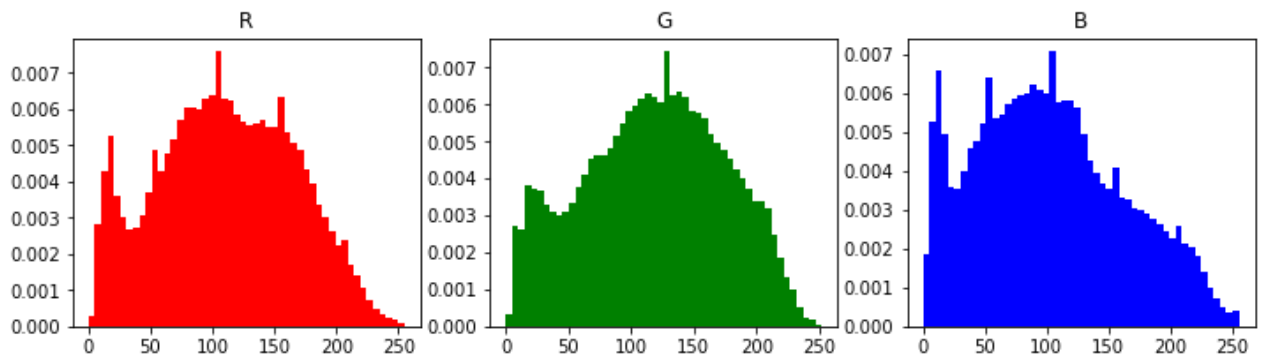
```

B2 = np.reshape(newImg[:, :, 2], newImg.shape[0]*newImg.shape[1]) # generating a sequence
plt.figure(3, figsize = (12, 3))
plt.subplot(131)
plt.hist(R2, 50, density=True, facecolor='r') # matplotlib version (plot)
plt.title("R")
plt.subplot(132)
plt.hist(G2, bins=50, density=True, facecolor='g') # matplotlib version (plot)
plt.title("G")
plt.subplot(133)
plt.hist(B2, bins=50, density=True, facecolor='b') # matplotlib version (plot)
plt.title("B")

plt.show()

```





Problem 3

Define domain transformation functions $t_x(x, y)$ and $t_y(x, y)$ that reflects an image around any given line $y = ax + b$. You can use line parameters a, b as parameters of the functions. t_x and t_y . NOTE: this is an abstract mathematical (geometric) question - you do not need to worry about real image boundaries, discreteness of the pixel grid, or anything like that, just assume that pixel (x, y) is an arbitrary real-valued points in R^2 and the output of functions t_x and t_y can map it on any other real-valued point in R^2 .

Solution: (show your work in this "Markdown" cell using basic text and latex for formulas, make sure you "run" this and other text or code cells before saving your notebook for submission so that it is easy to read and evaluate your writings and results). You can add empty lines to create new paragraphs in text (markdown) cells, as here...

My solution is

$$t_x(x, y) = x - 2a \frac{ax - y + b}{a^2 + 1} \quad (1)$$

$$t_y(x, y) = y + 2 \frac{ax - y + b}{a^2 + 1} \quad (2)$$

Use only plain text (no boldface or ### heading) in your solutions so that it is easier to distinguish your work from the provided problem statements. However, if necessary, you can insert additional cells, if that helps the structure your solution.

Do not change the order of the problems. Once you completed all written and code cells, run

Kernel->Restart & Run All

to generate a final "gradable" version of your notebook and save your ipynb file. Also use

File->Print Preview


and then print your report from your browser into a pdf file. Submit both .pdf and .ipynb files.

Problem 4

As stated in the lectures (topic 2), assuming fixed "image distance" a lens generates perfectly sharp image only for 3D points at some particular depth. Assuming an object has sharp image when image distance is h and that the focal length of the lens f is known, what is the object's depth

$$d(h) = ?$$

Your solution should show your derivation.

HINT: Use the illustration below to find similar triangles and to identify where the focal length f of the lens is relevant. 

Solution: (use text and latex formulas to justify/explain. Feel free to replace an image above with a modified version including your scribbles.)

Let the ray which is horizontal before reach the lens intersects horizontal axis at point P, intersects with lens at point A, intersects the image plane at point B. And the source of it at object point D.

Let the object intersects horizontal axis at point O, and image plane intersects horizontal axis at point I.

Note that the distance between C and P is the focal length f

We have

$$\frac{d}{AC} = \frac{h}{IB} \quad (3)$$

$$\frac{f}{AC} = \frac{h-f}{IB} \quad (4)$$

Therefore, we have

$$d = h \frac{AC}{IB} \quad (5)$$

$$\frac{AC}{IB} = \frac{f}{h-f} \quad (6)$$

$$d(h) = \frac{fh}{h-f} \quad (7)$$

Problem 5

(a) Find all points $x \in \mathbb{R}^1$ corresponding to local minima of function $f(x) = -5x^3 + 2x^2 - x$. Show your derivation.

Solution: (write your solution in this cell)

$$f'(x) = -15x^2 + 4x - 1 = 0 \implies \text{no real root of } x$$

therefore, no local minima exists

(b) Consider the following function of two variables $f(x, y) = \frac{1}{3}yx^2 - xy^2 + 2y$ and find all points with zero gradient $\nabla f = 0$ (extrema/saddle points). HINT: find all solutions $(x, y) \in \mathbb{R}^2$ to the following system of equations

$$\left\{ \begin{array}{l} \frac{\partial f}{\partial x} = 0 \\ \frac{\partial f}{\partial y} = 0 \end{array} \right. \quad (8)$$

Solution: (write your solution in this cell)

$$\left\{ \begin{array}{l} \frac{\partial f}{\partial x} = \frac{2}{3}yx - y^2 = 0 \\ \frac{\partial f}{\partial y} = \frac{1}{3}x^2 - 2xy + 2 = 0 \end{array} \right. \quad (10)$$

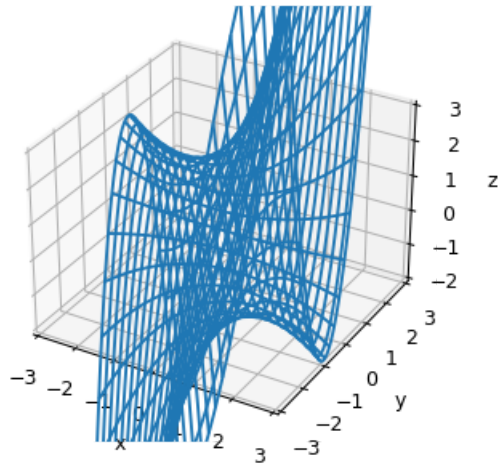
$$\Rightarrow (x, y) = (\sqrt{2}, \frac{2}{3}\sqrt{2}), (-\sqrt{2}, -\frac{2}{3}\sqrt{2}) \quad (11)$$

(c) Use "matplotlib" to display a 3D plot for function f(x,y). You should do this over the domain that includes all extrema points. Write your code in the cell below. Make sure you "run" the cell with your code below before saving and submitting your notebook. This will make your plot (the output of your code) visible when the notebook is opened for grading.

```
In [9]: # Solution: write your code in this cell
%matplotlib notebook
# NOTE: unlike "inline" mode activated in earlier cells, "notebook" allows interactive pl

import matplotlib.pyplot as plt

xs = np.linspace(-3, 3, 60)
ys = np.linspace(-3, 3, 60)
x, y = np.meshgrid(xs, ys)
z = y*x*x/3+x*y*y+2*y
ax = plt.figure(5).add_subplot(projection='3d')
ax.set_xlim(-3, 3)
ax.set_xlabel('x')
ax.set_ylim(-3, 3)
ax.set_ylabel('y')
ax.set_zlim(-2, 3)
ax.set_zlabel('z');
#ax.scatter(x, y, z)
ax.plot_wireframe(x, y, z)
```



Out[9]: <mpl_toolkits.mplot3d.art3d.Line3DCollection at 0x2989096b520>

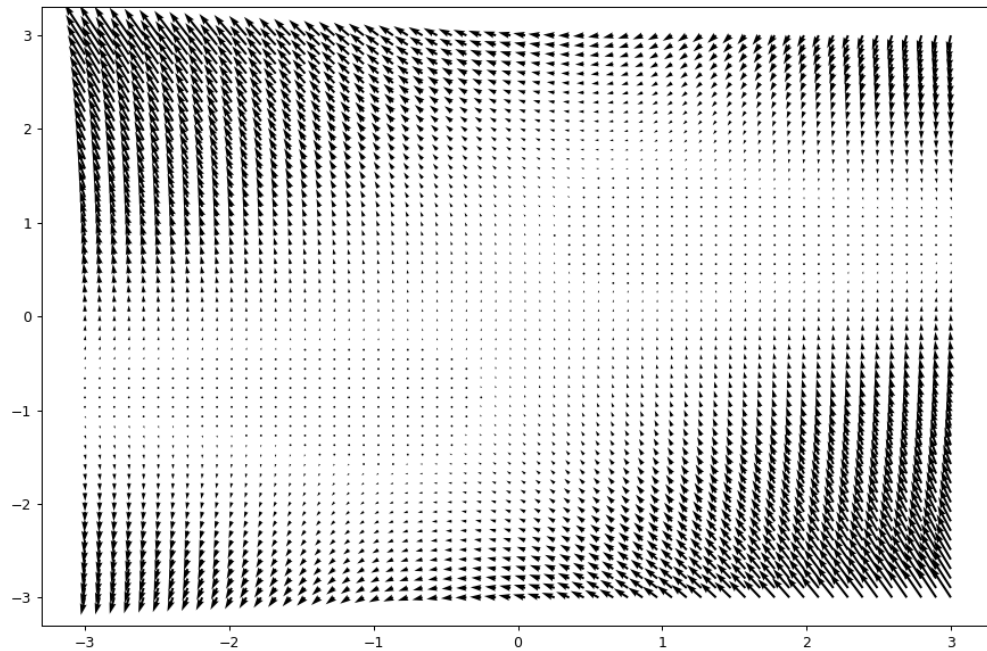
(d) Visualize vector field of gradients for $f(x, y)$ over the same domain. See one of the posted demo notebooks for inspiration.

```
In [10]: # Solution: write your code in this cell
# HINT: no need to repeat declaration """%matplotlib notebook""" if this cell is run after p

fig, ax = plt.subplots(figsize = (12, 8))
xs = np.linspace(-3, 3, 60)
ys = np.linspace(-3, 3, 60)
x, y = np.meshgrid(xs, ys)

u = 2*y*x/3 - y*y
v = x*x/3 - 2*x*y + 2

ax.quiver(x, y, u, v)
plt.show()
```



Problem 6

Prove that median filtering is not a linear image transformation. HINT: find a counter example showing that for some vectors of the same dimensions A and B ,

$$\text{Med}(A + B) \neq \text{Med}(A) + \text{Med}(B)$$

where operation $\text{Med}(X)$ returns median of the elements of vector X .

Solution: (write your solution in this cell)

Provide a counter-example to show the operation $\text{Med}(X)$ is not linear:

Assume A is

$$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Assume B is

$$\begin{pmatrix} 0 \\ -2 \\ 2 \end{pmatrix}$$

Note that $\text{Med}(A + B) = 1$ whereas $\text{Med}(A) + \text{Med}(B) = 2 + 0 = 2$

Therefore, $\text{Med}(X)$ is not a linear operation

Problem 7

(a) Image differentiation: Write code for a python function that estimates partial derivatives $d(x, y) := \frac{\partial}{\partial x} f(x, y)$ of any greyscale image f with respect to variable x . The function should return a real-valued matrix of the same size as the input image f . Use the "backward" difference approximation

$$\frac{\partial}{\partial x} f(x, y) \approx \frac{f(x, y) - f(x - \Delta, y)}{\Delta}$$

where Δ is the distance between pixels (you can assume $\Delta = 1$). You are not allowed to use colvolution (as in the sample notebook "convolution.ipynb"). **GENERAL NOTE ON NUMPY: as mentioned earlier, your numpy code should not use (double) for-loops for traversing the elsements of matrices. This is highly inefficient. You should learn to use appropriate numpy functions that avoid this.** For example, for this excercise you can use `numpy.roll` to compute image with pixels shifted to the left or right and use linear operations over images as matrices (pointwise addition/subtraction).

```
In [11]: # Solution: write your code in this cell
def ImgDifferentiate(gray_img):
    w=gray_img.shape[0]
    h=gray_img.shape[1]
    img = gray_img
    kernel = np.zeros(w)
    ans = np.empty([h,w])

    kernel[0] = -1
    kernel[1] = 1

    ans[:,0] = np.matmul(kernel, img)

    for i in range(w):
        if(i==0):
            continue;
        ans[:,i] = np.matmul(kernel, img)
        kernel = np.roll(kernel, 1)

    return ans.T
```

(b) Point processing: implement linear range transofrmation function $t(d)$ that maps partial derivatives $d = \frac{\partial}{\partial x} f$ computed above to the values in the range $[0, 255]$ so that $t(d_{max}) = 0$ for the maximum observed value of partial derivative d_{max} and $t(d_{min}) = 255$ for the minimum derivative d_{min} . Compute the transformed grayscale image $g(x, y) = t(d(x, y))$ and display the following images: the original function f (any grayscale image), its derivative $d = \frac{\partial}{\partial x} f$, and transformed image $g = t(d)$. Your matplotlib plots for d abd g should include the "color bar" indicating the corresponding ranges.

```
In [12]: # Solution: write your code in this cell

plt.figure(71,figsize = (12, 8))
plt.subplot(121)
img1 = toGrayScale_C(im)
plt.imshow((img1), cmap="gray")
plt.title("gray scale version")
```

```

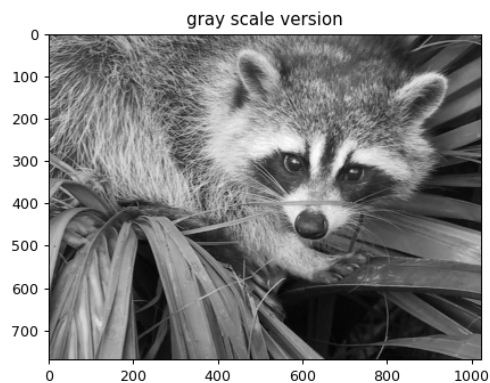
X, Y = np.meshgrid(np.arange(0, img1.shape[1], 1), np.arange(0, img1.shape[0], 1))
img2 = ImgDifferentiate(img1)
fig2 = plt.figure(72, figsize = (10,8))
ax2 = fig2.gca(projection='3d')
surf = ax2.plot_surface(X, Y, img2, rstride=1, cstride=1, cmap=matplotlib.cm.coolwarm, 1
fig2.colorbar(surf, shrink=0.5, aspect=5)

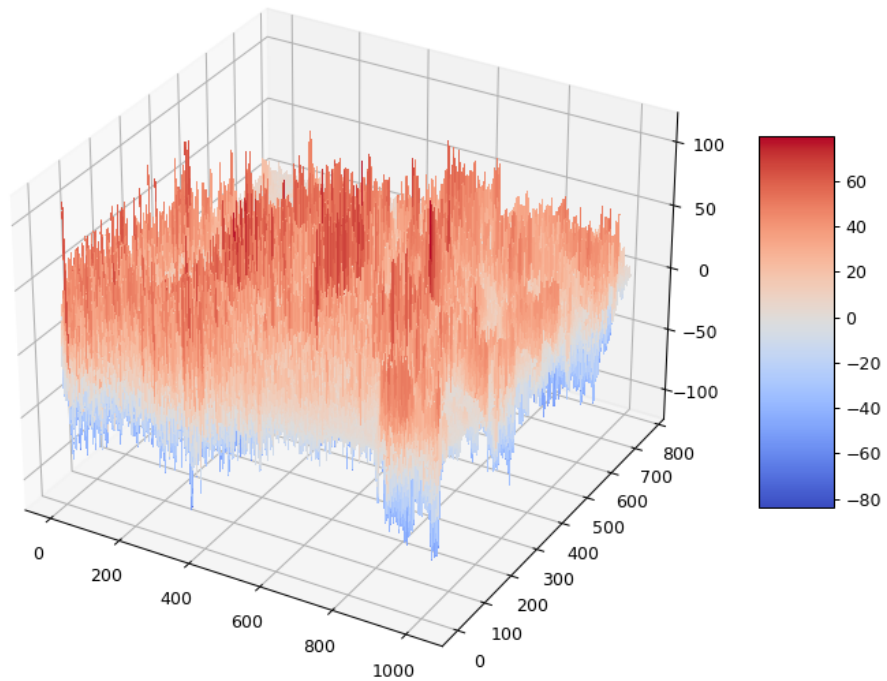
def func_t(diffimg):
    minvalue = np.min(diffimg)
    maxvalue = np.max(diffimg)
    return (255/(maxvalue - minvalue)) * (diffimg - minvalue)

img3 = func_t(img2)
fig3 = plt.figure(73, figsize = (10,8))
ax3 = fig3.gca(projection='3d')
surf2 = ax3.plot_surface(X, Y, img3, rstride=1, cstride=1, cmap=matplotlib.cm.coolwarm,
fig3.colorbar(surf2, shrink=0.5, aspect=5)

plt.show()

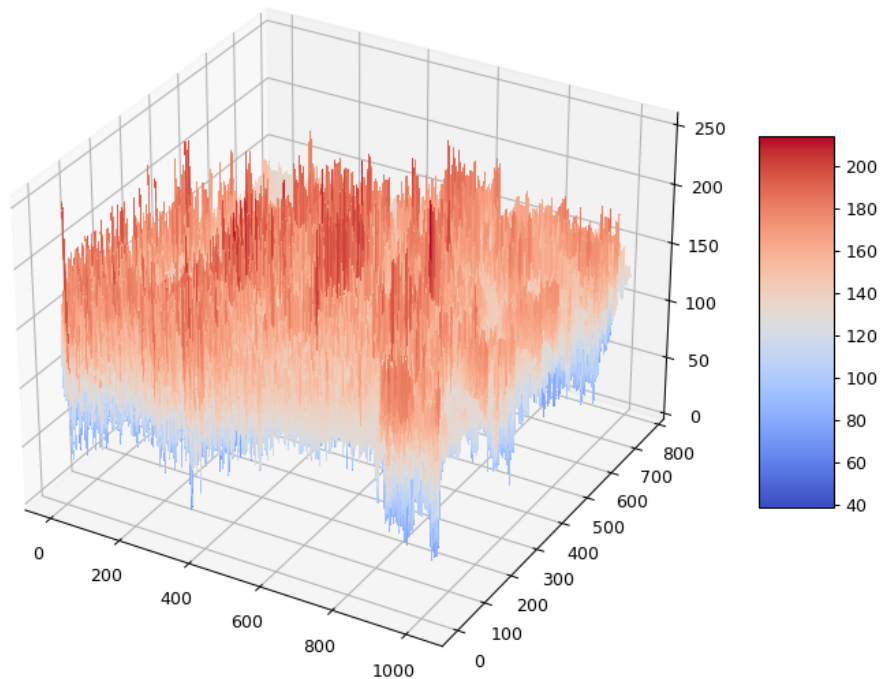
```





C:\Users\23700\AppData\Local\Temp\ipykernel_10596\3288931626.py:12: MatplotlibDeprecation Warning: Calling gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two minor releases later, gca() will take no keyword arguments. The gca() function should only be used to get the current axes, or if no axes exist, create new axes with default keyword arguments. To create a new axes with non-default arguments, use plt.axes() or plt.subplot().

```
ax2 = fig2.gca(projection='3d')
```



C:\Users\23700\AppData\Local\Temp\ipykernel_10596\3288931626.py:24: MatplotlibDeprecation Warning: Calling gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two minor releases later, gca() will take no keyword arguments. The gca() function should only be used to get the current axes, or if no axes exist, create new axes with default keyword arguments. To create a new axes with non-default arguments, use plt.axes() or plt.subplot().

```
ax3 = fig3.gca(projection='3d')
```

(c) Write code demonstrating partial derivatives for the same image with substantial amount of added Gaussian noise (you can use code for noise generation from Filtering.ipynb).

In []:

In [13]:

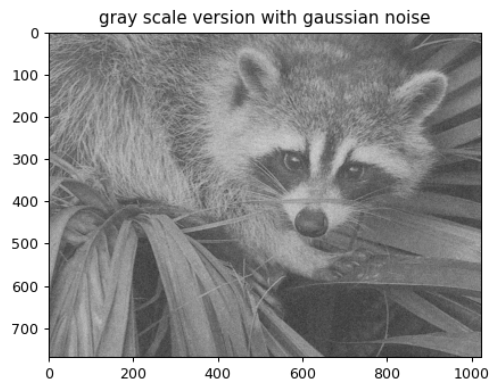
```
# Solution: write your code in this cell
sigma = 30.0
gauss = np.random.normal(0.0, sigma, (img1.shape[0], img1.shape[1]))
img4 = img1 + gauss
plt.figure(74, figsize = (12, 8))
plt.subplot(121)
plt.imshow((img4), cmap="gray")
plt.title("gray scale version with gaussian noise")
fig4 = plt.figure(75, figsize = (10, 8))
ax4 = fig4.gca(projection='3d')
surf3 = ax4.plot_surface(X, Y, img3, rstride=1, cstride=1, cmap=matplotlib.cm.coolwarm,
fig4.colorbar(surf3, shrink=0.5, aspect=5)
plt.title("partial derivatives without Gaussian noise")
```

```

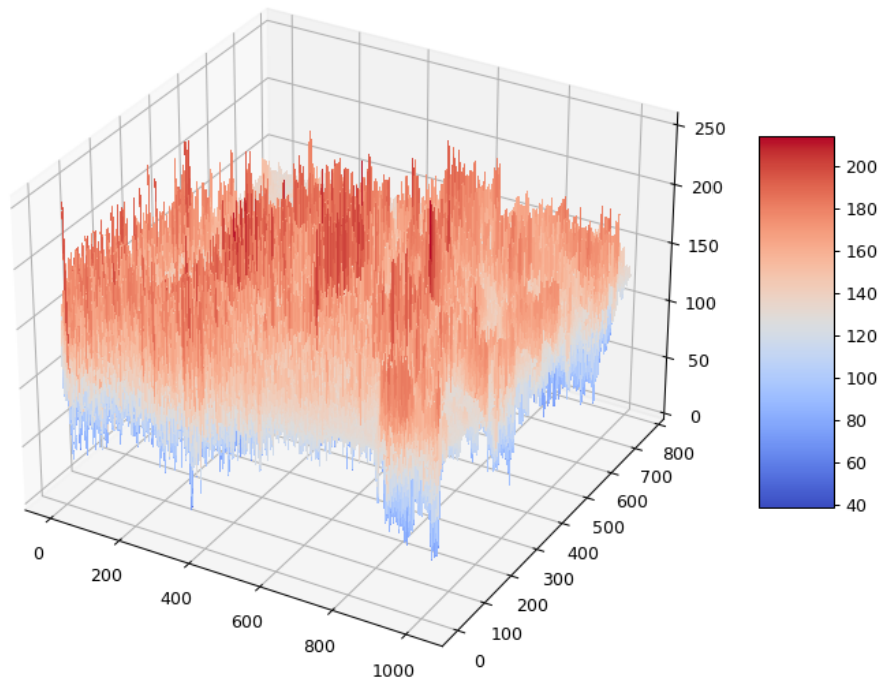
img5 = ImgDifferentiate(img4)
img6 = func_t(img5)
fig5 = plt.figure(76,figsize = (10,8))
ax5 = fig5.gca(projection='3d')
surf4 = ax5.plot_surface(X, Y, img6, rstride=1, cstride=1, cmap=matplotlib.cm.coolwarm,
fig5.colorbar(surf4, shrink=0.5, aspect=5)
plt.title("partial derivatives with Gaussian noise")

plt.show()

```



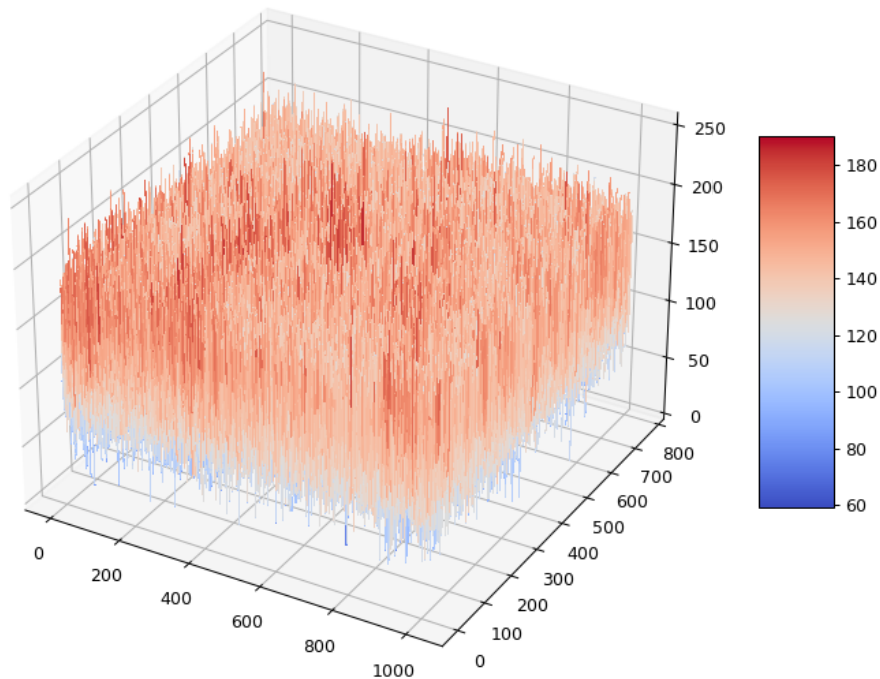
partial derivatives without Gaussian noise



C:\Users\23700\AppData\Local\Temp\ipykernel_10596\2369746284.py:10: MatplotlibDeprecation Warning: Calling `gca()` with keyword arguments was deprecated in Matplotlib 3.4. Starting two minor releases later, `gca()` will take no keyword arguments. The `gca()` function should only be used to get the current axes, or if no axes exist, create new axes with default keyword arguments. To create a new axes with non-default arguments, use `plt.axes()` or `plt.subplot()`.

```
ax4 = fig4.gca(projection='3d')
```

partial derivatives with Gaussian noise



C:\Users\23700\AppData\Local\Temp\ipykernel_10596\2369746284.py:18: MatplotlibDeprecation Warning: Calling gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting two minor releases later, gca() will take no keyword arguments. The gca() function should only be used to get the current axes, or if no axes exist, create new axes with default keyword arguments. To create a new axes with non-default arguments, use plt.axes() or plt.subplot().

```
ax5 = fig5.gca(projection='3d')
```

Problem 8


(a) In this problem we use $\nabla I(x, y)$ to denote a gradient of image intensities at point (x, y) only to emphasize this dependence of the gradient on location. Assume that $\nabla I(x, y)$ is a non-zero vector at a given point (x, y) . What is the rank of matrix $\nabla I(x, y) \cdot \nabla I^T(x, y)$ and why?

Solution: (formal arguments)

$$\text{Assume } \nabla I(x, y) = \begin{bmatrix} u \\ v \end{bmatrix}, \text{ then } \nabla I(x, y) \cdot \nabla I^T(x, y) = \begin{bmatrix} u^2 & uv \\ uv & v^2 \end{bmatrix}$$

Note that it is non-zero, therefore, the first row is not a multiple of the second row, therefore, if $u = v$, it is rank 1, otherwise, its rank have to be 2.

(b) Assume that an image patch (window w) contains a straight intensity edge (as in window W_b below). What should be the rank of Harris matrix at that patch/window $M_w = \sum_{(x,y) \in w} \nabla I(x, y) \cdot \nabla I^T(x, y)$ and why?

NOTE: here we assume that w stands for a subset of pixels in the window, rather than 0-1 indicator function for this window (as in the lecture notes). Both types of notation is common. While 0-1 indicators $w(x, y)$ easily extend to weighted support functions, we do not need this generality for this exercise and preferred a slightly simpler set notation. 

Solution:

The rank of Harris matrix should still be 2, since at this case, all pixels that have non-zero gradient seems to have the same gradient, thus, the Harris matrix is the multiple of matrix mentioned in a) which is showed that the rank is 2 or 1 (1 only occurs when $u = v$).

(c) What is the rank of Harris matrix for window W_c containing a corner at an intersection of two straight edges? Provide a formal proof. HINT: You can assume that pixels in W_c have either a zero-gradient, or one of two distinct gradient vectors corresponding to two straight edges.

Solution:

Assume that all pixels in this window have either a zero-gradient, or one of two non-zero distinct gradient vectors $\begin{bmatrix} u \\ v \end{bmatrix}$ or $\begin{bmatrix} x \\ y \end{bmatrix}$

Obviously, the Harris matrix in this case have the form: $\begin{bmatrix} u^2 + x^2 & uv + xy \\ uv + xy & v^2 + y^2 \end{bmatrix}$

Usually, its rank of 2 since the first row is not a multiple of the second row. Except when $u^2y^2 + x^2v^2 = 2uvxy$, the rank is 1.

Case 1: if $2uvxy = 0$

$$u = 0 \implies x = 0$$

$$v = 0 \implies y = 0$$

$$x = 0 \implies u = 0$$

$$y = 0 \implies v = 0$$

This is impossible, since the both horizontal or vertical part is 0, then two edges are parallel which may not lead to a corner.

Case 2: if $2uvxy \neq 0$

$$\text{We have } uy(uy - xv) = xv(uy - xv) \implies uy = xv$$

Note that this is impossible, since if $uy = xv$, two gradient vectors have the same direction, which means we are not at the corner.

Therefore, overall, the rank of Harris matrix in this case is 2.

In []: