DIP Assignment-1

Group No 7 Members

->PALETI KRISHNASAI- CED18I039

->KATTE PRAHLAD GOWTHAM - COE18B029

->HRISHIKESH RAJESH MENON - COE18B024

Question 1

1. Read a matrix of size 5*5 and find the following by using a user-defined function.

a. sum
b. maximum
c. mean
d. median
e. mode
f. standard deviation
h. frequency distribution

In [1]:
```
#Read the matrix
M=[]


# M=[ [1 ,  2 ,  3 ,  4 ,  5],
#     [6 ,  7 ,  8 ,  9 ,  1],
#     [2 ,  3 ,  4 ,  5 ,  6],
#     [7 ,  8 ,  9 ,  1 ,  2],
#     [3 ,  4 ,  5 ,  6 ,  7] ]

# size = int(input("Enter Matrix Sum:-"))

size = 5

for i in range(size):
    l = []
    for j in range(size):
        l.append(int(input("Enter Matrix Element:-")))
    M.append(l)

print('\n'.join([''.join(['{:4}'.format(item) for item in row]) for row in M]))
```

```
   1   2   3   4   5
   6   7   8   9   1
   2   3   4   5   6
   7   8   9   1   2
   3   4   5   6   7
```

In [2]:
```
def MatrixSum(mat):
    sum=0
    for i in range(size):
        for j in range(size):
            sum += mat[i][j]

    return sum


def MatrixMax(mat):
    max=0
    for i in range(size):
        for j in range(size):
            if max<=mat[i][j]:
                max = mat[i][j]

    return max

def MatrixMean(mat):
    mean=0
    for i in range(size):
        for j in range(size):
            mean += mat[i][j]

    mean= mean/(size*size)

    return mean

def MatrixMedian(mat):
    mi = mat[0][0]
    mx = 0
    for i in range(size):
        if mat[i][0] < mi:
            mi = mat[i][0]
        if mat[i][size-1] > mx :
            mx =  mat[i][size-1]

    desired = (size * size + 1) // 2

    while (mi < mx):
        mid = mi + (mx - mi) // 2
        place = [0];

        # Find count of elements smaller than mid
        for i in range(size):
             j = upper_bound(mat[i], mid)
             place[0] = place[0] + j
        if place[0] < desired:
            mi = mid + 1
        else:
            mx = mid
    # print ("Median is", mi)
    return mi

def MatrixMode(mat):

    count={}
```

```
    for row in mat:
        for element in row:
            # print(element)
            if(element in count):
                count[element] +=1
            else:
                count[element]=1

    mode=0
    mcount=0
    for item in count:
        if count[item]>mcount:
            mode=item
            mcount=count[item]

    return mode


def MatrixStd(Mat): # We find variance and then square root it
    Mat
    sum=0
    mean= MatrixMean(Mat)
    Var=[]
    for i in range(size):
        for j in range(size):
            Var.append(Mat[i][j] - mean)

            Var[i*size+j] *= Var[i*size+j]

    for i in range(size):
        for j in range(size):
            sum += Var[i*size+j]

    std = float((sum / (size*size)) ** (1/2))

    return std

def FreqDist(mat):
    count={}

    for row in mat:
        for element in row:
            # print(element)
            if(element in count):
                count[element] +=1
            else:
                count[element]=1

    return count

print("Sum:-"+str(MatrixSum(M)))

print("Max:-"+str(MatrixMax(M)))

print("Mean:-"+str(MatrixMean(M)))

print("Median:-"+str(MatrixMedian(M)))

print("Mode:-"+str(MatrixMode(M)))

print("Standard Deviation:-"+str(MatrixStd(M)))

freq=FreqDist(M)

print("-"*25+"\nFrequency Distribution\n"+"-"*25)
print("{:<8} {:<10}".format('Element','Frequency'))

for item in freq:
    print("{:<8} {:<10}".format(item,freq[item]))
```

```
Sum:-118
Max:-9
Mean:-4.72
Median:-6
Mode:-1
Standard Deviation:-2.474186735070738
-------------------------
Frequency Distribution
-------------------------
Element  Frequency
1        3
2        3
3        3
4        3
5        3
6        3
7        3
8        2
9        2
```

Question 2

Read the matrix size through the keyboard and create a random matrix of integers ranging from 0 to 10 and compute all the above functions listed in question 1.

In [3]:
```
import random

M=[]

size = int(input("Enter Matrix Sum:-"))

for i in range(size):
    l = []
    for j in range(size):
        l.append(random.randint(0,10))
    M.append(l)

print('\n'.join([''.join(['{:4}'.format(item) for item in row]) for row in M]))
```

```
   4   8   8   1   7
   9   4   6   8   2
   2   7   9   9   8
   1   8   5   5   6
   5  10   0   0   5
```

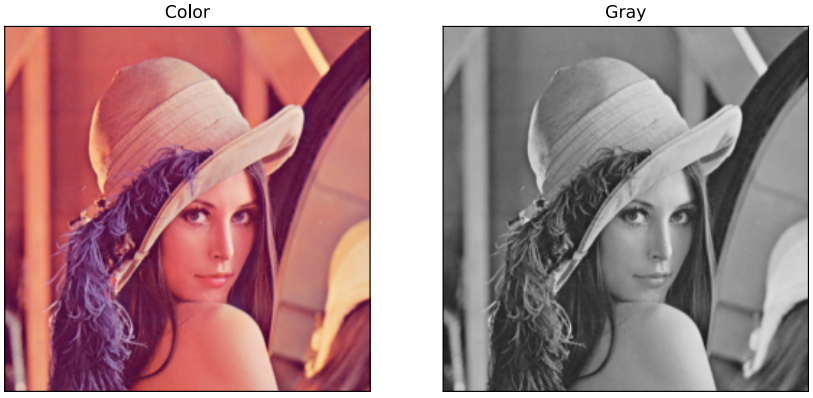
In [4]:
```
#Recalling Previous Functions
print("Sum:-"+str(MatrixSum(M)))

print("Max:-"+str(MatrixMax(M)))

print("Mean:-"+str(MatrixMean(M)))

print("Median:-"+str(MatrixMedian(M)))

print("Mode:-"+str(MatrixMode(M)))

print("Standard Deviation:-"+str(MatrixStd(M)))

freq=FreqDist(M)

print("-"*25+"\nFrequency Distribution\n"+"-"*25)
print("{:<8} {:<10}".format('Element','Frequency'))

for item in freq:
    print("{:<8} {:<10}".format(item,freq[item]))
```

```
Sum:-137
Max:-10
Mean:-5.48
Median:-5
Mode:-8
Standard Deviation:-2.994929047573582
-------------------------
Frequency Distribution
-------------------------
Element  Frequency
4        2
8        5
1        2
7        2
9        3
6        2
2        2
5        4
10       1
0        2
```

Question 3

Take a Lena image and convert it into grayscale. Add three different types of noises(salt and pepper, additive Gaussian noise, speckle), each noise in the sets of 5,10,15,20,25,30. Take average for each set and display the average images. Report the observation made.

In [2]:
```
from matplotlib import pyplot as plt
import numpy as np
import cv2
from skimage.util import random_noise

image = cv2.imread("lena512color.tiff",1)
#Converting to grey
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#Setting from BGR to RGB Mode as Matplotlib supports RGB
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
gray = cv2.cvtColor(gray, cv2.COLOR_BGR2RGB)


fig, (ax1, ax2) = plt.subplots(1,2,figsize=(10,10))

ax1.imshow(image)
ax1.set_title("Color")
ax1.set_xticks([])
ax1.set_yticks([])


ax2.imshow(gray)
ax2.set_title("Gray")
ax2.set_xticks([])
ax2.set_yticks([])

fig.show()



# cv2.imshow('Original image',image)
# cv2.imshow('Gray image', gray)
# cv2.waitKey(0)
# cv2.destroyAllWindows()
```



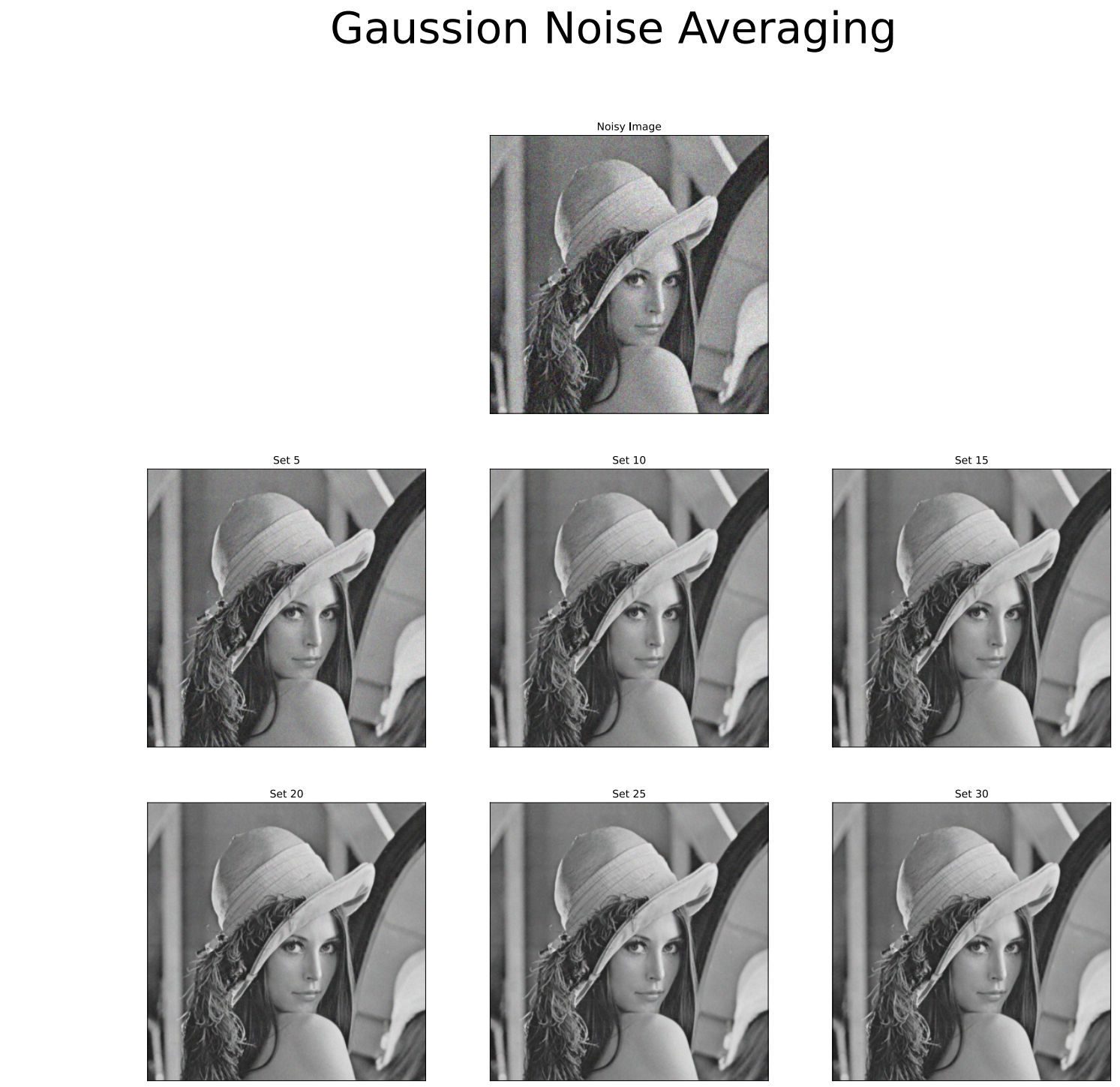
In [4]:

```
#Create Noisy Images
gauss=[]
sp=[]
speckle=[]

for i in range(30):

    gauss.append(random_noise(gray,mode='gaussian',seed=None,clip=True))

    sp.append(random_noise(gray,mode='s&p',seed=None,clip=True))

    speckle.append(random_noise(gray,mode='speckle',seed=None,clip=True))
```

In [5]:

```
#Gauss Noise

Guass_Array = np.array([np.array(im) for im in gauss[:5]])

Gauss_5= np.average(Guass_Array,axis=0)

Guass_Array = np.array([np.array(im) for im in gauss[:10]])

Gauss_10= np.average(Guass_Array,axis=0)

Guass_Array = np.array([np.array(im) for im in gauss[:15]])

Gauss_15= np.average(Guass_Array,axis=0)

Guass_Array = np.array([np.array(im) for im in gauss[:20]])

Gauss_20= np.average(Guass_Array,axis=0)

Guass_Array = np.array([np.array(im) for im in gauss[:25]])

Gauss_25= np.average(Guass_Array,axis=0)

Guass_Array = np.array([np.array(im) for im in gauss[:30]])

Gauss_30= np.average(Guass_Array,axis=0)

fig, axs = plt.subplots(3, 3,figsize=(20,20))

axs[0,1].imshow(gauss[0])
axs[0,1].set_title("Noisy Image")
axs[0,1].set_xticks([])
axs[0,1].set_yticks([])


axs[1,0].imshow(Gauss_5)
axs[1,0].set_title("Set 5")
axs[1,0].set_xticks([])
axs[1,0].set_yticks([])


axs[1,1].imshow(Gauss_10)
axs[1,1].set_title("Set 10")
axs[1,1].set_xticks([])
axs[1,1].set_yticks([])


axs[1,2].imshow(Gauss_15)
axs[1,2].set_title("Set 15")
axs[1,2].set_xticks([])
axs[1,2].set_yticks([])


axs[2,0].imshow(Gauss_20)
axs[2,0].set_title("Set 20")
axs[2,0].set_xticks([])
axs[2,0].set_yticks([])

axs[2,1].imshow(Gauss_25)
axs[2,1].set_title("Set 25")
axs[2,1].set_xticks([])
axs[2,1].set_yticks([])

axs[2,2].imshow(Gauss_30)
axs[2,2].set_title("Set 30")
axs[2,2].set_xticks([])
axs[2,2].set_yticks([])

axs[0, 0].axis('off')
axs[0, 0].set_visible(False)

axs[0, 2].axis('off')
axs[0, 2].set_visible(False)

# ax3.imshow(sp[0])
# ax3.set_title("Salt & Pepper")
# ax3.set_xticks([])
# ax3.set_yticks([])


# ax4.imshow(speckle[0])
# ax4.set_title("Speckle")
```

```
# ax4.set_xticks([])
# ax4.set_yticks([])

plt.suptitle('Gaussion Noise Averaging',fontsize=50)
fig.show()
```



In [6]:

```
#Salt and Pepper Noise

SaltPep_Array = np.array([np.array(im) for im in sp[:5]])

SaltPep_5= np.average(SaltPep_Array,axis=0)

SaltPep_Array = np.array([np.array(im) for im in sp[:10]])

SaltPep_10= np.average(SaltPep_Array,axis=0)

SaltPep_Array = np.array([np.array(im) for im in sp[:15]])

SaltPep_15= np.average(SaltPep_Array,axis=0)

SaltPep_Array = np.array([np.array(im) for im in sp[:20]])

SaltPep_20= np.average(SaltPep_Array,axis=0)

SaltPep_Array = np.array([np.array(im) for im in sp[:25]])

SaltPep_25= np.average(SaltPep_Array,axis=0)

SaltPep_Array = np.array([np.array(im) for im in sp[:30]])

SaltPep_30= np.average(SaltPep_Array,axis=0)

fig, axs = plt.subplots(3, 3,figsize=(20,20))

axs[0,1].imshow(sp[0])
axs[0,1].set_title("Noisy Image")
axs[0,1].set_xticks([])
axs[0,1].set_yticks([])


axs[1,0].imshow(SaltPep_5)
axs[1,0].set_title("Set 5")
axs[1,0].set_xticks([])
axs[1,0].set_yticks([])


axs[1,1].imshow(SaltPep_10)
axs[1,1].set_title("Set 10")
axs[1,1].set_xticks([])
axs[1,1].set_yticks([])


axs[1,2].imshow(SaltPep_15)
axs[1,2].set_title("Set 15")
axs[1,2].set_xticks([])
axs[1,2].set_yticks([])


axs[2,0].imshow(SaltPep_20)
axs[2,0].set_title("Set 20")
axs[2,0].set_xticks([])
axs[2,0].set_yticks([])

axs[2,1].imshow(SaltPep_25)
axs[2,1].set_title("Set 25")
axs[2,1].set_xticks([])
axs[2,1].set_yticks([])

axs[2,2].imshow(SaltPep_30)
axs[2,2].set_title("Set 30")
axs[2,2].set_xticks([])
axs[2,2].set_yticks([])

axs[0, 0].axis('off')
axs[0, 0].set_visible(False)

axs[0, 2].axis('off')
axs[0, 2].set_visible(False)

# ax3.imshow(sp[0])
# ax3.set_title("Salt & Pepper")
# ax3.set_xticks([])
# ax3.set_yticks([])


# ax4.imshow(sp[0])
# ax4.set_title("SaltPep")
# ax4.set_xticks([])
# ax4.set_yticks([])

plt.suptitle('Salt and Pepper Noise Averaging',fontsize=50)
fig.show()
```

Salt and Pepper Noise Averaging

Noisy Image

Set 5

Set 10

Set 15

Set 20

Set 25

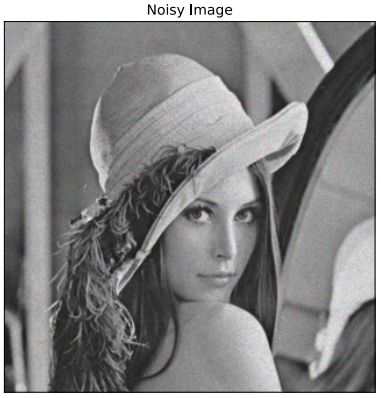
Set 30

In [7]:

```
#Speckle Noise

Speckle_Array = np.array([np.array(im) for im in speckle[:5]])

Speckle_5= np.average(Speckle_Array,axis=0)

Speckle_Array = np.array([np.array(im) for im in speckle[:10]])

Speckle_10= np.average(Speckle_Array,axis=0)

Speckle_Array = np.array([np.array(im) for im in speckle[:15]])

Speckle_15= np.average(Speckle_Array,axis=0)

Speckle_Array = np.array([np.array(im) for im in speckle[:20]])

Speckle_20= np.average(Speckle_Array,axis=0)

Speckle_Array = np.array([np.array(im) for im in speckle[:25]])

Speckle_25= np.average(Speckle_Array,axis=0)

Speckle_Array = np.array([np.array(im) for im in speckle[:30]])

Speckle_30= np.average(Speckle_Array,axis=0)

fig, axs = plt.subplots(3, 3,figsize=(20,20))

axs[0,1].imshow(speckle[0])
axs[0,1].set_title("Noisy Image")
axs[0,1].set_xticks([])
axs[0,1].set_yticks([])


axs[1,0].imshow(Speckle_5)
axs[1,0].set_title("Set 5")
axs[1,0].set_xticks([])
axs[1,0].set_yticks([])


axs[1,1].imshow(Speckle_10)
axs[1,1].set_title("Set 10")
axs[1,1].set_xticks([])
axs[1,1].set_yticks([])


axs[1,2].imshow(Speckle_15)
axs[1,2].set_title("Set 15")
axs[1,2].set_xticks([])
axs[1,2].set_yticks([])


axs[2,0].imshow(Speckle_20)
axs[2,0].set_title("Set 20")
axs[2,0].set_xticks([])
axs[2,0].set_yticks([])

axs[2,1].imshow(Speckle_25)
axs[2,1].set_title("Set 25")
axs[2,1].set_xticks([])
axs[2,1].set_yticks([])

axs[2,2].imshow(Speckle_30)
axs[2,2].set_title("Set 30")
axs[2,2].set_xticks([])
axs[2,2].set_yticks([])

axs[0, 0].axis('off')
axs[0, 0].set_visible(False)

axs[0, 2].axis('off')
axs[0, 2].set_visible(False)

# ax3.imshow(sp[0])
# ax3.set_title("Salt & Pepper")
# ax3.set_xticks([])
# ax3.set_yticks([])


# ax4.imshow(speckle[0])
# ax4.set_title("Speckle")
# ax4.set_xticks([])
# ax4.set_yticks([])

plt.suptitle('Speckle Noise Averaging',fontsize=50)
fig.show()
```

Speckle Noise Averaging

Noisy Image



Set 5

Set 10

Set 15

Set 20

Set 25

Set 30

Observation

Averaging reduces noise and improves with more image samples

Question 4

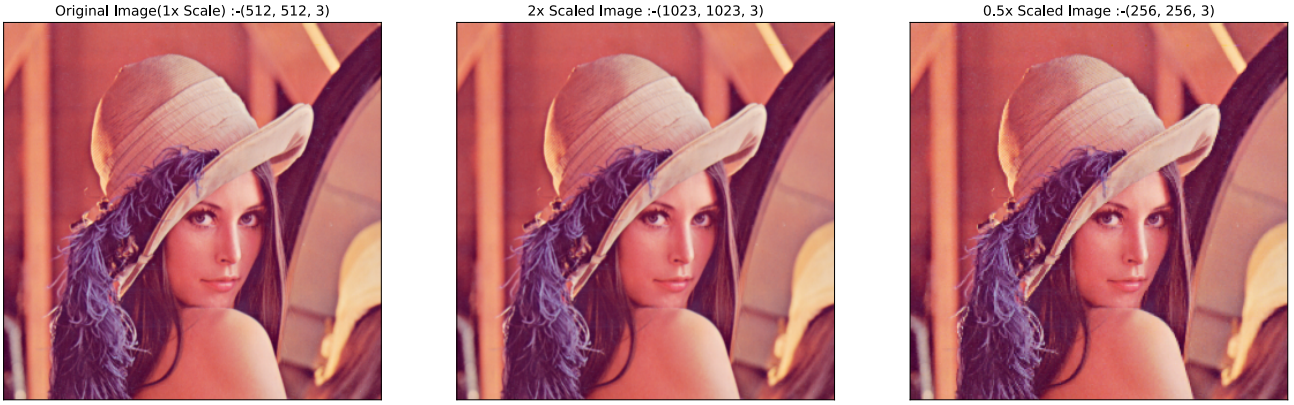
Download Lena image and scale it by factors of 1,2,0.5 using bilinear interpolation and display the scaled images. Also, display the output of built-in functions for doing scaling by factors of 0.5,2. Compare the results.

In [22]:
```
from matplotlib import pyplot as plt
import numpy as np
import cv2
from skimage.util import random_noise
import scipy

image = cv2.imread("lena512color.tiff",1)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

type(image)
```

In [49]:
```
def Blinear_Scaling(image,scale):
  scaled=np.zeros((int((image.shape[0]-1)*scale+1),int((image.shape[1]-1)*scale+1),3))
  for i in range(scaled.shape[0]):
    for j in range(scaled.shape[1]):
      x_i = i/scale; y_j = j/scale
      x0 = int (x_i); y0 = int (y_j)
      x1 = x0+1; y1 = y0+1
      if x_i==x0 and y_j==y0:
        scaled[i][j] = image[x0][y0]
      elif x_i!=x0 and y_j==y0:
        scaled[i][j] = np.round ( image[x0][y0]*(x0+1-x_i) + image[x0+1][y0]*(x_i-x0) )
      elif x_i==x0 and y_j!=y0:
        x_i = i/scale
        y_j = j/scale
        x = int (x_i)
        y = int (y_j)
        scaled[i][j] = np.round ( image[x0][y0]*(y+1-y_j) + image[x0][y0+1]*(y_j-y0) )
      else:
        scaled[i][j] = np.round ( image[x0][y0]*(x1-x_i)*(y1-y_j) + image[x0][y1]*(x1-x_i)*(y_j-y0) + image[x1][y0]*(x_i-x0)*(y1-y_j) + image[x1][y1]*(x_i-x0)*(y_j-y0) )

  scaled=scaled.astype(np.uint8)
  return scaled
```

In [51]:
```
#Create Scaled Images
Scale_2 = Blinear_Scaling(image,2)

Scale_half=Blinear_Scaling(image,0.5)

fig, axs = plt.subplots(1, 3,figsize=(20,20))

axs[0].imshow(image)
axs[0].set_title("Original Image(1x Scale) :-" + str(image.shape))
axs[0].set_xticks([])
axs[0].set_yticks([])

axs[1].imshow(Scale_2)
axs[1].set_title("2x Scaled Image :-"+ str(Scale_2.shape) )
axs[1].set_xticks([])
axs[1].set_yticks([])

axs[2].imshow(Scale_half)
axs[2].set_title("0.5x Scaled Image :-"+ str(Scale_half.shape) )
axs[2].set_xticks([])
axs[2].set_yticks([])

plt.suptitle('Billinear Interpolation Scaling',fontsize=50)
fig.show()
```

Billinear Interpolation Scaling



In [56]:

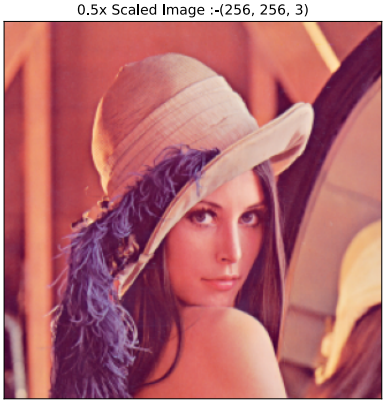
```
#Using Inbult Functions
new_height =  image.shape[0] * 2
new_width  =  image.shape[1] * 2

Scale_IN2=cv2.resize(image,(new_height,new_width),interpolation=cv2.INTER_AREA)

new_height =  int(image.shape[0] * 0.5)
new_width  =  int(image.shape[1] * 0.5)

Scale_INHalf = cv2.resize(image,(new_height,new_width),interpolation=cv2.INTER_AREA)

fig, axs = plt.subplots(1, 3,figsize=(20,20))

axs[0].imshow(image)
axs[0].set_title("Original Image(1x Scale) :-" + str(image.shape))
axs[0].set_xticks([])
axs[0].set_yticks([])

axs[1].imshow(Scale_IN2)
axs[1].set_title("2x Scaled Image :-"+ str(Scale_IN2.shape) )
axs[1].set_xticks([])
axs[1].set_yticks([])

axs[2].imshow(Scale_INHalf)
axs[2].set_title("0.5x Scaled Image :-"+ str(Scale_INHalf.shape) )
axs[2].set_xticks([])
axs[2].set_yticks([])

plt.suptitle('InBuilt Scaling',fontsize=50)
fig.show()
```

InBuilt Scaling

Original Image(1x Scale) :-(512, 512, 3)

2x Scaled Image :-(1024, 1024, 3)

0.5x Scaled Image :-(256, 256, 3)



Observation

Inbuilt Scaling is faster than Bilinear Scaling

Question 5

---

5. Download the leaning tower of PISA image and find the angle of inclination using appropriate rotations with bilinear interpolation.

In [4]:

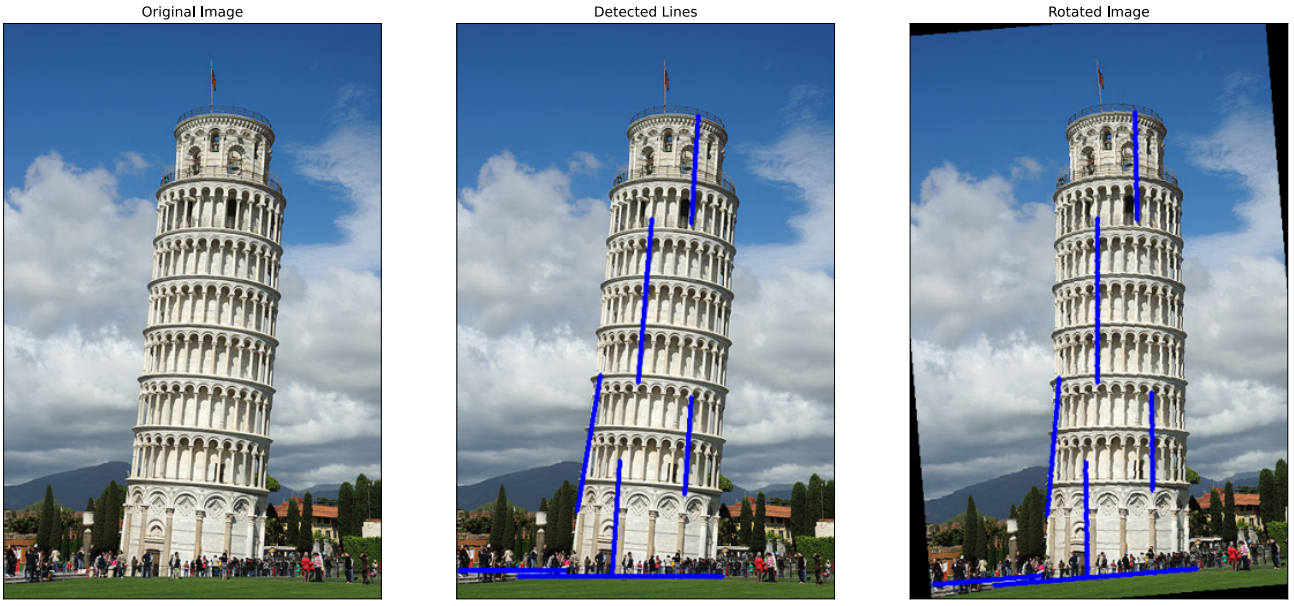
```
import numpy as np
import cv2
import math
from scipy import ndimage
from matplotlib import pyplot as plt

def rotate_image(img, degrees):

    #  convert rotation amount to radian
    rot_rad = degrees * np.pi / 180.0
    rotate_m = np.array([[np.cos(rot_rad), np.sin(rot_rad)],
                         [- np.sin(rot_rad), np.cos(rot_rad)]])

    h, w, c = img.shape

    rotated_image = np.zeros((h, w, c))

    # Rotate and shift the indices, from PICTURE to SOURCE
    indices_org = np.array(np.meshgrid(np.arange(h), np.arange(w))).reshape(2, -1)
    indices_new = indices_org.copy()
    indices_new = np.dot(rotate_m, indices_new).astype(int)   # Apply the affineWrap
    mu1 = np.mean(indices_new, axis=1).astype(int).reshape((-1, 1))
    mu2 = np.mean(indices_org, axis=1).astype(int).reshape((-1, 1))
    indices_new += (mu2-mu1)   # Shift the image back to the center

    # Remove the pixels in the rotated image, that are now out of the bounds of the result image
    t0, t1 = indices_new
    t0 = (0 <= t0) & (t0 < h)
    t1 = (0 <= t1) & (t1 < w)
    valid = t0 & t1
    indices_new = indices_new.T[valid].T
    indices_org = indices_org.T[valid].T

    xind, yind = indices_new
    xi, yi = indices_org
    rotated_image[xi, yi, :] = img[xind, yind, :]

    return rotated_image.astype(np.uint8)
```

In [6]:

```
img_before = cv2.imread('rotate_me.jpeg')


img_gray = cv2.cvtColor(img_before, cv2.COLOR_BGR2GRAY)
img_edges = cv2.Canny(img_gray, 100, 100, apertureSize=3)
```

```python
lines = cv2.HoughLinesP(img_edges, 1, math.pi / 180.0, 100, minLineLength=100, maxLineGap=5)

angles = []

for [[x1, y1, x2, y2]] in lines:
    cv2.line(img_before, (x1, y1), (x2, y2), (255, 0, 0), 3)
    angle = math.degrees(math.atan2(y2 - y1, x2 - x1))
    angles.append(angle)

median_angle = np.median(angles)
# img_rotated = ndimage.rotate(img_before, median_angle) #Testing with inbuilt function
img_rotated = rotate_image(img_before, median_angle + 90) #Using our own rotate function

# print(f"Angle is {median_angle:.04f}")
# cv2.imwrite('rotated.jpg', img_rotated)

fig, axs = plt.subplots(1, 3,figsize=(20,20))

img= cv2.imread('rotate_me.jpeg')

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img_before = cv2.cvtColor(img_before, cv2.COLOR_BGR2RGB)
img_rotated = cv2.cvtColor(img_rotated, cv2.COLOR_BGR2RGB)

axs[0].imshow(img)
axs[0].set_title("Original Image")
axs[0].set_xticks([])
axs[0].set_yticks([])

axs[1].imshow(img_before)
axs[1].set_title("Detected Lines")
axs[1].set_xticks([])
axs[1].set_yticks([])

axs[2].imshow(img_rotated)
axs[2].set_title("Rotated Image")
axs[2].set_xticks([])
axs[2].set_yticks([])

plt.suptitle('Image Rotation \n ANGLE OF INCLINATION = '+ str(abs(median_angle)) ,fontsize=20)
fig.show()
```

Image Rotation
ANGLE OF INCLINATION = 85.29214775562778



Question 6

6. Do histogram equalization on pout-dark and display the same

In [62]:

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

img = cv2.imread('pout-dark.jpg',0)


equ = cv2.equalizeHist(img)

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
equ = cv2.cvtColor(equ, cv2.COLOR_BGR2RGB)

fig, axs = plt.subplots(1, 2,figsize=(20,20))

axs[0].imshow(img)
axs[0].set_title("Original Image")
axs[0].set_xticks([])
axs[0].set_yticks([])

axs[1].imshow(equ)
axs[1].set_title("Equalized Image")
axs[1].set_xticks([])
axs[1].set_yticks([])

plt.suptitle('Histogram Equalisation',fontsize=20)
fig.show()
```

Histogram Equalisation

Question 7

7. Do histogram matching(specification) on the pout-dark image, keeping pout-bright as a reference image.

In [68]:

```
# Python 2/3 compatibility
from __future__ import print_function

import cv2 # Import the OpenCV library
import numpy as np # Import Numpy library
import matplotlib.pyplot as plt # Import matplotlib functionality
import sys # Enables the passing of arguments

# Define the file name of the images
SOURCE_IMAGE = "pout-dark.jpg"
REFERENCE_IMAGE = "pout-bright.jpg"

OUTPUT_IMAGE = "result"

def calculate_cdf(histogram):

    # Get the cumulative sum of the elements
    cdf = histogram.cumsum()
    normalized_cdf = cdf / float(cdf.max())

    return normalized_cdf

def calculate_lookup(src_cdf, ref_cdf):
    lookup_table = np.zeros(256)
    lookup_val = 0
    for src_pixel_val in range(len(src_cdf)):
        lookup_val
        for ref_pixel_val in range(len(ref_cdf)):
            if ref_cdf[ref_pixel_val] >= src_cdf[src_pixel_val]:
                lookup_val = ref_pixel_val
                break
        lookup_table[src_pixel_val] = lookup_val
    return lookup_table

def match_histograms(src_image, ref_image):
    # Split the images into the different color channels
    # b means blue, g means green and r means red
    src_b, src_g, src_r = cv2.split(src_image)
    ref_b, ref_g, ref_r = cv2.split(ref_image)

    # Compute the b, g, and r histograms separately
    # The flatten() Numpy method returns a copy of the array c
    # collapsed into one dimension.
    src_hist_blue, bin_0 = np.histogram(src_b.flatten(), 256, [0,256])
    src_hist_green, bin_1 = np.histogram(src_g.flatten(), 256, [0,256])
    src_hist_red, bin_2 = np.histogram(src_r.flatten(), 256, [0,256])
    ref_hist_blue, bin_3 = np.histogram(ref_b.flatten(), 256, [0,256])
    ref_hist_green, bin_4 = np.histogram(ref_g.flatten(), 256, [0,256])
    ref_hist_red, bin_5 = np.histogram(ref_r.flatten(), 256, [0,256])

    # Compute the normalized cdf for the source and reference image
    src_cdf_blue = calculate_cdf(src_hist_blue)
    src_cdf_green = calculate_cdf(src_hist_green)
    src_cdf_red = calculate_cdf(src_hist_red)
    ref_cdf_blue = calculate_cdf(ref_hist_blue)
    ref_cdf_green = calculate_cdf(ref_hist_green)
    ref_cdf_red = calculate_cdf(ref_hist_red)

    # Make a separate lookup table for each color
    blue_lookup_table = calculate_lookup(src_cdf_blue, ref_cdf_blue)
    green_lookup_table = calculate_lookup(src_cdf_green, ref_cdf_green)
    red_lookup_table = calculate_lookup(src_cdf_red, ref_cdf_red)

    # Use the lookup function to transform the colors of the original
    # source image
    blue_after_transform = cv2.LUT(src_b, blue_lookup_table)
    green_after_transform = cv2.LUT(src_g, green_lookup_table)
    red_after_transform = cv2.LUT(src_r, red_lookup_table)

    # Put the image back together
    image_after_matching = cv2.merge([
        blue_after_transform, green_after_transform, red_after_transform])
    image_after_matching = cv2.convertScaleAbs(image_after_matching)

    return image_after_matching


# Pull system arguments
try:
    image_src_name = 'pout-dark.jpg'
    image_ref_name = 'pout-bright.jpg'
except:
    image_src_name = SOURCE_IMAGE
    image_ref_name = REFERENCE_IMAGE

# Load the images and store them into a variable
image_src = cv2.imread(cv2.samples.findFile(image_src_name))
image_ref = cv2.imread(cv2.samples.findFile(image_ref_name))

# Check if the images loaded properly
if image_src is None:
    print('Failed to load source image file:', image_src_name)
    sys.exit(1)
elif image_ref is None:
    print('Failed to load reference image file:', image_ref_name)
    sys.exit(1)
else:
    # Do nothing
    pass

output_image = match_histograms(image_src, image_ref)

#cv2.imwrite(OUTPUT_IMAGE, output_image)

## Display images, used for debugging
# cv2.imshow('Source Image', image_src)
# cv2.imshow('Reference Image', image_ref)
# cv2.imshow('Output Image', output_image)
# cv2.waitKey(0) # Wait for a keyboard event
```

```
# if __name__ == '__main__':
#     print(__doc__)
#     main()
#     cv2.destroyAllWindows()


fig, axs = plt.subplots(1, 3,figsize=(20,20))

axs[0].imshow(image_src)
axs[0].set_title("Image Source")
axs[0].set_xticks([])
axs[0].set_yticks([])


axs[1].imshow(image_ref)
axs[1].set_title("Reference Image")
axs[1].set_xticks([])
axs[1].set_yticks([])

axs[2].imshow(output_image)
axs[2].set_title("Ouput Image")
axs[2].set_xticks([])
axs[2].set_yticks([])

plt.suptitle('Histogram Matching',fontsize=50)
fig.show()
```

Histogram Matching