OBLIG 1

Alle importeringene jeg trenger til oblig 1

In [2]:
```
from scipy import ndimage
from skimage import io,util,color
from IPython import display
import numpy
import sys
import math

import ipywidgets as widgets
from ipywidgets import interactive
from IPython.display import display, clear_output

import matplotlib.pyplot as plt

%matplotlib inline
```

Oppgave 1

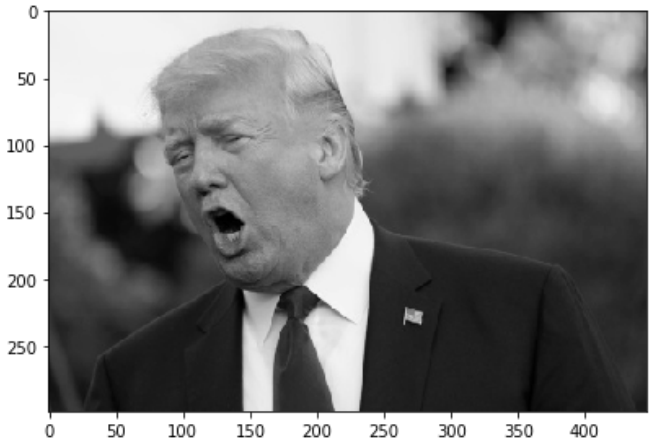
Dette er koden for konvolusjons metodene

In [93]:
```
def isOutside(x,y,width,height):
    if(x < 0 or y < 0):
        return True
    if(x >= width or y >= height):
        return True
    return False

def convolve(filtr, image, mode, value = 0):
    width, height = image.shape
    Fwidth, Fheight = filtr.shape

    Mx = math.ceil(Fwidth/2)
    My = math.ceil(Fheight/2)

    ret = numpy.zeros(image.shape, dtype=image.dtype)

    for x in range(width):
        for y in range(height):
            s = 0

            for x2 in range(Fwidth):
                for y2 in range(Fheight):
                    imX = x - (x2 + 1 - Mx)
                    imY = y - (y2 + 1 - My)

                    if(isOutside(imX, imY, width, height)):
                        outValue = 0

                        if(mode == 'constant'):
                            outValue = value
                        elif(mode == 'wrap'):
                            outValue = image[imX % width, imY % height]
                        else:
                            outValue = 0

                        s += outValue * filtr[x2,y2]
                    else:
                        s += image[imX, imY] * filtr[x2,y2]

            ret[x,y] = s

    return ret
```

Her kan du velge hvilket bilde som skal bli prosessert

In [42]:
```
def readAndShowImage(image):
    im = io.imread(image, as_gray=True)
    io.imshow(im)
    io.show()

inputImage = widgets.Dropdown(
    options={
        'Minecraft':'image.png',
        'Trump':'trump1.jpg'
    },
    description='Bilde: ',
    disabled=False,
)

interactive(readAndShowImage, image=inputImage)
```

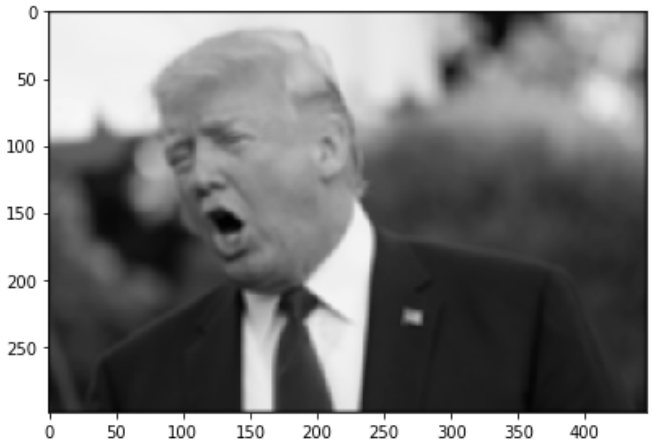
Bilde: Trump



Her kjører jeg en 5*5 box blur på bildet som ble valgt ved hjelp av metoden jeg lagde

In [43]:
```
def update(image):
    im = util.img_as_ubyte(color.rgb2gray(io.imread(image)))
    filtr = numpy.ones((5,5)) / (5*5)

    image = convolve(filtr, im, 'wrap', 0.0)
    io.imshow(image, cmap='gray')
    io.show()


interactive(update, image = inputImage)
```

Bilde: Trump

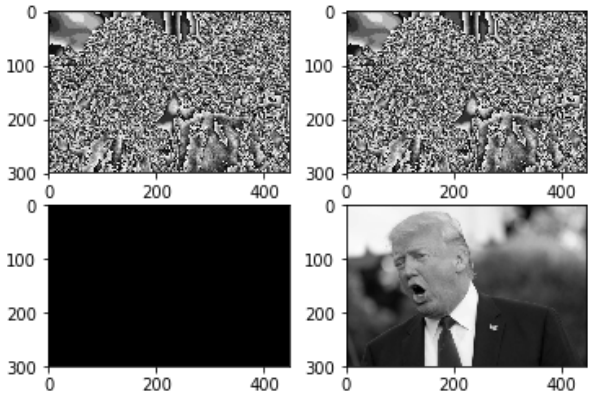


For å dokumentere at filtreringen fungerer har jeg valgt å kjøre samme filter på samme bildet ved bruk av min egenskrevene algoritme, og en annen ved hjelp av den innebygde.Da kan jeg se på differansen mellom de to bildene for å sjekke at differansen blir 0 for hver pixel.

Jeg velger også å bruke ett filter som ikke er symmetrisk for å dobbelsjekke at jeg går gjennom filteret i riktig rekkefølge.

In [97]:
```
def updateTest(image):
    im = util.img_as_ubyte(color.rgb2gray(io.imread(image)))
    filtr = numpy.array([[1,2,3],[4,5,6],[7,8,9]])

    mine = convolve(filtr, im, 'wrap', 0.0)

    builtin = ndimage.convolve(im,filtr,mode='wrap', cval=0)

    diff = mine - builtin

    io.imshow_collection([mine,builtin,diff,im], cmap='gray')
    io.show()


interactive(updateTest, image = inputImage)
```

Bilde: Trump



Her er enda en test ved hjelp av ett 3x3 bilde og ett 3x3 filter, jeg gjør som i forrige test konvolusjon ved hjelp av min og den innbebygde og ser da at jeg får det samme fra begge.

In [94]:
```
def updateTest2():
    im = numpy.array([[10,20,30],[40,50,60],[70,80,90]])
    filtr = numpy.array([[1,2,3],[4,5,6],[7,8,9]])

    mine = convolve(filtr, im, 'wrap', 1)

    builtin = ndimage.convolve(im,filtr,mode='wrap', cval=1)

    print(mine)
    print(builtin)
    io.show()


updateTest2()
```

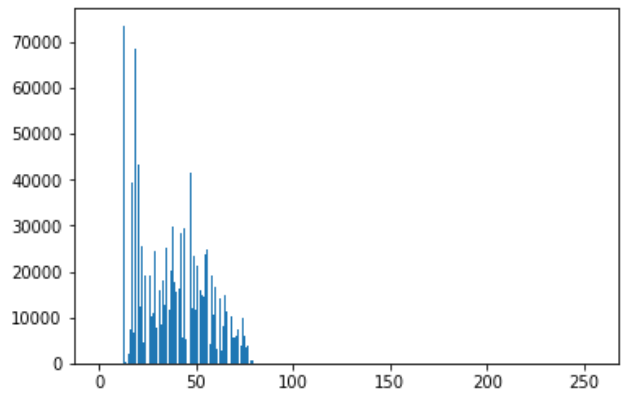
```
[[2550 2460 2550]
 [1740 1650 1740]
 [2550 2460 2550]]
[[2550 2460 2550]
 [1740 1650 1740]
 [2550 2460 2550]]
```

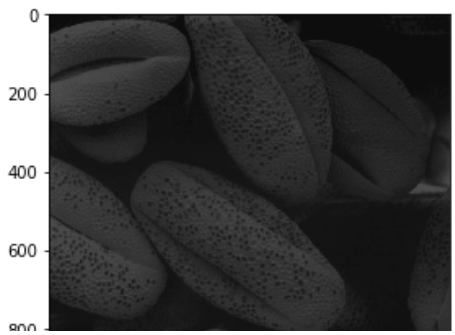
Oppgave 2

Her kan du velge hvilket bilde som skal bli prossessert og se ett histogram over bildet

In [44]:
```
def histogram(im):
    ret = numpy.zeros(256, dtype=numpy.uint)

    height, width = im.shape

    for x in range(height):
        for y in range (width):
            ret[im[x][y]] += 1

    return ret

def printHistogramBilde(image):
    hist = histogram(image)
    ind = numpy.arange(len(hist))

    fig, ax = plt.subplots()
    rects1 = ax.bar(ind, hist)

    plt.show()

def readAndShowImageWHistogram(image):
    im = util.img_as_ubyte(color.rgb2gray(io.imread(image)))

    printHistogramBilde(im)

    io.imshow(im)
    io.show()

inputImage2 = widgets.Dropdown(
    options={
        'Pollen Mørkt':'magnified-pollen-dark.tif',
        'Pollen Lav Kontrast':'pollen-lowcontrast.tif',
        'Einstein Lav Kontrast':'einstein-low-contrast.tif'},
    description='Bilde: ',
    disabled=False,
)

interactive(readAndShowImageWHistogram, image=inputImage2)
```

Bilde: Pollen Mørkt

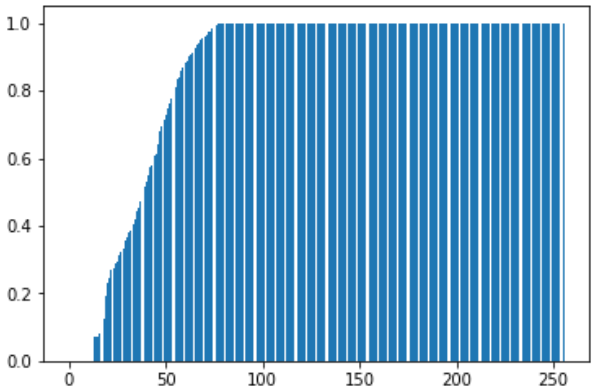




Her genereres den kumulative distibusjonen for bildet og vises

In [45]:
```
def cumulativeDistribution(hist):
    ret = numpy.zeros(256)
    ret[0] = hist[0]

    for x in range(1,256):
        ret[x] = ret[x-1] + hist[x]

    return ret

def cumulativeDistributionImage(im):
    hist = histogram(im)

    width, height = im.shape

    hist2 = numpy.zeros(256)

    for x in range(256):
        hist2[x] = hist[x] / (width*height)

    return cumulativeDistribution(hist2)

def plotCD(image):
    im = util.img_as_ubyte(color.rgb2gray(io.imread(image)))

    CD = cumulativeDistributionImage(im)

    ind = numpy.arange(len(CD))  # the x locations for the groups

    fig, ax = plt.subplots()
    rects1 = ax.bar(ind, CD)

    plt.show()

interactive(plotCD, image=inputImage2)
```

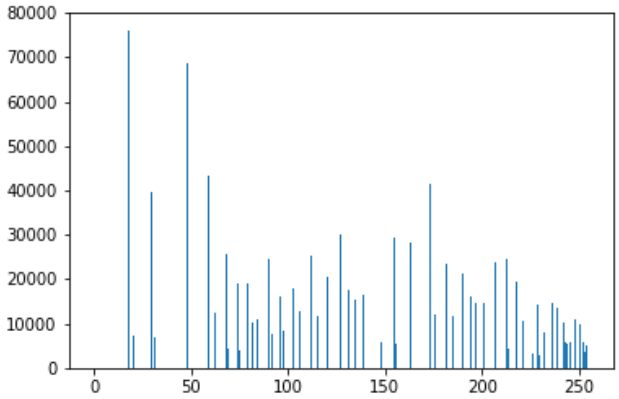
Bilde: Pollen Mørkt

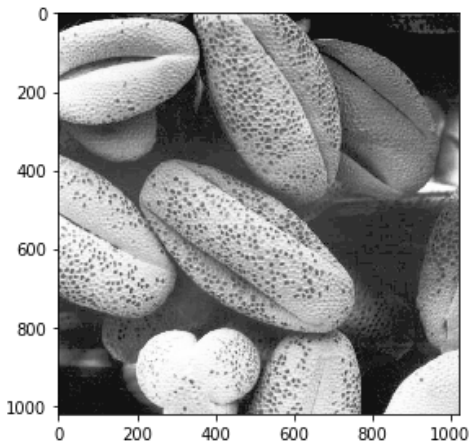


Bildet etter histogramnormalisering

In [46]:
```
def histogramEqualization(im):
    width, height = im.shape

    CD = cumulativeDistributionImage(im)

    for x in range(width):
        for y in range(height):
            im[x,y] = numpy.floor(CD[im[x,y]] * 255)

    return im

def histogramEq(image):
    im = util.img_as_ubyte(color.rgb2gray(io.imread(image)))

    im = histogramEqualization(im)

    printHistogramBilde(im)

    io.imshow(im, cmap='gray')
    io.show()

interactive(histogramEq, image=inputImage2)
```

Bilde: Pollen Mørkt





Histogramspesifikasjon

Her er ett par ting ikke implementert som i boka, om flere pixler i spesifikasjonen matcher til samme pixler vil den pixelen med høyest verdi være den som blir brukt, jeg så i eksemplet at man skal gjerne bruke den som er nærmest sin orginalplassering i orginalbildet.

Jeg har implementert noen spesifikasjoner som var enkle å kode og som muligens vil være interresante, om ikke ligger det en fil som heter spec.txt i samme mappe som denne notebook fila som kan brukes til å spesifisere ett egenvalgt spesifikasjon.

In [47]:

```
specificationOp = widgets.Dropdown(
    options=[
        'Linear Increase','Linear Decrease','Exponential Growth', 'Exponential
        ],
    description='Specification: ',
    disabled=False,
)

def inverseCumulative(cumulative):
    ret = numpy.zeros(256)

    cumulative = numpy.floor(cumulative*255)

    for x in range(256):
        ret[int(cumulative[x])] = x

    for x in range(1,256):
        if(ret[x] == 0):
            ret[x] = ret[x-1]

    return ret

def histogramMatch(image, specification):
    im = util.img_as_ubyte(color.rgb2gray(io.imread(image)))

    if(specification == 'Linear Increase'):
        specification = numpy.arange(256)
    elif(specification == 'Linear Decrease'):
        specification = numpy.flip(numpy.arange(256))
    elif(specification == 'Exponential Growth'):
        specification = numpy.zeros(256)
        for x in range(256):
            specification[x] = x**2
    elif(specification == 'Exponential Decay'):
        specification = numpy.zeros(256)
        for x in range(256):
            specification[x] = 256/(0.5*(x+1))
    elif(specification == 'Logarithmic'):
        specification = numpy.zeros(256)
        for x in range(256):
            specification[x] = math.log(x+1)
    elif(specification == 'spec.txt'):
        specification = numpy.loadtxt('spec.txt')

    specification = specification / sum(specification)

    ind = numpy.arange(len(specification))

    fig, ax = plt.subplots()
    rects1 = ax.bar(ind, specification)

    plt.show()

    specification = inverseCumulative(cumulativeDistribution(specification))

    width, height = im.shape

    #Normalize before matching
    im = histogramEqualization(im)

    for x in range(width):
        for y in range(height):
            im[x,y] = specification[im[x,y]]

    printHistogramBilde(im)
    io.imshow(im, cmap='gray')
    io.show()
```

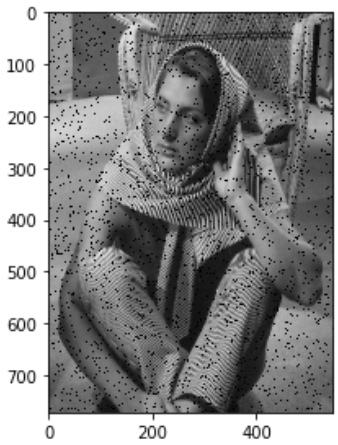
Denne var litt vanskeligere å se at fungerer, men som man kan se om man tar linjær increase, eller decrease, så ser man at bildene blir lysere og mørkere. Man kan også se om man velger exponential decay at histogrammet til bildet blir veldig nærme spesifikasjonen

Oppgave 3

Her kan du velge hvilket bilde som skal bli prosessert

In [3]:
```
def readAndShowImage(image):
    im = io.imread(image, as_gray=True)
    io.imshow(im, cmap='gray')
    io.show()

inputImage3 = widgets.Dropdown(
    options={
        'Barbara':'barbara.png',
        'Barbara Noise 1':'barbara_noise1.png',
        'Barbara Noise 2':'barbara_noise2.png'
    },
    description='Bilde: ',
    disabled=False,
)

interactive(readAndShowImage, image=inputImage3)
```

Bilde: Barbara Noise 1



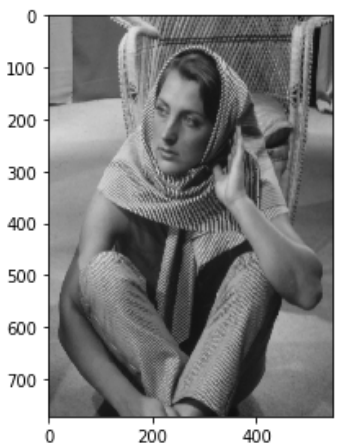
Valg av type filter samt størelsen på valgt filter

In [31]:
```
filterOptions = widgets.Dropdown(
    options=[
        'Box Blur','Gaussian Blur','UnSharpen','Median'
    ],
    description='Filter: ',
    disabled=False,
)

sizeOption = widgets.IntSlider(
    value = 5,
    min = 1,
    max = 30,
    description='Size: ',
    disabled=False,
    continuous_update = False
)


def filtrer(image, size, method):
    im = util.img_as_float(color.rgb2gray(io.imread(image)))

    if(method == 'Box Blur'):
        sizeOption.description = 'Size: '
        mask = numpy.ones((size,size))
        mask = mask / numpy.sum(mask)
        im = ndimage.convolve(im,mask,mode='constant', cval=0)
    elif(method == 'Gaussian Blur'):
        sizeOption.description = 'Sigma: '
        im = ndimage.gaussian_filter(im,size)
    elif(method == 'UnSharpen'):
        sizeOption.description = 'Sigma: '
        imBlurred = ndimage.gaussian_filter(im,size)
        imMask = im - imBlurred
        im = im + imMask
    elif(method == 'Median'):
        sizeOption.description = 'Size: '
        im = ndimage.median_filter(im,(size,size))


    io.imshow(numpy.clip(im,0,1), cmap='gray')
    io.show()

interactive(filtrer, image=inputImage3, size=sizeOption, method=filterOptions)
```

Bilde: Barbara Noise 1

Size: 3

Sigma: Median



Som konklusjon vil oppgave 3 vil jeg si at det filteret som fungerte best til å fjerne støy var medianfilteret, spesielt på bilde nummer 1 med støy. Salt og pepper støy fjernes ekstremt godt ved hjelp av median filtrering.

På dette bildet som har mye tekstur spesielt i buksene og stolen bak fungerer unsharpening ganske dårlig, det genereres mye støy her. Unsharp fungerer også veldig dårlig på bilder med støy da støyen blir amplifisert av skarpingen