Isotropic smoothing of image via Heat equation

import library

In [ ]:
```
import numpy as np
import matplotlib.image as img
import matplotlib.pyplot as plt
from matplotlib import cm
import matplotlib.colors as colors
from skimage import color
from skimage import io
```

load input image

- filename for the input image is 'barbara_color.jpeg'

In [ ]:
```

I0 = io.imread('barbara_color.jpeg')
```

check the size of the input image

In [ ]:
```
# ++++++++++++++++++++++++++++++++++++++++++++++++++
# complete the blanks
#
num_row     = I0.shape[0]
num_column  = I0.shape[1]
num_channel = I0.shape[2]
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++

print('number of rows of I0 = ', num_row)
print('number of columns of I0 = ', num_column)
print('number of channels of I0 = ', num_channel)
```

```
number of rows of I0 =  512
number of columns of I0 =  512
number of channels of I0 =  3
```

convert the color image into a grey image

In [ ]:
```
# ++++++++++++++++++++++++++++++++++++++++++++++++++
# complete the blanks
#
I = color.rgb2gray(I0)

num_row     = I.shape[0]
num_column  = I.shape[1]
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
print('number of rows of I = ', num_row)
print('number of columns of I = ', num_column)
```

```
number of rows of I =  512
number of columns of I =  512
```

normalize the converted image

- normalize the converted grey scale image so that its maximum value is 1 and its minimum value is 0

In [ ]:

```
# ++++++++++++++++++++++++++++++++++++++++++++++++++
# complete the blanks
#
I = (I - np.min(I))/np.ptp(I)
#
# ++++++++++++++++++++++++++++++++++++++++++++++++++

print('maximum value of I = ', np.max(I))
print('minimum value of I = ', np.min(I))
```

```
maximum value of I =  1.0
minimum value of I =  0.0
```

define a function to compute the derivative of input matrix in x(row)-direction

- forward difference : $I[x+1, y] - I[x, y]$

In [ ]:

```
def compute_derivative_x_forward(I):

    D = np.zeros(I.shape)

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    D = np.pad(I, (1,1), mode= 'edge')
    D = np.roll(D, shift=-1, axis=0) - D
    D = D[1:I.shape[0]+1 , 1:I.shape[1]+1]


    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return D
```

- backward difference : $I[x, y] - I[x-1, y]$

In [ ]:

```
def compute_derivative_x_backward(I):

    D = np.zeros(I.shape)

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #
```

```
    D = np.pad(I, (1,1), mode = 'edge')
    D = D - np.roll(D, shift=1, axis=0)
    D = D[1:I.shape[0]+1, 1:I.shape[1]+1]


    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return D
```

define a function to compute the derivative of input matrix in y(column)-direction

- forward difference : $I[x, y+1] - I[x, y]$

In [ ]:

```
def compute_derivative_y_forward(I):

    D = np.zeros(I.shape)

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    D = np.pad(I, (1,1), mode = 'edge')
    D = np.roll(D, shift=-1, axis =1) -D
    D = D[1:I.shape[0]+1, 1:I.shape[1]+1]

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return D
```

- backward difference : $I[x, y] - I[x, y-1]$

In [ ]:

```
def compute_derivative_y_backward(I):

    D = np.zeros(I.shape)

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    D = np.pad(I, (1,1), mode = 'edge')
    D = D - np.roll(D, shift=1, axis=1)
    D = D[1:I.shape[0]+1, 1:I.shape[1]+1]


    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return D
```

define a function to compute the laplacian of input matrix

- $\Delta I = \nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$
- $\Delta I = I[x+1, y] + I[x-1, y] + I[x, y+1] + I[x, y-1] - 4 * I[x, y]$
- ΔI = derivative_x_forward - derivative_x_backward + derivative_y_forward - derivative_y_backward

In [ ]:

```
def compute_laplace(I):

    laplace = np.zeros(I.shape)

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    laplace = compute_derivative_x_forward(I) - compute_derivative_x_backward(I) + co

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return laplace
```

define a function to compute the heat equation of data I with a time step

- $I = I + \delta t * \Delta I$

In [ ]:

```
def heat_equation(I, time_step):

    I_update = np.zeros(I.shape)

    # ++++++++++++++++++++++++++++++++++++++++++++++++++
    # complete the blanks
    #

    I_update = I + time_step * compute_laplace(I)

    #
    # ++++++++++++++++++++++++++++++++++++++++++++++++++

    return I_update
```

run the heat equation over iterations

In [ ]:

```
def run_heat_equation(I, time_step, number_iteration):

    I_update = np.zeros(I.shape)

    for t in range(number_iteration):
        # ++++++++++++++++++++++++++++++++++++++++++++++++++
        # complete the blanks
        #

        I_update = heat_equation(I, time_step)
        I = I_update

        #
        # ++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
    return I_update
```

---

functions for presenting the results

In [ ]:
```
def function_result_01():

    plt.figure(figsize=(8,6))
    plt.imshow(I0)
    plt.show()
```

In [ ]:
```
def function_result_02():

    plt.figure(figsize=(8,6))
    plt.imshow(I, cmap='gray', vmin=0, vmax=1, interpolation='none')
    plt.show()
```

In [ ]:
```
def function_result_03():

    L = compute_laplace(I)

    plt.figure(figsize=(8,6))
    plt.imshow(L, cmap='gray')
    plt.show()
```

In [ ]:
```
def function_result_04():

    time_step   = 0.25
    I_update    = heat_equation(I, time_step)

    plt.figure(figsize=(8,6))
    plt.imshow(I_update, vmin=0, vmax=1, cmap='gray')
    plt.show()
```

In [ ]:
```
def function_result_05():

    time_step           = 0.25
    number_iteration    = 128

    I_update = run_heat_equation(I, time_step, number_iteration)

    plt.figure(figsize=(8,6))
    plt.imshow(I_update, vmin=0, vmax=1, cmap='gray')
    plt.show()
```

In [ ]:
```
def function_result_06():

    time_step           = 0.25
    number_iteration    = 512
```

```
    I_update = run_heat_equation(I, time_step, number_iteration)

    plt.figure(figsize=(8,6))
    plt.imshow(I_update, vmin=0, vmax=1, cmap='gray')
    plt.show()
```

In [ ]:

```
def function_result_07():

    L = compute_laplace(I)

    value1 = L[0, 0]
    value2 = L[-1, -1]
    value3 = L[100, 100]
    value4 = L[200, 200]

    print('value1 = ', value1)
    print('value2 = ', value2)
    print('value3 = ', value3)
    print('value4 = ', value4)
```

In [ ]:

```
def function_result_08():

    time_step   = 0.25
    I_update    = heat_equation(I, time_step)

    value1 = I_update[0, 0]
    value2 = I_update[-1, -1]
    value3 = I_update[100, 100]
    value4 = I_update[200, 200]

    print('value1 = ', value1)
    print('value2 = ', value2)
    print('value3 = ', value3)
    print('value4 = ', value4)
```

In [ ]:

```
def function_result_09():

    time_step           = 0.25
    number_iteration    = 128

    I_update = run_heat_equation(I, time_step, number_iteration)

    value1 = I_update[0, 0]
    value2 = I_update[-1, -1]
    value3 = I_update[100, 100]
    value4 = I_update[200, 200]

    print('value1 = ', value1)
    print('value2 = ', value2)
    print('value3 = ', value3)
    print('value4 = ', value4)
```

In [ ]:

```
def function_result_10():

    time_step           = 0.25
    number_iteration    = 512

    I_update = run_heat_equation(I, time_step, number_iteration)
```

```
    value1 = l_update[0, 0]
    value2 = l_update[-1, -1]
    value3 = l_update[100, 100]
    value4 = l_update[200, 200]

    print('value1 = ', value1)
    print('value2 = ', value2)
    print('value3 = ', value3)
    print('value4 = ', value4)
```

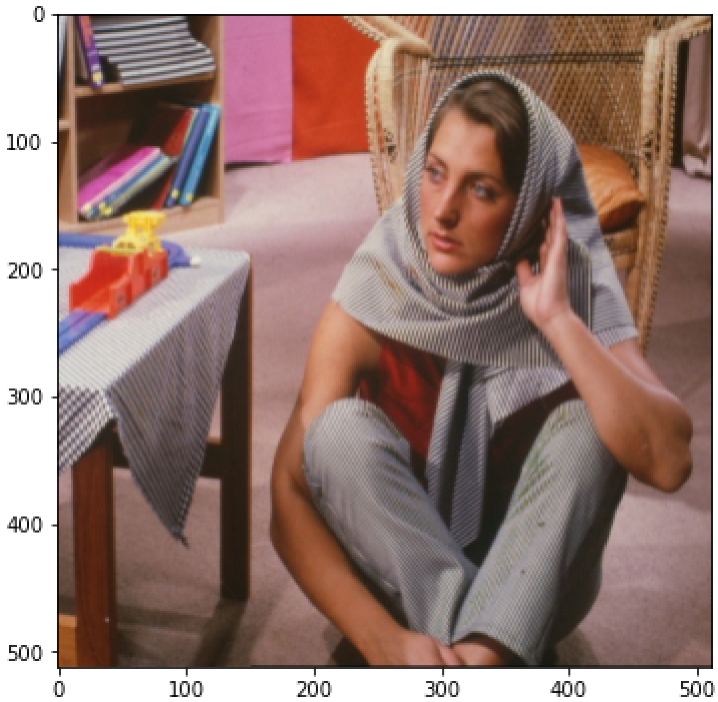
---

results

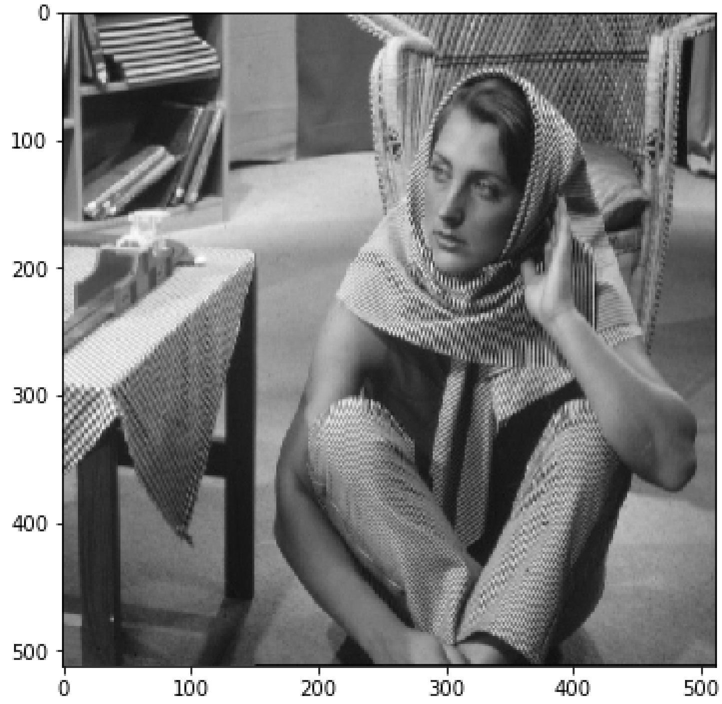
In [ ]:

```
number_result = 10

for i in range(number_result):
    title = '## [RESULT {:02d}]'.format(i+1)
    name_function = 'function_result_{:02d}()'.format(i+1)

    print('**************************************************')
    print(title)
    print('**************************************************')
    eval(name_function)
```

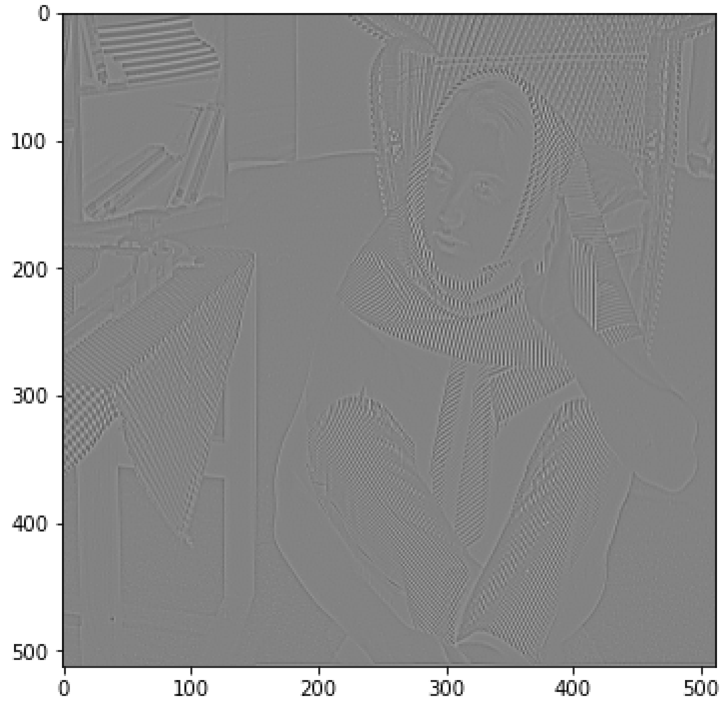
```
**************************************************
## [RESULT 01]
**************************************************
```



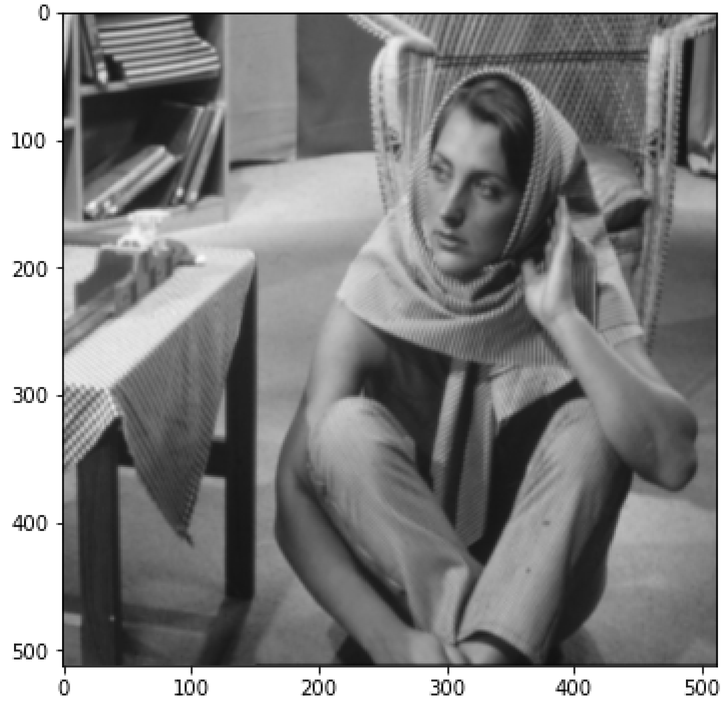

```
**************************************************
## [RESULT 02]
**************************************************
```



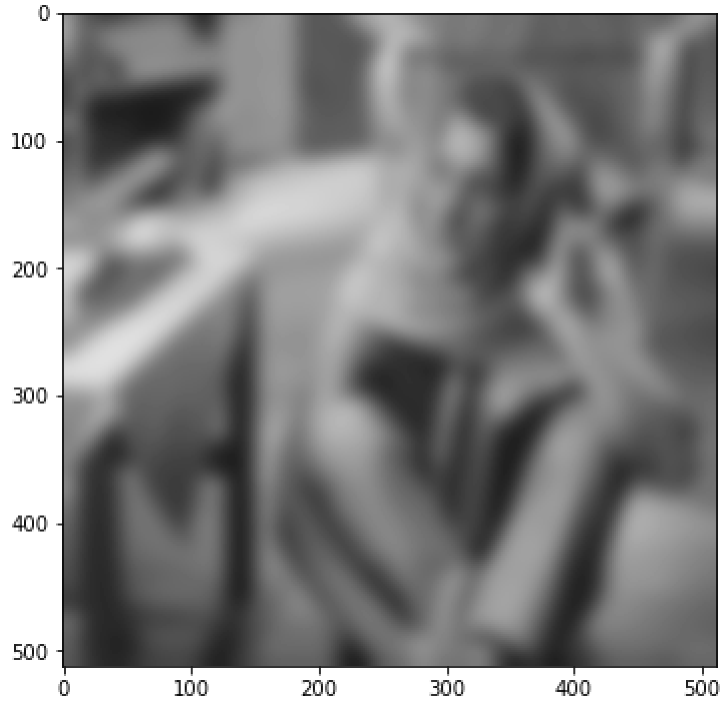

```
**************************************************
## [RESULT 03]
**************************************************
```



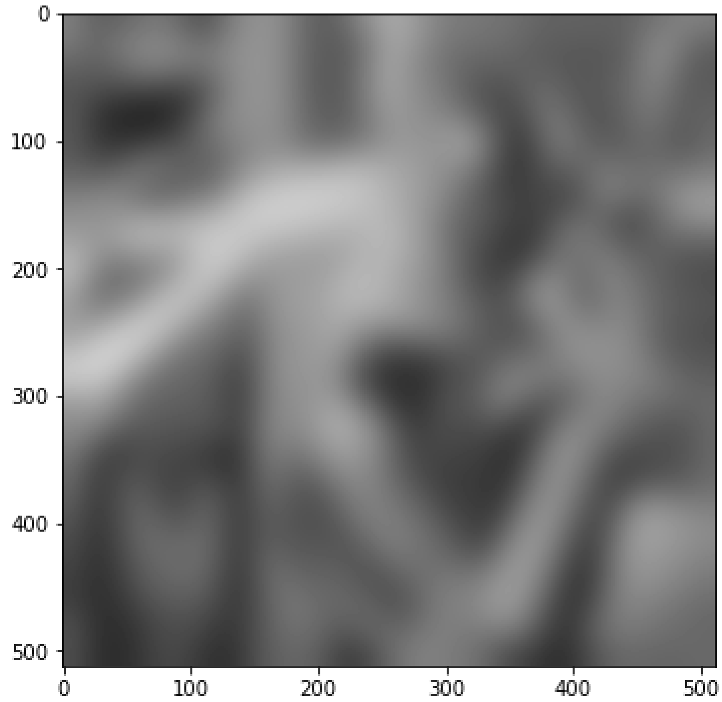

```
**************************************************
## [RESULT 04]
**************************************************
```




```
************************************************
## [RESULT 05]
************************************************
```




```
************************************************
## [RESULT 06]
************************************************
```



```
**************************************************
## [RESULT 07]
**************************************************
value1 =  0.3842307424134238
value2 =  0.3473015945865081
value3 =  -0.11096969959074021
value4 =  -0.057368751329410106
**************************************************
## [RESULT 08]
**************************************************
value1 =  0.1205466403535327
value2 =  0.16065986420375264
value3 =  0.5126456173363071
value4 =  0.5929656202312226
**************************************************
## [RESULT 09]
**************************************************
value1 =  0.588913112688924
value2 =  0.4010963411433222
value3 =  0.3678389043817923
value4 =  0.6014962322332662
**************************************************
## [RESULT 10]
**************************************************
value1 =  0.4905270260686319
value2 =  0.4088943172743327
value3 =  0.351276787536652
value4 =  0.6310803621724141
```

In [ ]:

In [ ]: