Oppgave 1

In [2]:

```
from scipy import ndimage
from skimage import io,util,color
from IPython import display
import cmath
import numpy

from scipy import fftpack

%matplotlib inline
```

Kode for DFT og IDFT. Lagde en generell FT funksjon som kunne benyttes i begge funksjonene, som tar imot om den skal bruke j eller -j i formelen:

In [1]:

```
%%latex

\[ \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} f(x,y)e^{j2\pi(\frac{ux}{M}+\frac{vy}{N})} \]
```

$$\sum_{u=0}^{M-1} \sum_{v=0}^{N-1} f(x,y)e^{j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

In [3]:

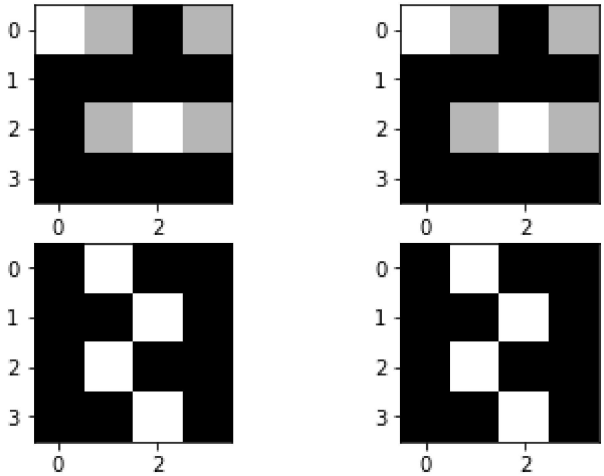
```
def FT(im,u,v,imaginary):
    s = complex(0,0)

    M,N = im.shape

    for x in range(M):
        for y in range(N):
            s += im[x,y] * cmath.exp(imaginary * 2 * cmath.pi * (((u*x) / M) + ((v*y) / N))
    return s


def DFT(im):
    height, width = im.shape

    out = numpy.zeros(im.shape,dtype=numpy.complex)

    for u in range(height):
        for v in range(width):
            out[u,v] = FT(im,u,v,-1j)

    return out

def IDFT(im):
    height, width = im.shape

    out = numpy.zeros(im.shape,dtype=numpy.complex)

    for u in range(height):
        for v in range(width):
            out[u,v] = (1/(height*width)) * FT(im,u,v,1j)

    return out
```

Under er koden fra lars vidar modifisert til å teste min implementasjon av funksjonene

In [4]:

```
# Read image, convert to gray, convert to float
im = numpy.asarray([[0, 1, 0, 0], [0, 0, 1, 0], [0, 1, 0, 0], [0, 0, 1, 0]], dtype=numpy.fl

# Retrieve the image dimensions
height,width = im.shape

# Perform the fourier transformation
ft_builtin = fftpack.fft2(im)
ft = DFT(im)

# Get the fourier spectrum by finding the length of the complex number vectors
ft_abs = numpy.absolute(ft)
ft_builtin_abs = numpy.absolute(ft_builtin)

#transformerer med invers fourier
transformed_back_builtin = numpy.real(fftpack.ifft2(ft_builtin))
transformed_back = numpy.real(IDFT(ft))


# Show the result
io.imshow_collection([ft_abs,ft_builtin_abs, transformed_back,transformed_back_builtin], cm
io.show()
```

```
C:\Users\Jim-Alexander\Anaconda3\lib\site-packages\scipy\fftpack\basic.py:16
0: FutureWarning: Using a non-tuple sequence for multidimensional indexing i
s deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future thi
s will be interpreted as an array index, `arr[np.array(seq)]`, which will re
sult either in an error or a different result.
  z[index] = x
```



Som vi kan se fra eksemplet over fungerer min implementasjon av DFT og IDFT. På venstre siden er min implementasjon, mens på høyre side er de innebygde funksjonene i fftpack. På øverse rad er frekvensdomenet, underste rad er transformert tilbake fra frekvensdomenet.

Oppgave 2

In [1]:

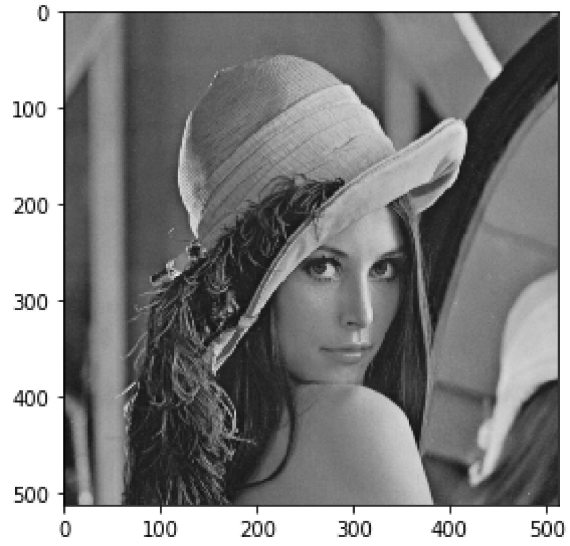
```
from skimage import io, util, color
from scipy import fftpack, ndimage
import numpy

import ipywidgets as widgets
from ipywidgets import interactive
```

Her velger man bildet som skal prosesseres, jeg har lagt inn bildene fra oppgaven.

In [2]:

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

inputImage = widgets.Dropdown(
    options={
        'Lena':'lena.png',
        'Barbara':'barbara.png'
    },
    description='Bilde: ',
    disabled=False,
)

interactive(readAndShowImage, image=inputImage)
```

Bilde: Lena



Funksjoner som trengs for frekvensfiltrering.

dist funksjonen brukes for å genere butterworth filtrene. fourrierFilter tar imot ett bilde og ett filter som skal bruker for frekvensfiltrering.

In [3]:

```
def dist(i,j,height,width):
    center_i, center_j = (height/2,width/2)
    return ((i-center_i)**2 + (j-center_j)**2)**0.5

def butterworth(height, width, cutoff, order):
    filtr = numpy.zeros((height,width))

    for i in range(height):
        for j in range(width):
            filtr[i,j] = 1 / (1 + (cutoff/ max(dist(i,j,height,width),1) )**(2*order))

    return filtr

def butterworthLowPass(height, width, cutoff, order):
    filtr = numpy.zeros((height,width))

    for i in range(height):
        for j in range(width):
            filtr[i,j] = 1 / (1 + ( max(dist(i,j,height,width),1) / cutoff )**(2*order))

    return filtr

def fourrierFilter(im, filtr):
    # Retrieve the image dimensions
    height,width = im.shape

    #finding P and Q
    padded_height = height*2
    padded_width = width*2

    #Padding image width 0s
    padded = numpy.zeros((padded_height,padded_width))
    padded[0:height,0:width] = im

    # Create an image to store the transformed image
    image_trans = numpy.zeros(padded.shape)
    # Transform each pixel in the image by multiplying with -1**(i+j) (see
    # slides/book) to translate the fourier transformed image to height/2 width/2
    # i.e. the center of the image
    for i in range(padded_height):
      for j in range(padded_width):
        image_trans[i,j] = padded[i,j]*((-1)**(i+j))

    # Perform the fourier transformation
    fourier_transform = fftpack.fft2(image_trans)

    #filtrerer den fourier transformerte
    filtered = filtr*fourier_transform

    #transformerer med invers fourier
    filtered = fftpack.ifft2(filtered)

    #translerer tilbake
    filtered_translated = numpy.zeros(padded.shape)

    for i in range(padded_height):
        for j in range(padded_width):
            filtered_translated[i,j] = filtered[i,j]*((-1)**(i+j))
```

```
    #fjerner padding
    filtered_cropped = filtered_translated[0:height,0:width]

    return numpy.clip(filtered_cropped,0,1)
```

Her kan man velge forskjellige typer filtrering og størelsen på filtrene.

In [8]:

```
filterOptions = widgets.Dropdown(
    options=[
        'Box Blur','UnSharp using box blur','Butterworth High Pass', 'Butterworth Low Pass'
    ],
    description='Filter: ',
    disabled=False,
)

sizeOption = widgets.IntSlider(
    value = 15,
    min = 1,
    max = 100,
    description='Size: ',
    disabled=False,
    continuous_update = False
)



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

    height,width = im.shape

    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 == 'UnSharp using box blur'):
        sizeOption.description = 'Size: '
        mask = numpy.ones((size,size))
        mask = mask / numpy.sum(mask)
        imBlurred = ndimage.convolve(im,mask,mode='constant', cval=0)
        imMask = im - imBlurred
        im = im + imMask
    elif(method == 'Butterworth High Pass'):
        sizeOption.description = 'Cutoff%: '
        size = (size/100) * (width/2)
        filtr = butterworth(height*2,width*2,size,2.25)
        im = fourrierFilter(im,filtr)
    elif(method == 'Butterworth Low Pass'):
        sizeOption.description = 'Cutoff%: '
        size = (size/100) * (width/2)
        filtr = butterworthLowPass(height*2,width*2,size,2.25)
        im = fourrierFilter(im,filtr)
    elif(method == 'Unsharp using butterworth'):
        sizeOption.description = 'Cutoff%: '
        size = (size/100) * (width/2)
        filtr = butterworthLowPass(height*2,width*2,size,2.25)
        imBlurred = fourrierFilter(im,filtr)
        imMask = im - imBlurred
        im = im + imMask



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

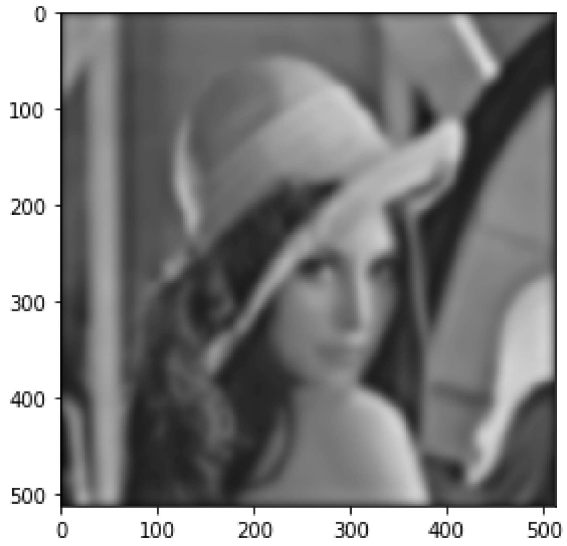
```
interactive(filtrer, image=inputImage, size=sizeOption, method=filterOptions)
```

Bilde: Lena

Cutoff%: 16

Filter: Butterworth Low Pass

```
C:\Users\Jim-Alexander\Anaconda3\lib\site-packages\ipykernel_launcher.py:5
8: ComplexWarning: Casting complex values to real discards the imaginary pa
rt
```



For å konkludere med noe her så må jeg si at jeg kan ikke se noen særlig forskjell på box filtret i spatial domet eller butterworth filtret i frekvensdomenet. Man har noe mer finkontroll over hvordan utjevningen skal bli med butterworth, men siden selve kalkuleringen tar mye lenger tid ville jeg heller gått for ett enkelt box filter. Selvfølgelig om jeg var en grafisk designer eller lignende hadde nok den ekstra finkontrollen kunnet gitt meg noe.

Oppgave 3

In [1]:

```
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

# Our imports
import skimage
import scipy
from skimage import util,color,exposure,io
from scipy import fftpack

%matplotlib inline
```

Funksjon for transformering til frekvensplanet

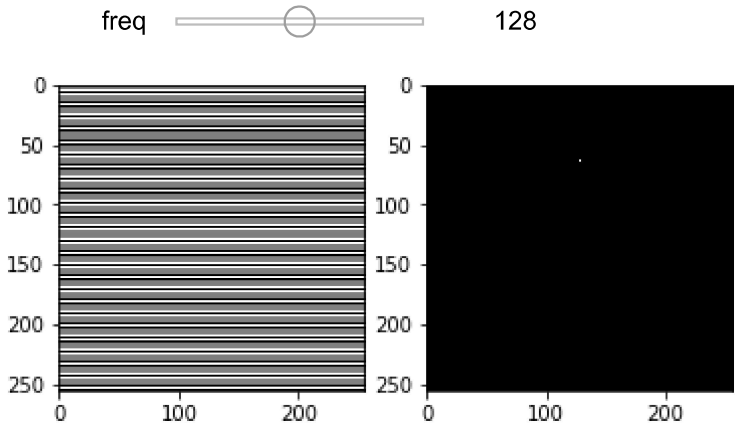
In [2]:

```
#Transformerer til frekvensplanet
def fourrier(im):
    # Retrieve the image dimensions
    height,width = im.shape

    # Create an image to store the transformed image
    image_trans = numpy.zeros(im.shape)
    # Transform each pixel in the image by multiplying with -1**(i+j) (see
    # slides/book) to translate the fourier transformed image to height/2 width/2
    # i.e. the center of the image
    for i in range(height):
      for j in range(width):
        image_trans[i,j] = im[i,j]*((-1)**(i+j))

    # Perform the fourier transformation
    ft = fftpack.fft2(image_trans)

    # Get the fourier spectrum by finding the length of the complex number vectors
    ft = numpy.absolute(ft)

    # Log transformation (see own lecture)
    out = numpy.zeros(im.shape)
    for i in range(0, im.shape[0]):
      for j in range(0, im.shape[1]):
        out[i,j] = numpy.log(1+ft[i,j])

    return out
```

Vil se hvordan ett bilde som kun har ett sinus signal i vertikal retning ser ut i frekvens domenet, sånn at jeg vet hva som skal filtreres vekk

In [10]:

```
def showSinusInFreq(freq):
    sinusImage = numpy.zeros((256,256))
    freq = (freq*math.pi)/256

    for x in range(256):
        for y in range(256):
            sinusImage[x,y] = math.sin(x*freq)

    io.imshow_collection([sinusImage,fourrier(sinusImage)], cmap='gray')
    io.show()

interactive(showSinusInFreq, freq=(1,256))
```



Som vi kan se legger frekvensen seg som en prikk en plass langs vertikal akse, sentrert horisontalt. Klarer ikke finne noen sammenheng om priken er plassert over eller under midten, men når frekvensen øker flytter prikken seg lenger vekk fra midten.

Min plan nå er å finne den frekvensen som er nærmest den frekvensen i bildet ved å sjekke differansen mellom genererte sinus bilder og saturn bildet, til jeg finner den hvor frekvensen er nærmest den i bildet. Så lar jeg filteret være det inverse av den nærmeste bildet i frekvens domenet.

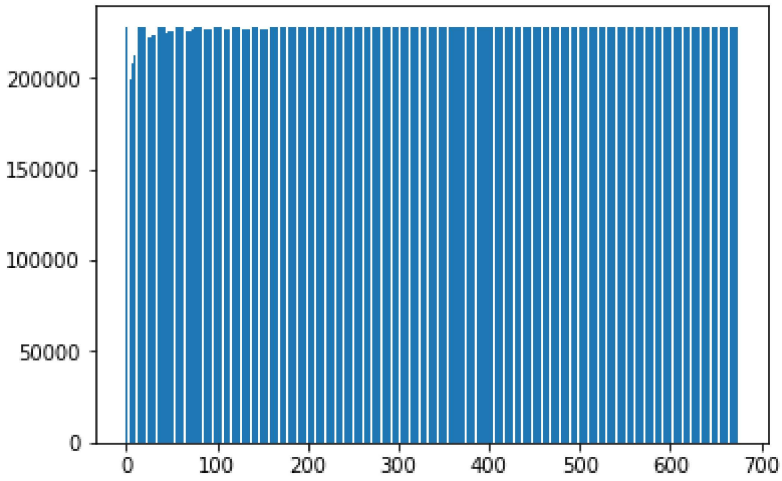
OBS ikke kjør denne dersom du ikke har god tid

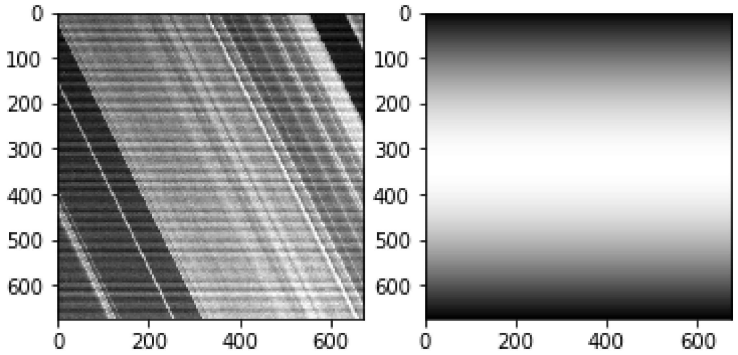
(det fungerte ikke uansett)

In [4]:

```
def plotHistogram(hist):
    ind = numpy.arange(len(hist))

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

    plt.show()


inputImage = widgets.Dropdown(
    options={
        'Saturn Med støy':'cassini-interference.png'
        },
    description='Bilde: ',
    disabled=True,
)

def findFrequencyImage(sourceImage):
    im = io.imread(sourceImage, as_gray=True)

    width, height = im.shape

    diff = numpy.zeros(height)

    minDiff = math.inf
    minFreq = 0

    for freq0 in range(height):

        sinusImage = numpy.zeros(im.shape)
        freq = (freq0*math.pi)/height

        for x in range(width):
            for y in range(height):
                sinusImage[x,y] = math.sin(x*freq)

        diffImage = im - 0.5*sinusImage
        difference = numpy.sum(diffImage)
        if(difference < minDiff):
            minFreq = freq0
            minDiff = difference

        diff[freq0] = difference

    sinusImage = numpy.zeros(im.shape)
    freq = (minFreq*math.pi)/height

    for x in range(width):
        for y in range(height):
            sinusImage[x,y] = 0.5 * math.sin(x*freq)


    plotHistogram(diff)
    io.imshow_collection([im, sinusImage], cmap='gray')

interactive(findFrequencyImage, sourceImage = inputImage)
```

Bilde: Saturn Med støy





Dette innser jeg nå ikke vil fungere da, det bildet med mest hvitt innhold vil være det som matcher orginal bildet best.

Jeg vil helle da generere filteret manuelt til jeg finner de verdiene som passer best til å fjerne støy.

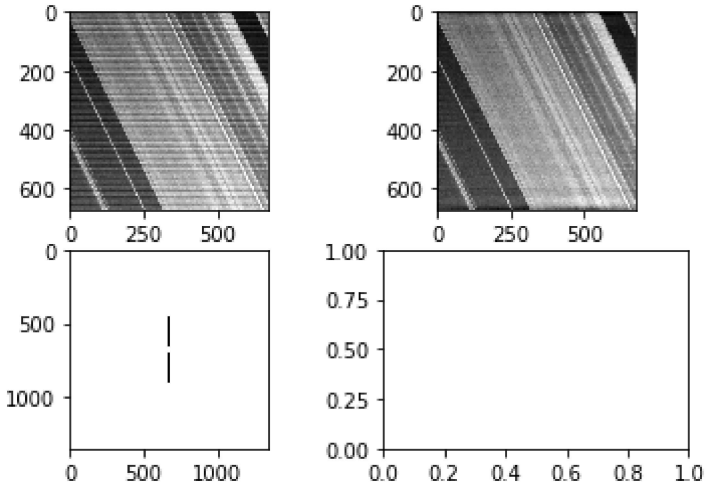
In [8]:

```
def fourrierFilter(image,cutoff):
    # Read image, convert to gray, convert to float
    im = util.img_as_float(color.rgb2gray(io.imread(image)))

    # Retrieve the image dimensions
    height,width = im.shape

    #finding P and Q
    padded_height = height*2
    padded_width = width*2

    #Padding image width 0s
    padded = numpy.zeros((padded_height,padded_width))
    padded[0:height,0:width] = im

    # Create an image to store the transformed image
    image_trans = numpy.zeros(padded.shape)
    # Transform each pixel in the image by multiplying with -1**(i+j) (see
    # slides/book) to translate the fourier transformed image to height/2 width/2
    # i.e. the center of the image
    for i in range(padded_height):
      for j in range(padded_width):
        image_trans[i,j] = padded[i,j]*((-1)**(i+j))

    # Perform the fourier transformation
    fourier_transform = fftpack.fft2(image_trans)



    #lage filter
    filtr = numpy.zeros(padded.shape)
    center_w = padded_width/2
    center_h = padded_height/2
    for x in range(padded_height):
        for y in range(padded_width):
            filtr[x,y] = 0 if ((math.fabs(y - center_w) < 5) and (math.fabs(x - center_h) >

    #filtrerer den fourier transformerte
    filtered = filtr*fourier_transform

    #transformerer med invers fourier
    filtered = fftpack.ifft2(filtered)

    #translerer tilbake
    filtered_translated = numpy.zeros(padded.shape)

    for i in range(padded_height):
        for j in range(padded_width):
            filtered_translated[i,j] = filtered[i,j]*((-1)**(i+j))

    #fjerner padding
    filtered_cropped = filtered_translated[0:width,0:height]

    # Show the result
    io.imshow_collection([numpy.clip(im,0,1),numpy.clip(filtered_cropped,0,1),filtr], cmap=
    io.show()

interactive(fourrierFilter,image=inputImage,cutoff=widgets.IntSlider(min=1,max=100,step=1,v
```

Bilde: Saturn Med støy

cutoff 20

```
C:\Users\Jim-Alexander\Anaconda3\lib\site-packages\ipykernel_launcher.py:4
9: ComplexWarning: Casting complex values to real discards the imaginary pa
rt
```



Som dere kan se er mesteparten av sinussignalet forsvunnet. I bilde 3 kan dere se filteret som ble produset, fiteret ble justert ved å prøve flere verdier av cutoff til det jeg synes er det beste resultatet ble produsert.