Home Assignment 4. Poisson distribution

Probability and Statistics, Spring 2019

30.4.2019, Salla Vesterinen
Helsinki Metropolia University of Applied Sciences

In [1]:
```
# Import necessary libraries and functions
%pylab inline
import numpy.random as rnd
from scipy.misc import factorial
```

```
Populating the interactive namespace from numpy and matplotlib
```

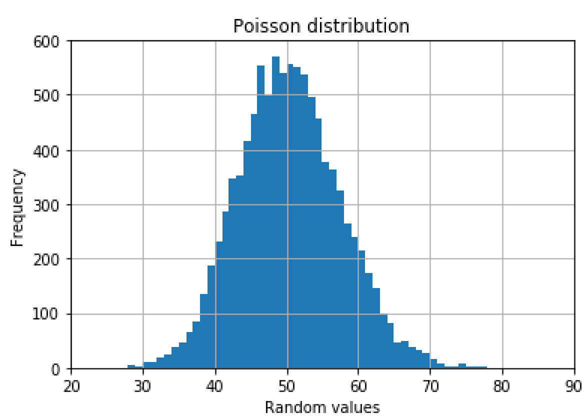
Problem 1

1. Generate 10,000 random numbers from *Poission distribution* having mean value of 50. Make a histogram of the values. Pay special attention to the bins parameter.
2. Calculate the mean and standard deviation values for the generated random numbers.
3. Generate 10,000 random numbers from *normal distribution* having the same mean and standard deviation as for the Poisson distribution.
4. Overlap the histograms of the generated data (Poisson and normal distribution).
5. How much do they differ? Explain why.

1. Random numbers from Poisson distribution

In [2]:
```
# Generate random numbers from Poisson distribution
lm = 50
N = 10000
# Generate random data
x = rnd.poisson(lm, N)

# Show the frequency histogram
bins = np.arange(20, 90)
p = plt.hist(x, bins)
plt.xlabel('Random values')
plt.ylabel('Frequency')
plt.title('Poisson distribution')
xl = plt.xlim(20, 90)
plt.grid()

print("Mean:",np.mean(x))
print("Std:",np.std(x))
```

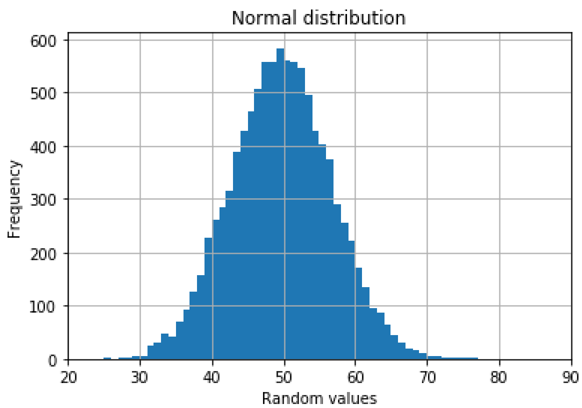
```
Mean: 49.9409
Std: 7.084744680650108
```



1. Random numbers from Normal distribution

In [3]:
```
# Generate random data
x = numpy.random.normal(50, 7, 10000)

# Show the frequency histogram
bins = np.arange(20, 90)
p = plt.hist(x, bins)
title('Normal distribution')
plt.xlabel('Random values')
plt.ylabel('Frequency')
xl = plt.xlim(20, 90)
plt.grid()

print("Mean:",np.mean(x))
print("Std:",np.std(x))
```

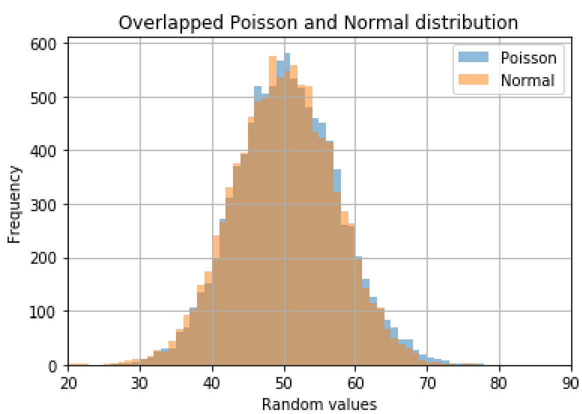
```
Mean: 49.70238599782331
Std: 6.948906933203151
```



1. Overlap Poisson and normal distributions

In [4]:
```
# Generate random numbers from Poisson distribution
x = rnd.poisson(50, 10000)

# Show the frequency histogram
bins = np.arange(20, 90)
p = plt.hist(x, bins, alpha=0.5, label='Poisson')

print("Poisson Mean:",np.mean(x))
print("Poisson Std:",np.std(x))


# Generate random data
x = numpy.random.normal(np.mean(x), np.std(x), 10000)

# Show the frequency histogram
bins = np.arange(20, 90)
p = plt.hist(x, bins, alpha=0.5, label='Normal')
title('Overlapped Poisson and Normal distribution')
plt.xlabel('Random values')
plt.ylabel('Frequency')
xl = plt.xlim(20, 90)
plt.grid()
plt.legend()

print("Normal dist. Mean:",np.mean(x))
print("Normal dist. Std:",np.std(x))
```

```
Poisson Mean: 50.0809
Poisson Std: 7.0986868637798075
Normal dist. Mean: 50.13135371457986
Normal dist. Std: 7.070245909306651
```



1. How much do they differ? Explain why.
   Poisson distribution is discrete, normal distribution is continuous
   In this case though since the mean is so big they are very similar to each other

Problem 2

In this problem the aim is to simulate the functioning of a desktop image scanner (see Lab 4 -> Image pattern simulator). The scanner has a maximum resolution of 600 DPI (dots-per-inch) and its width is 21 cm and height is 30 cm (roughly the size of A4). The color-resolution of the scanner is 14 bits.

Imagine that you are scanning 10 A4-sized black-and-white documents having different average brightness. The brightness for each document is [0.001, 0.002, 0.005, 0.01, 0.02, 0.03, 0.10, 0.20, 0.50, 0.99] of the full-scale color-resolution (14-bits = 2^14).

Write a code that simulates the scanning of the documents having different brightness. calculate the mean and the standard deviation values of the brightness values for each document.

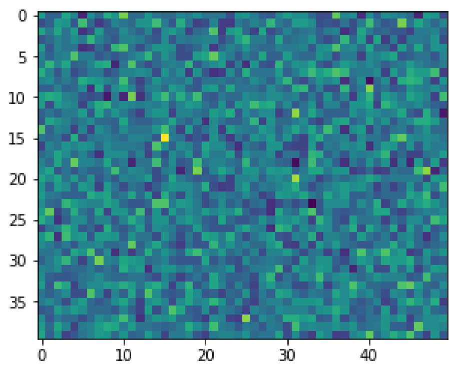
Plot how the signal-to-noise ratio changes when the brightness increases.

In [334]:
```
#brightnesses
b=np.array([0.001, 0.002, 0.005, 0.01, 0.02, 0.03, 0.1, 0.2, 0.5, 0.99])
print(b)
b_int=(b*2**14).astype(int)
print(b_int)
```

```
[0.001 0.002 0.005 0.01  0.02  0.03  0.1   0.2   0.5   0.99 ]
[   16    32    81   163   327   491  1638  3276  8192 16220]
```

In [352]:
```
W=40
H=50
S1=np.zeros((W,H))
size(S1)
for n in range(W*H*16):
    w=np.random.randint(W)
    h=np.random.randint(H)
    S1[w,h]+=1
imshow(S1)
```

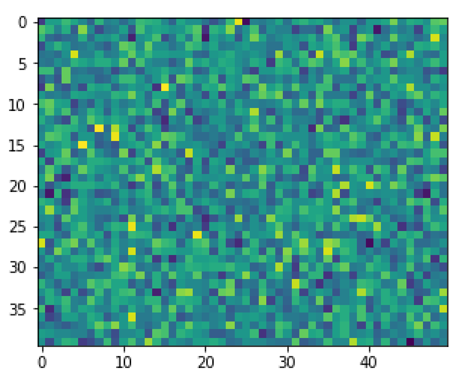
Out[352]: <matplotlib.image.AxesImage at 0x1e0112de080>



In [ ]:
```
hist(S[:].flatten());
```

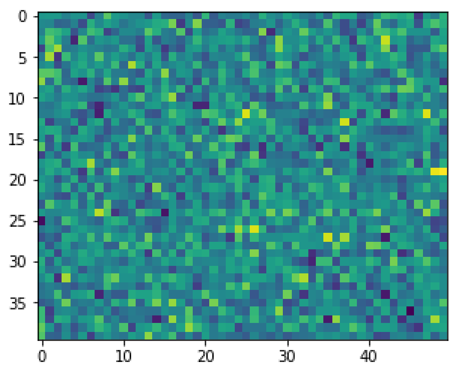
In [353]:
```
W=40
H=50
S2=np.zeros((W,H))
size(S2)
for n in range(W*H*32):
    w=np.random.randint(W)
    h=np.random.randint(H)
    S2[w,h]+=1
imshow(S2)
```

Out[353]: <matplotlib.image.AxesImage at 0x1e00af04e48>



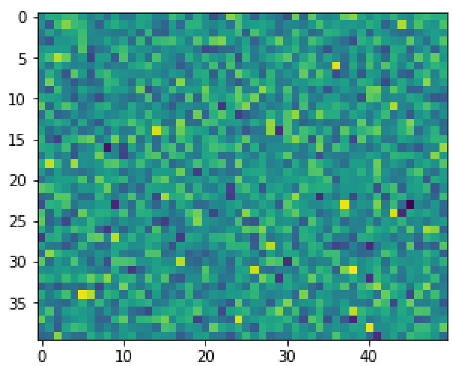
In [354]:
```
W=40
H=50
S3=np.zeros((W,H))
size(S3)
for n in range(W*H*81):
    w=np.random.randint(W)
    h=np.random.randint(H)
    S3[w,h]+=1
imshow(S3)
```

Out[354]: `<matplotlib.image.AxesImage at 0x1e00a6f6fd0>`



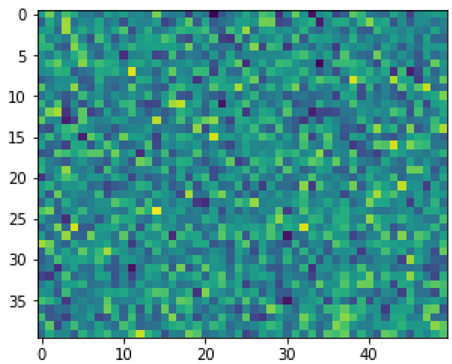
In [355]:
```
W=40
H=50
S4=np.zeros((W,H))
size(S4)
for n in range(W*H*163):
    w=np.random.randint(W)
    h=np.random.randint(H)
    S4[w,h]+=1
imshow(S4)
```

Out[355]: `<matplotlib.image.AxesImage at 0x1e0051b11d0>`



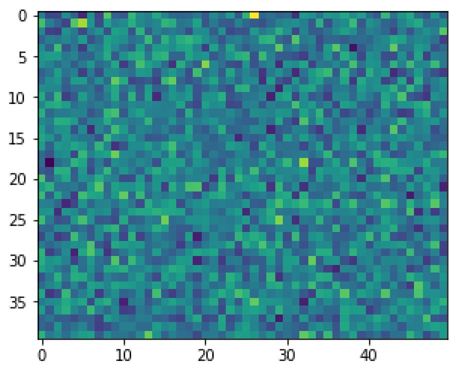
In [356]:
```
W=40
H=50
S5=np.zeros((W,H))
size(S5)
for n in range(W*H*327):
    w=np.random.randint(W)
    h=np.random.randint(H)
    S5[w,h]+=1
imshow(S5)
```

Out[356]: <matplotlib.image.AxesImage at 0x1e011ad54a8>



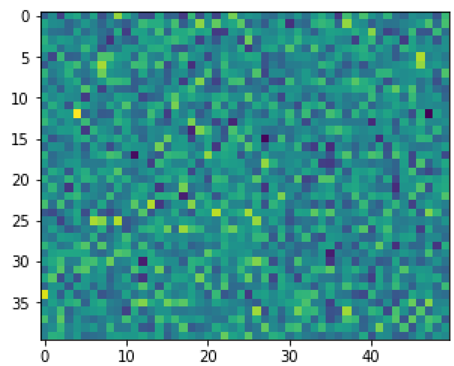
In [357]:
```
W=40
H=50
S6=np.zeros((W,H))
size(S6)
for n in range(W*H*491):
    w=np.random.randint(W)
    h=np.random.randint(H)
    S6[w,h]+=1
imshow(S6)
```

Out[357]: <matplotlib.image.AxesImage at 0x1e0050ae1d0>



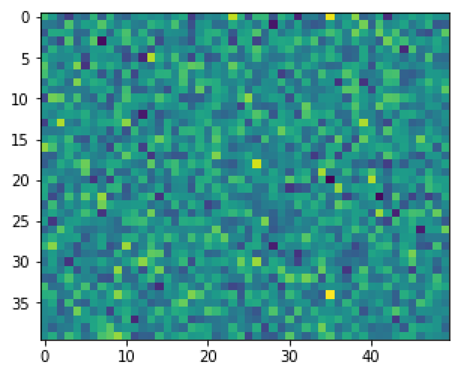
In [358]:
```
W=40
H=50
S7=np.zeros((W,H))
size(S7)
for n in range(W*H*1638):
    w=np.random.randint(W)
    h=np.random.randint(H)
    S7[w,h]+=1
imshow(S7)
```

Out[358]: <matplotlib.image.AxesImage at 0x1e0134f1390>



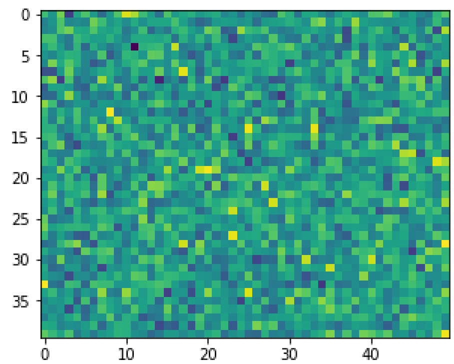
In [359]:
```
W=40
H=50
S8=np.zeros((W,H))
size(S8)
for n in range(W*H*3276):
    w=np.random.randint(W)
    h=np.random.randint(H)
    S8[w,h]+=1
imshow(S8)
```

Out[359]: <matplotlib.image.AxesImage at 0x1e00b09b400>



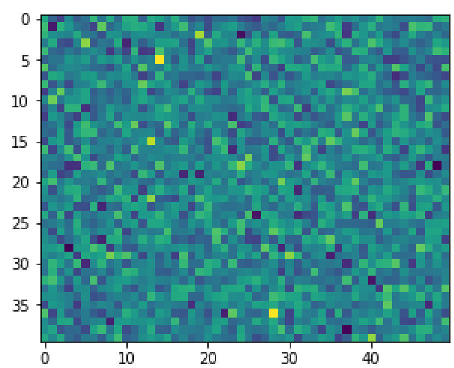
In [366]:
```
W=40
H=50
S9=np.zeros((W,H))
size(S9)
for n in range(W*H*8192):
    w=np.random.randint(W)
    h=np.random.randint(H)
    S9[w,h]+=1
imshow(S9)
```

Out[366]: `<matplotlib.image.AxesImage at 0x1e010eb6780>`



In [367]:
```
W=40
H=50
S10=np.zeros((W,H))
size(S10)
for n in range(W*H*16220):
    w=np.random.randint(W)
    h=np.random.randint(H)
    S10[w,h]+=1
imshow(S10)
```

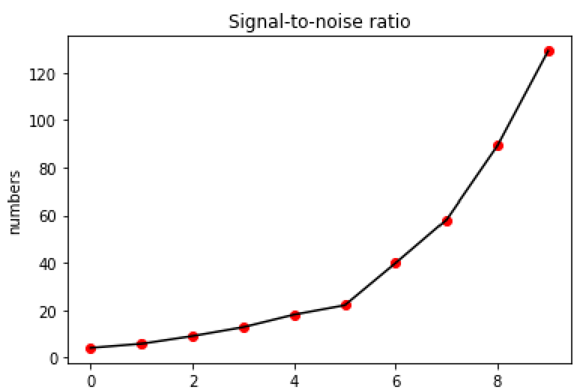
Out[367]: `<matplotlib.image.AxesImage at 0x1e0053e57f0>`



In [368]:
```
r1=np.mean(S1)/np.std(S1)
print("SNR1:",r1)
r2=np.mean(S2)/np.std(S2)
print("SNR2:",r2)
r3=np.mean(S3)/np.std(S3)
print("SNR3:",r3)
r4=np.mean(S4)/np.std(S4)
print("SNR4:",r4)
r5=np.mean(S5)/np.std(S5)
print("SNR5:",r5)
r6=np.mean(S6)/np.std(S6)
print("SNR6:",r6)
r7=np.mean(S7)/np.std(S7)
print("SNR7:",r7)
r8=np.mean(S8)/np.std(S8)
print("SNR8:",r8)
r9=np.mean(S9)/np.std(S9)
print("SNR9:",r9)
r10=np.mean(S10)/np.std(S10)
print("SNR10:",r10)
```

```
SNR1: 4.121713172586386
SNR2: 5.872399949013806
SNR3: 9.120141319498662
SNR4: 12.868077967868787
SNR5: 18.149393750333733
SNR6: 22.117700539524613
SNR7: 39.88753571575536
SNR8: 58.171762173808716
SNR9: 89.21927555216618
SNR10: 129.20934814063565
```

In [369]:
```
import matplotlib.pyplot as plt
r=[r1,r2,r3,r4,r5,r6,r7,r8,r9,r10]
plt.plot(r, 'ro', r, 'k')
plt.ylabel('numbers')
title('Signal-to-noise ratio')
plt.show()
```

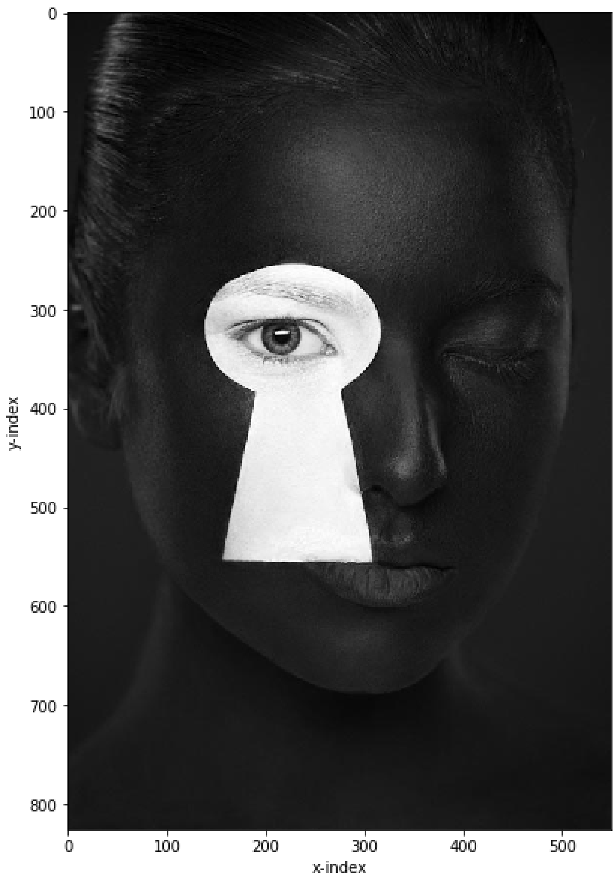


Problem 3

Find or take a photo having smooth surfaces having different lightning conditions. You could search for example [black-and-white portrait photography (https://duckduckgo.com/?q=black+and+white+portrait+photography&t=ffab&atb=v150-1&iax=images&ia=images)](https://duckduckgo.com/?q=black+and+white+portrait+photography&t=ffab&atb=v150-1&iax=images&ia=images) or [black-and-white scenery photography (https://duckduckgo.com/?q=black+and+white+scenery+photograph&t=ffab&atb=v150-1&iar=images&iax=images&ia=images)](https://duckduckgo.com/?q=black+and+white+scenery+photograph&t=ffab&atb=v150-1&iar=images&iax=images&ia=images).

Select parts of the images where the brightness have different intensities and calculate the mean, the standard devation and the SNR-values for those parts of the images. Study does the SNR vs. mean intensity value rule work also here.

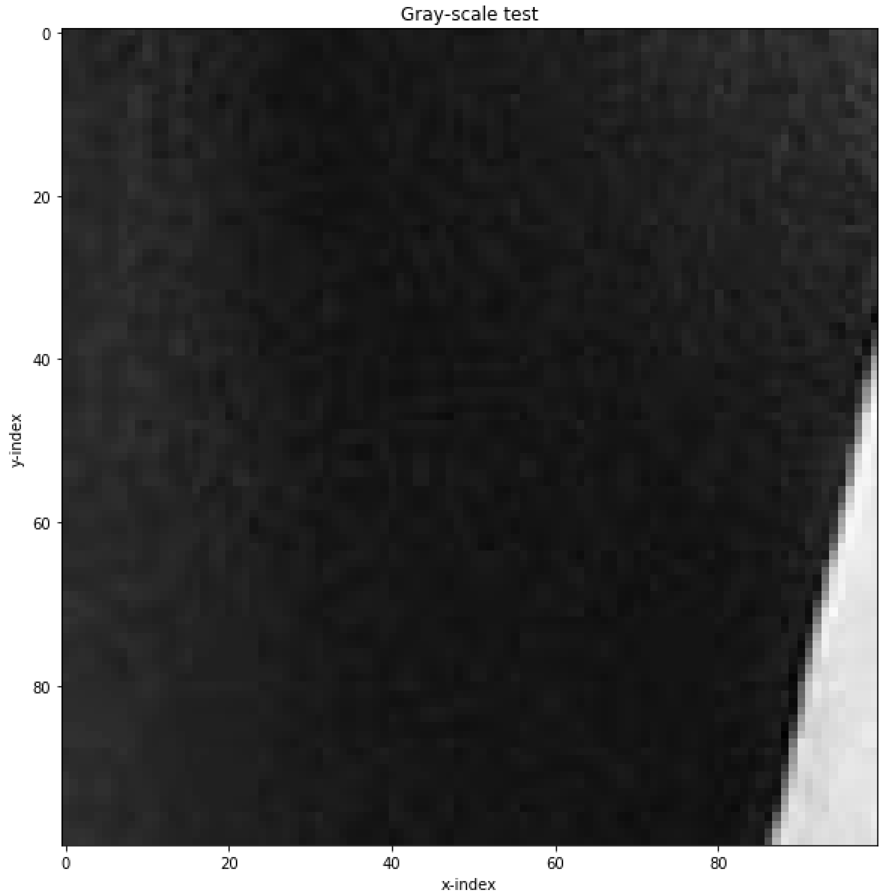
In [199]:
```
import skimage.io as skio

# The url address to find the image
fpath = 'https://weandthecolor.com/wp-content/uploads/2012/08/'

# Filename for ISO 3200 image
fname = 'Weird-Beauty-Photographic-Black-and-White-Portrait-63253.jpg'

# Read the image
image = skio.imread(fpath + fname)

# Show it in large figure
plt.figure(figsize=(10,10))
plt.imshow(image)
plt.xlabel('x-index')
plt.ylabel('y-index');
```

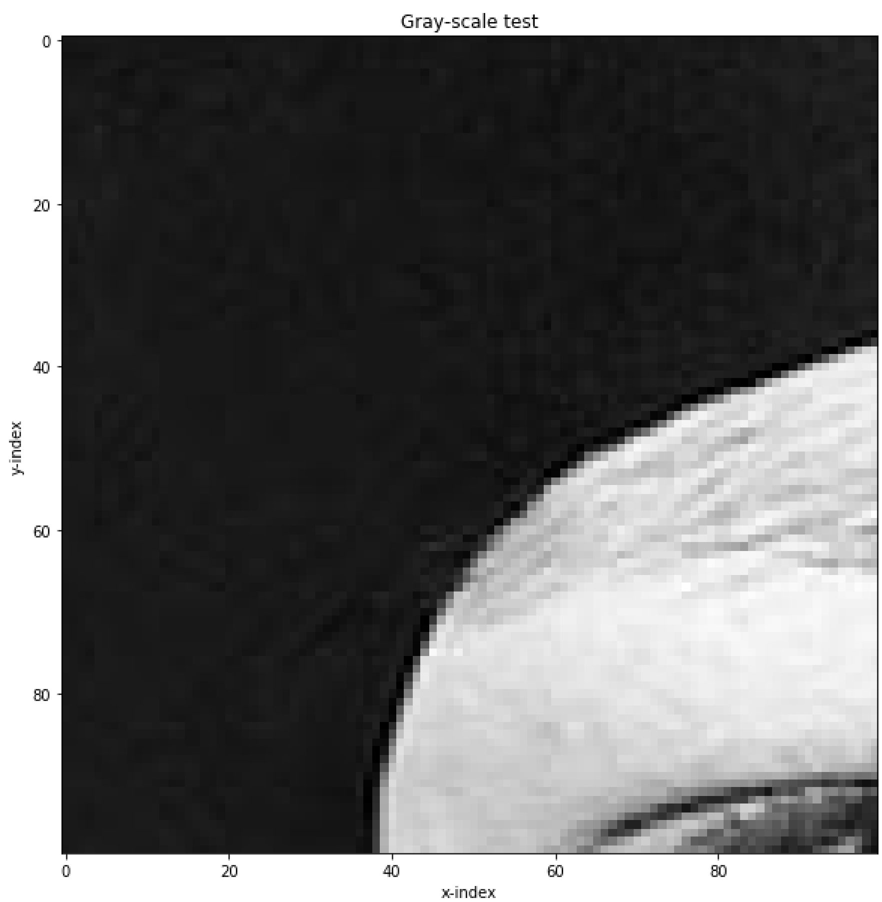


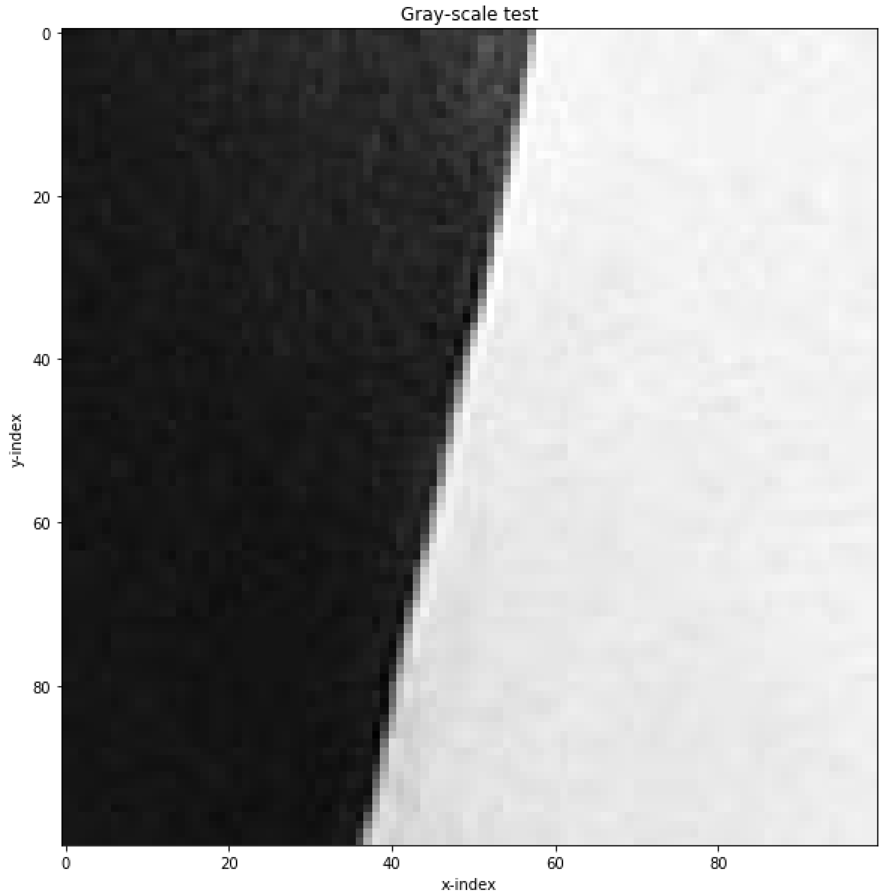
In [87]:
```
# What are the shape and dimensions of the 1-channel gray-color image?
print('shape = %s dimensions = %s' % (grayimage.shape, grayimage.ndim))
```

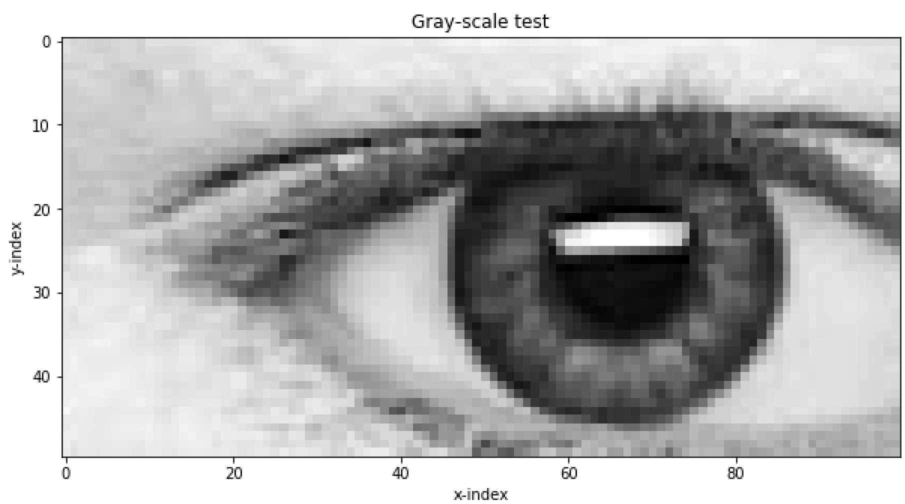
```
shape = (826, 550) dimensions = 2
```

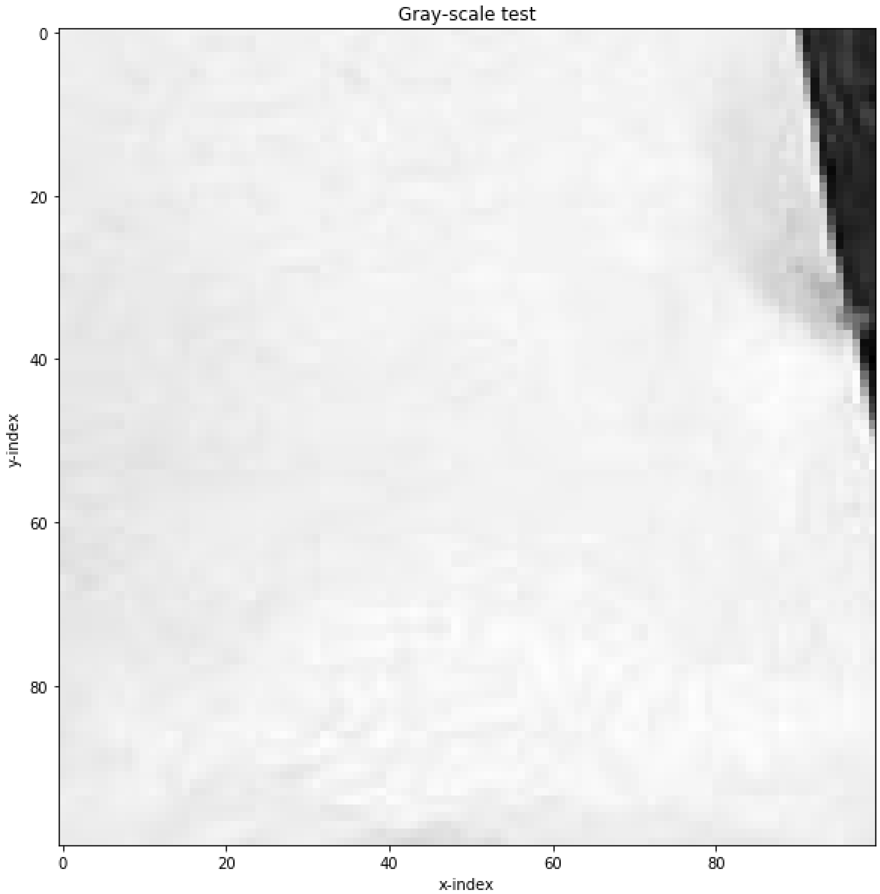
In [179]:
```
# Select the part from the image where the gray-scale test strips locate
image_part1 = grayimage[400:500, 80:180]
image_part2 = grayimage[220:320, 100:200]
image_part3 = grayimage[400:500, 130:230]
image_part4 = grayimage[300:350, 150:250]
image_part5 = grayimage[450:550, 200:300]


# Show the tests
plt.figure(figsize=(10,10))
plt.imshow(image_part1, cmap='gray', interpolation='none', aspect='equal')
plt.title('Gray-scale test')
plt.xlabel('x-index')
plt.ylabel('y-index');

plt.figure(figsize=(10,10))
plt.imshow(image_part2, cmap='gray', interpolation='none', aspect='equal')
plt.title('Gray-scale test')
plt.xlabel('x-index')
plt.ylabel('y-index');

plt.figure(figsize=(10,10))
plt.imshow(image_part3, cmap='gray', interpolation='none', aspect='equal')
plt.title('Gray-scale test')
plt.xlabel('x-index')
plt.ylabel('y-index');

plt.figure(figsize=(10,10))
plt.imshow(image_part4, cmap='gray', interpolation='none', aspect='equal')
plt.title('Gray-scale test')
plt.xlabel('x-index')
plt.ylabel('y-index');

plt.figure(figsize=(10,10))
plt.imshow(image_part5, cmap='gray', interpolation='none', aspect='equal')
plt.title('Gray-scale test')
plt.xlabel('x-index')
plt.ylabel('y-index');
```


Gray-scale test
y-index
x-index
0
20
40
60
80

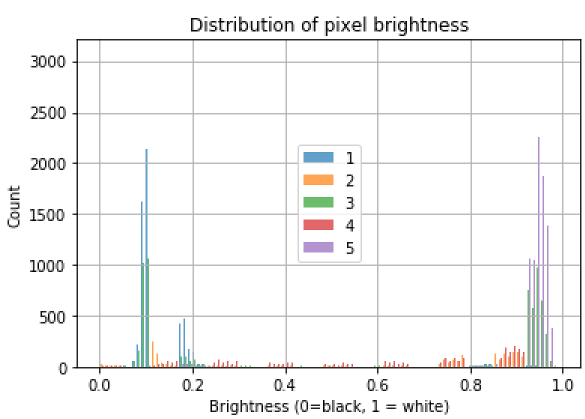

Gray-scale test
y-index
x-index
0
20
40
60
80


Gray-scale test
y-index
x-index
0
20
40
60
80


Gray-scale test
y-index
x-index
0
10
20
30
40
0
20
40
60
80



In [180]:
```
# First flatten the 2D matrix of pixels in a row
data1 = image_part1.flatten()
data2 = image_part2.flatten()
data3 = image_part3.flatten()
data4 = image_part4.flatten()
data5 = image_part5.flatten()

# Make a histogram, use custom bins
plt.figure()

plt.hist([data1,data2,data3,data4,data5], bins=np.arange(0, 1, 0.01), alpha=0.7, label=['1','2', '3', '4', '5'
])

plt.xlabel('Brightness (0=black, 1 = white)')
plt.ylabel('Count')
plt.title('Distribution of pixel brightness');
plt.legend(loc='center')
plt.grid()
```

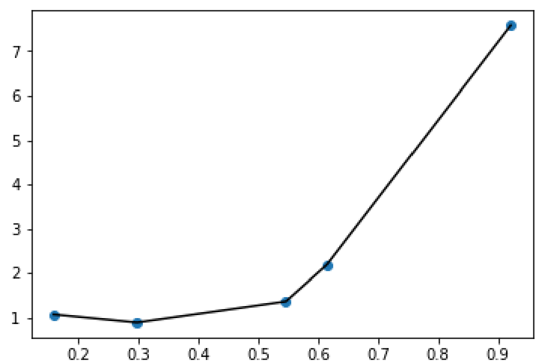


In [181]:
```
mn1=np.mean(data1)
snr1=np.mean(data1)/np.std(data1)
print("1:", mn1, snr1)

mn2=np.mean(data2)
snr2=np.mean(data2)/np.std(data2)
print("2:", mn2, snr2)

mn3=np.mean(data3)
snr3=np.mean(data3)/np.std(data3)
print("3:", mn3, snr3)

mn4=np.mean(data4)
snr4=np.mean(data4)/np.std(data4)
print("4:", mn4, snr4)

mn5=np.mean(data5)
snr5=np.mean(data5)/np.std(data5)
print("5:", mn5, snr5)
```

```
1: 0.15815215686274509 1.0714866436658603
2: 0.29716627450980393 0.891001897162556
3: 0.5460407843137255 1.3637325056656262
4: 0.6146619607843138 2.188846676424593
5: 0.921483137254902 7.582207765289462
```

In [222]:
```
means=[mn1, mn2, mn3, mn4, mn5]
snrs=[snr1, snr2, snr3, snr4, snr5]
plot(means, snrs, 'o', means, snrs, 'k')
```

Out[222]:
```
[<matplotlib.lines.Line2D at 0x1e011d5c1d0>,
 <matplotlib.lines.Line2D at 0x1e011d5c2e8>]
```



The relationship between the SNR and mean intensity value makes the function look like an exponential function