Scaling of Images

Importing important libraries

In [1]:

```
#%matplotlib inline
import numpy as np
#from matplotlib import pyplot as plt
import cv2
import imageio
import SimpleITK
import sys
from pylab import *
```

Reading the images

In [2]:

```
cells = cv2.imread("cells_scale.png",0)
lena = cv2.imread("lena_translate.png",0)
pisa = cv2.imread("pisa_rotate.png",0)
```

Defining Bilinear Interpolation

This function Computes the intensity at source point by bilinearly interpolating intensities in the immediate 2 X 2 neighborhood of the source point(si,sj).

In [3]:

```
def bilinear_interpolation(src,si,sj):
    #si , sj = src_pt
    #si= ti-ty
    #sj= tj-tx
    ## Translating back
    i=int(np.floor(si)) +cx
    j=int(np.floor(sj)) +cy     ## Here i,j are the co-ordinate points of the
    t_l = i , j

    ##Now the remaining three co-ordinates with respect to i,j will be
    t_r = i ,j+1    # Top right
    b_l = i+1 , j   # Bottom Left
    b_r = i+1 , j+1 # Bottom Right

    ## distance of source point from the top left corner would be
    di = si +cx - i
    dj = sj +cy - j

    ## Now calculating the pixel value at the source point by using bilinear
    if t_l[0] >= np.shape(src)[0]-1 or t_l[1] >=np.shape(src)[1]-1 or t_l[0]<
        pxl_val = 0
    else :
        pxl_val = (1-di)*(1-di)*src[t_l] + (1-di)*(dj)*src[t_r] + (di)*(1-dj)

    #return np.unit8(pxl_val)
    return pxl_val
```

Defining transform

In [4]:

```
#Initialising the target image
trg = np.zeros(np.shape(cells))

#Defining the scaling factor about x and y axis
sx = 1.3
sy = 1.3

# Defining the centre of the source image as the scaling would be done about
cx = int(np.floor(np.shape(cells)[0]/2))
cy = int(np.floor(np.shape(cells)[1]/2))

#Defing the transform for the
def transform(src):
    r , c = np.shape(src)

    ## iterate over the target image and assign all the pixel values to them
    for ti in range(r):
        for tj in range(c):
            """
            # To scale around a point(centre) we will first translate to that
            # Then we will apply scaling as we would apply scaling about the

            """
            si= (ti-cx)/sx #+ cx
            sj= (tj-cy)/sy #+cy
            #si= ti-ty
            #si = np.cos(theta)*(ti-cx) - np.sin(theta)*(tj-cy) +cx
            #sj = np.sin(theta)*(ti-cx) + np.cos(theta)*(tj-cy) +cy
            #si,sj=np.array([ti,tj,1])#@transformation.T
            #si,sj=si/z,sj/z

            #sj= tj-tx
            #if (0<= si < r-1  & 0<= sj < c-1):
            trg[ti][tj]=bilinear_interpolation(src,si,sj)
            #else:
                #trg[ti][tj]=0
    return trg
```

Defining Scale function

In [5]:

```
def scale(src):


    return transform(src)
```

Calling scale function

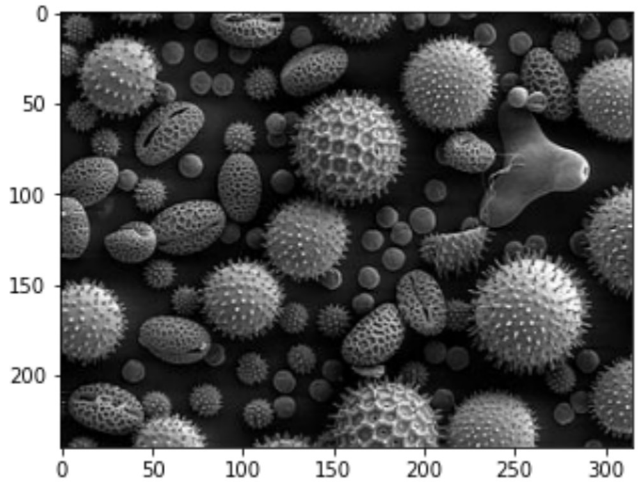
In [6]:

```
cells_scaled=scale(cells)
```

Original Image

In [7]:

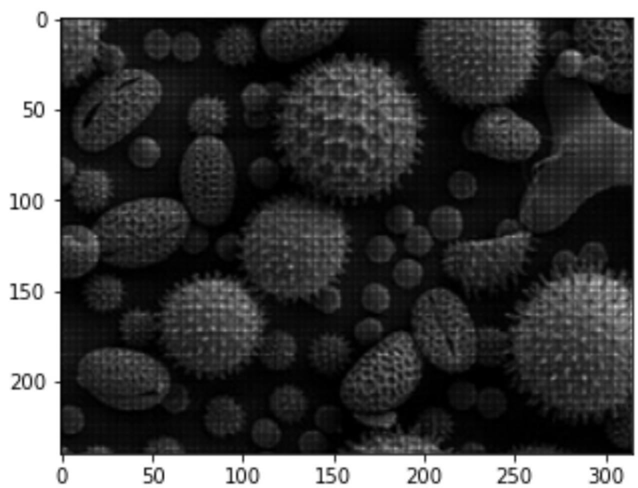
```
plt.imshow(cells,cmap='gray')
plt.show()
```



Scaled Image

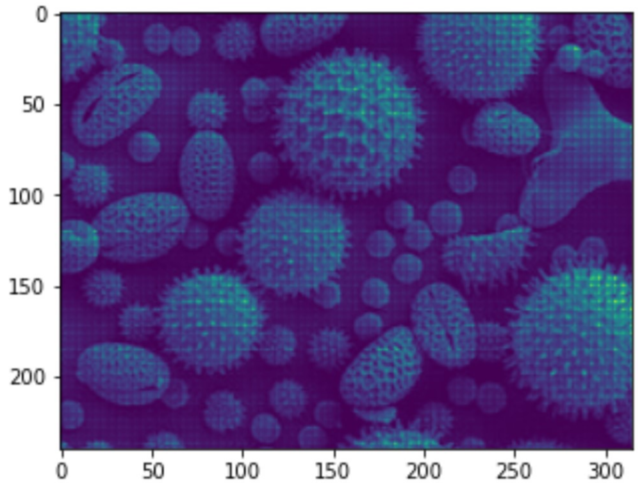
In [8]:

```
plt.imshow(cells_scaled,cmap='gray')
plt.show()
```



In [9]:

```
plt.imshow(cells_scaled)
plt.show()
```



Observations

1. The value of theta that will make the pisa tower to becom straight is 4 degree in anti-clockwise direction.
2. In the output images (target image) we have black pixels because we do not have the information corresponding to it as to what pixel value we should at the part. So we assign 0 values at that point and hence we get black part in the output image.
3. Since we did target to source mapping so we do not get holes in the target image as all points in the target image is interated to get some corresponding pixel value from the image source.
4. By doing the nearest neighbour interpolation we will get checker board effect which we do not get using the bilinear interpolation.