In [137]:

```
import numpy as np
from imageio import imread
import matplotlib.pyplot as plt
from scipy.signal import convolve2d

car = imread("../images/car.png", as_gray=True)
F = np.fft.fft(car[100])
M = len(F)
N = len(car)
sinus = lambda i, u: np.sin(2*np.pi*u*i/M)
cosinus = lambda i, u: np.cos(2*np.pi*u*i/M)
```

Litt om forrige forelesning

Det som skal skje:

- Litt om sinus
- 1D Eksempel på Fourier transform
- 2D Forklaring på Fourier transform

Sinus og cosinus

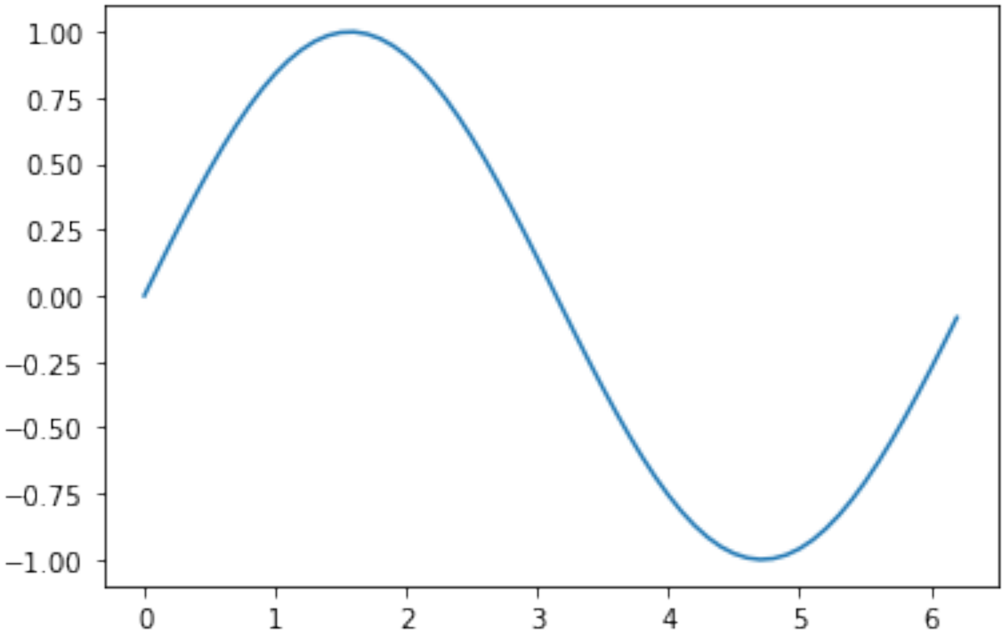
$$\sin(x), x \in [0, 2\pi)$$

In [135]:

```
index = np.arange(0,2*np.pi,0.1)
plt.plot(index, [np.sin(i) for i in index])
```

Out[135]:

```
[<matplotlib.lines.Line2D at 0x1c2726d6d0>]
```



I Fourier transform får vi denne funksjonen til å mappe fra 0 til M slik vi trenger. Dvs, alle våre x-er skal skaleres til mellom 0 og 2π. Dette gjør vi slik:

$$\frac{x}{M}2\pi, \frac{x}{M} \in [0, 1)$$

Feks, for x=0 og M = 256:

$$\frac{0}{256}2\pi = 0$$

for x=128:

$$\frac{128}{256}2\pi = \pi$$

Når vi øker frekvensen, mapper vi til et større intervall, feks 4π, 6π, osv.

Feks, for x=128 som mappes til 6π:

$$\frac{128}{256}6\pi = 3\pi$$

Dette betegnes med $\frac{x}{M}u2\pi$ og slik får vi den sinusen vi bruker:

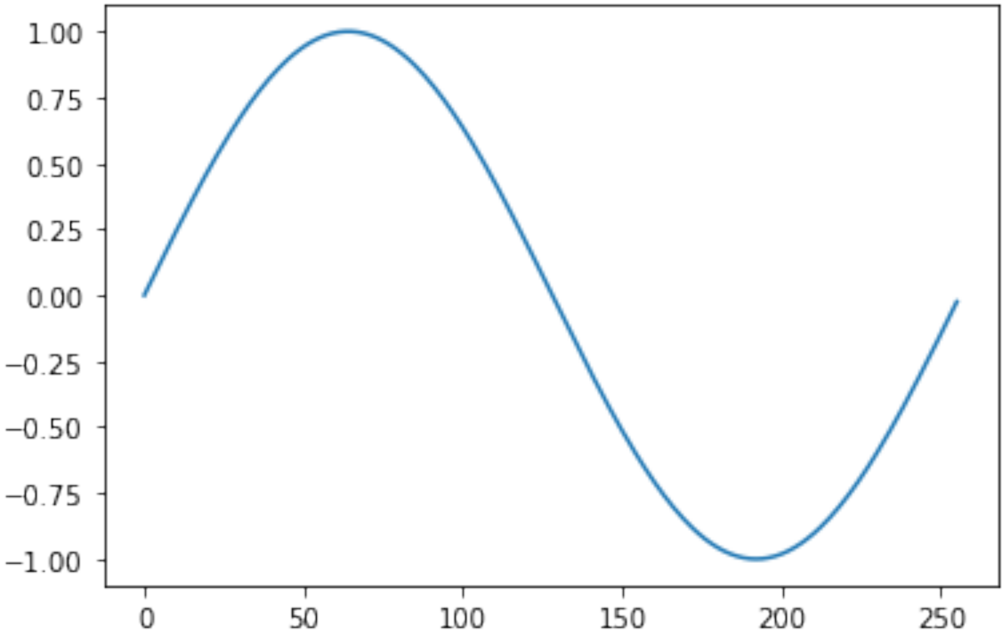
$$\sin\left(\frac{x}{M}u2\pi\right)$$

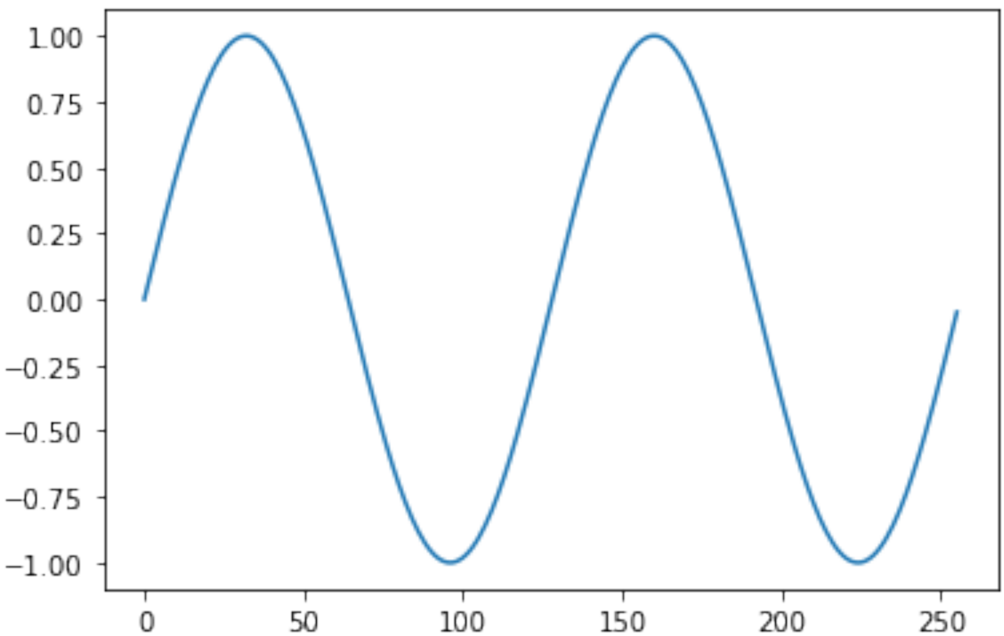
In [136]:

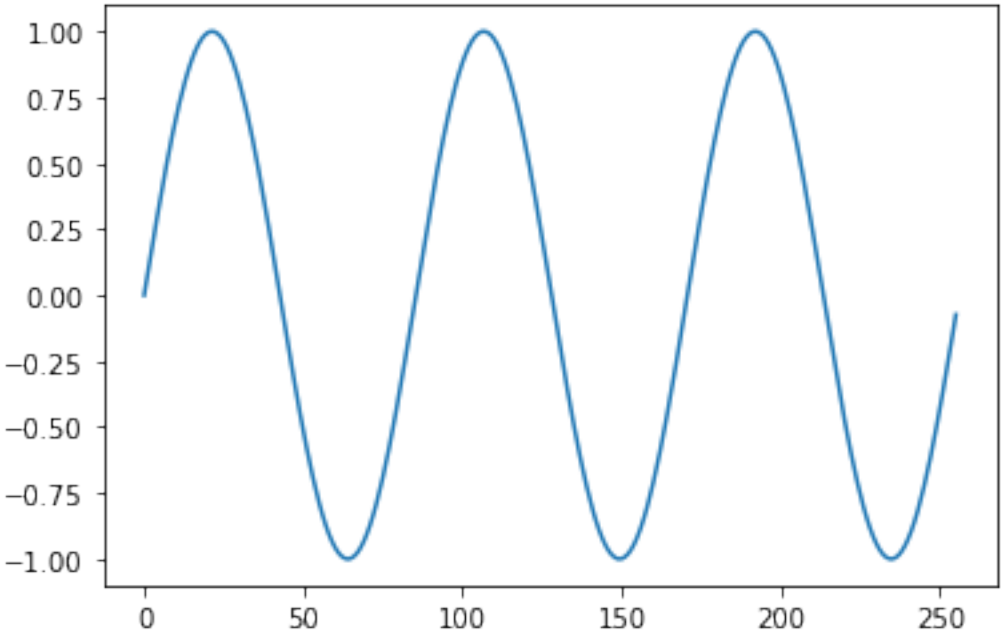
```
plt.figure()
plt.plot([sinus(i,1) for i in range(M)])
plt.figure()
plt.plot([sinus(i,2) for i in range(M)])
plt.figure()
plt.plot([sinus(i,3) for i in range(M)])
```

Out[136]:

```
[<matplotlib.lines.Line2D at 0x1c26b45350>]
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
50
100
150
200
250


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
50
100
150
200
250



Så, en til dimensjon, der y mapper likedan:

$$\sin\left(\frac{x}{M}u2\pi + \frac{y}{N}v2\pi\right)$$

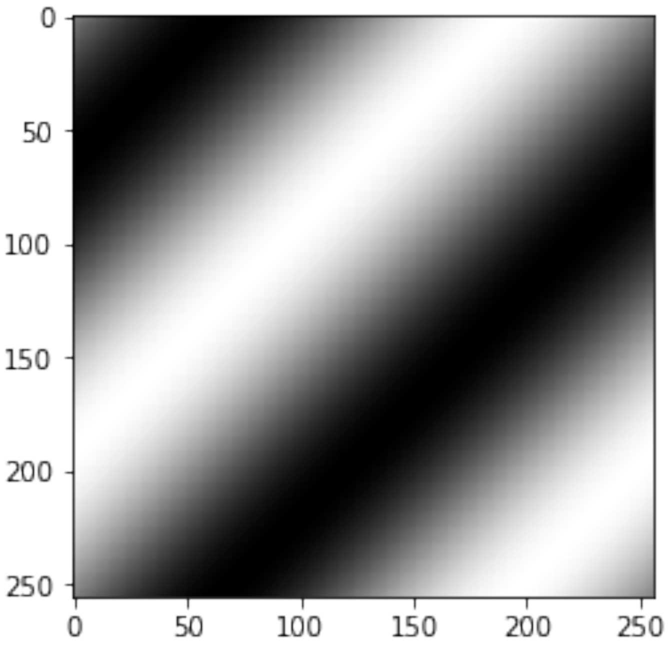
Som gir slike bilder:

In [247]:

```
sinus2D = lambda x,y,u,v: np.sin((-x*u*2*np.pi/M-y*v*2*np.pi/N))
plt.imshow([[sinus2D(i,j,1,1) for j in range(M)] for i in range(
N)], cmap="gray")
```

Out[247]:

```
<matplotlib.image.AxesImage at 0x1c49192250>
```



Vi bruker den samme sinus funksjonen, det er ingenting spesielt med at vi putter inn flere tall. Vi mapper en x og en y til en normal sinus fortsatt. Hvis vi putter inn en viss x, gir det mening at en y vil få oss lengre frem i sinusen.

Fourier transform

Alle funksjoner kan tilnærmes med uendelig mange sinus og cosinus funksjoner. Vi jobber med diskré funksjoner, og dette er ikke noe problem for Fourier.

Et 1D eksempel:

$$F(u) = \sum_{x=0}^{M-1} f(x)e^{-2\pi j \frac{ux}{M}}$$

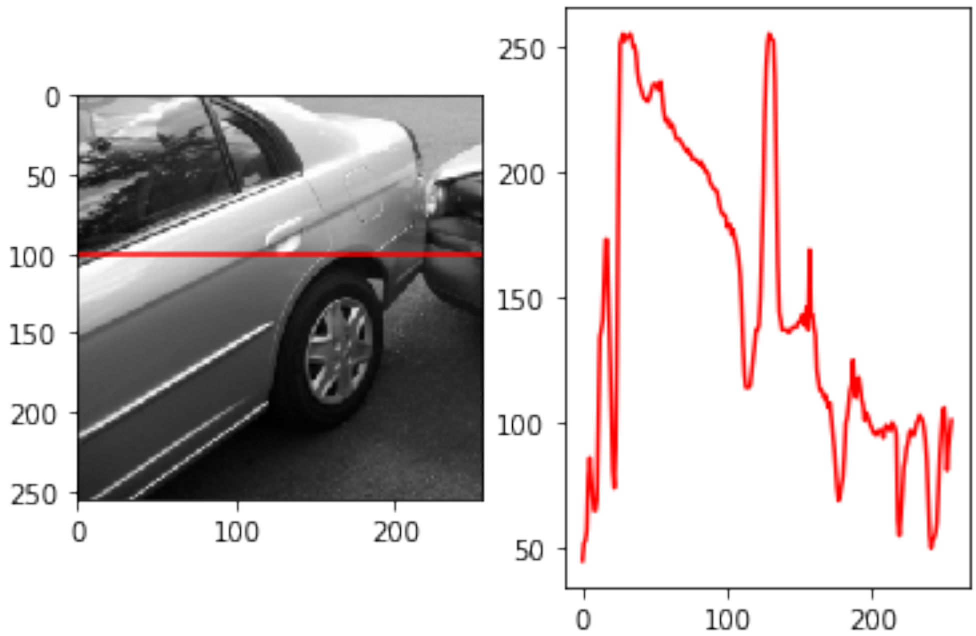
Vi vil tilnærme en linje fra bildet car.png:

In [125]:

```
fig = plt.figure()
fig.add_subplot(1,2,1)
plt.plot([100 for i in range(len(car))], color="red")
plt.imshow(car, cmap="gray")
fig.add_subplot(1,2,2)
plt.plot(car[100], color="red")
```

Out[125]:

```
[<matplotlib.lines.Line2D at 0x1c259589d0>]
```

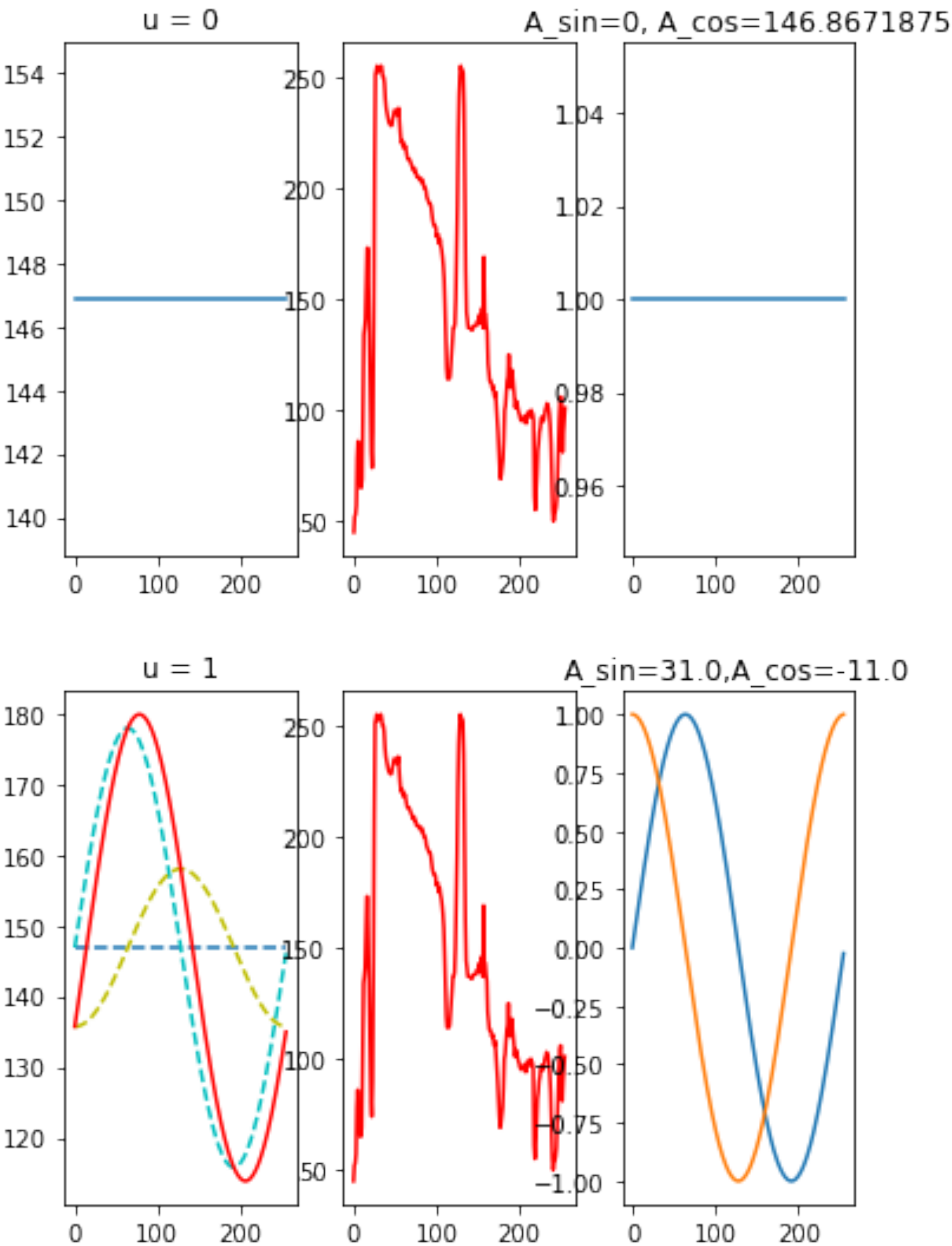


Vi vil tilnærme denne funksjonen med sinus og cosinus.

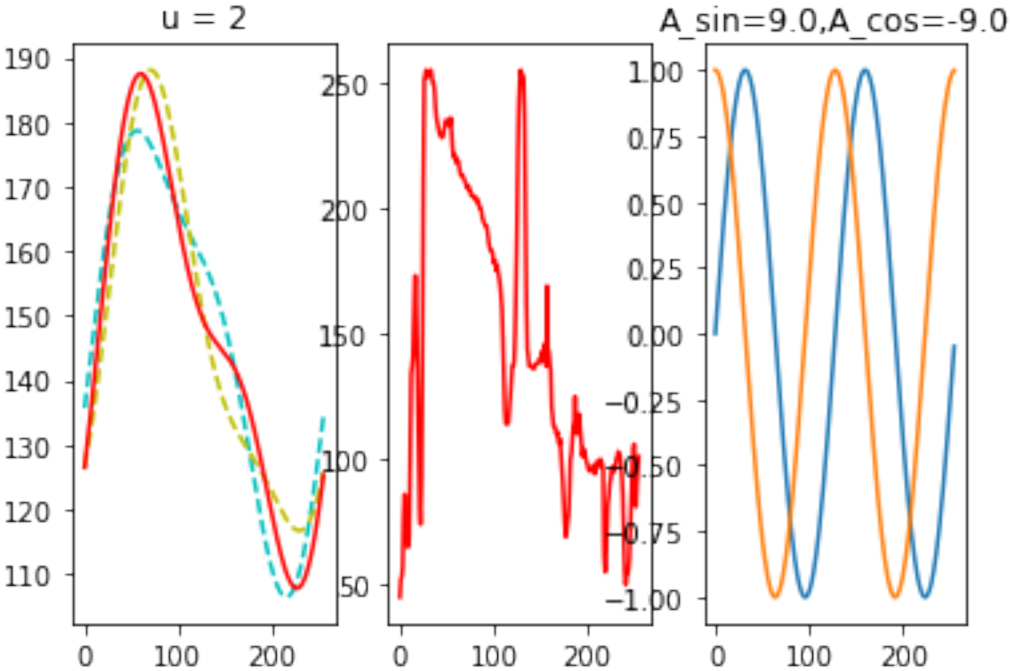
In [256]:

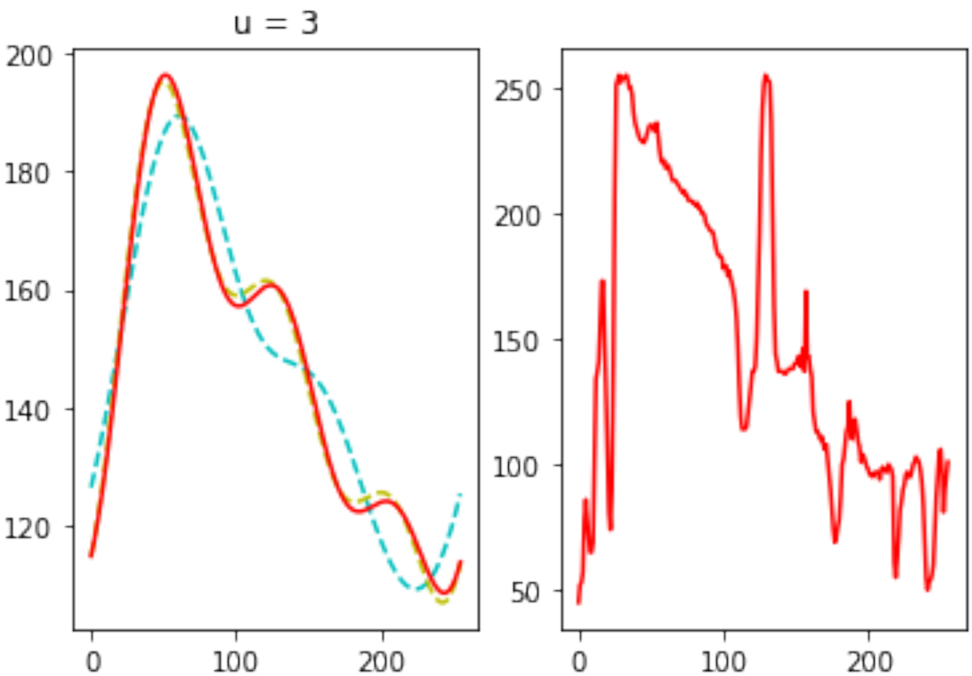
```
middelverdi = np.real(F[0])/M
basis0 = np.array([middelverdi for i in range(M)])


fig = plt.figure()
fig.add_subplot(1,3,1)
plt.plot(basis0)
plt.title("u = 0")
fig.add_subplot(1,3,2)
plt.plot(car[100], color="red")
fig.add_subplot(1,3,3)
plt.plot(basis0/middelverdi)
```

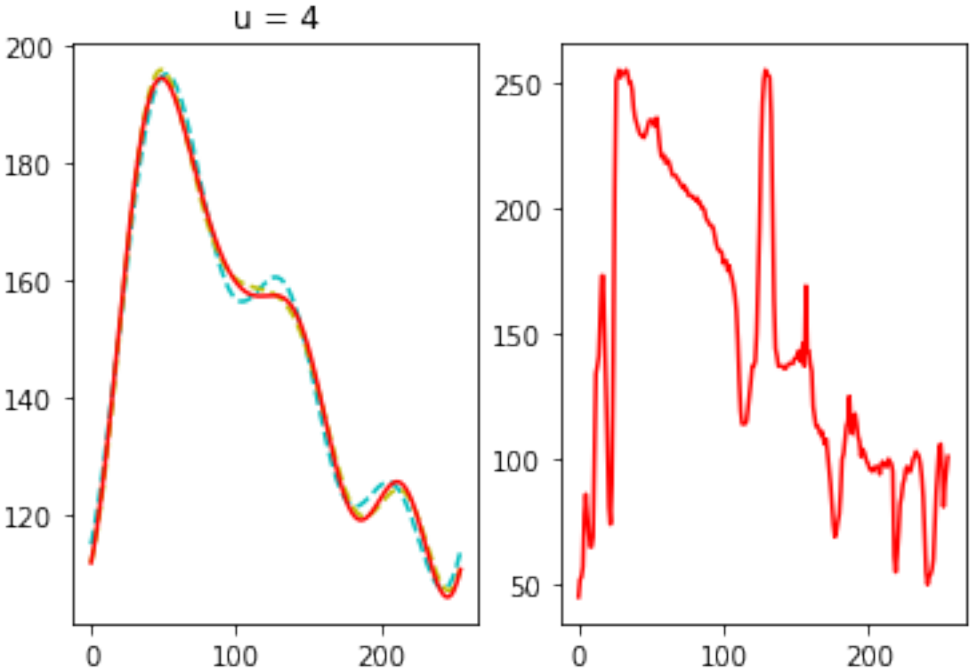
```
plt.title("A_sin=0, A_cos="+str(middelverdi))




middelverdi = np.real(F[0])/M
basis0 = np.array([middelverdi for i in range(M)])
lookAlike = basis0.copy()

for u in range(1,M):
    unscaledSin = np.array([sinus(i,u) for i in range(M)])
    unscaledCos = np.array([cosinus(i,u) for i in range(M)])
    basisUsin = -np.imag(F[u])/M * unscaledSin
    basisUcos = np.real(F[u])/M * unscaledCos

    if u < 10 or u == M-1 or u == M//2:
        fig = plt.figure()
        fig.add_subplot(1,3 if u < 3 or u == 255 else 2,1)
        if u < 2:
            plt.plot(lookAlike, "--")
        plt.plot(lookAlike + basisUsin, "c--")
        plt.plot(lookAlike + basisUcos, "y--")
        plt.title("u = "+str(u))

    lookAlike += (basisUsin + basisUcos)
    if u < 10 or u == M-1 or u == M//2:
        plt.plot(lookAlike, color="red")

        fig.add_subplot(1,3 if u < 3 or u == 255 else 2,2)
        plt.plot(car[100], color="red")

        if u < 3 or u == 255:
            fig.add_subplot(1,3,3)
            plt.plot(unscaledSin)
            plt.plot(unscaledCos)
            plt.title("A_sin="+str(np.round(-np.imag(F[u])/M))+"
,A_cos="+str(np.round(np.real(F[u])/M)))
```

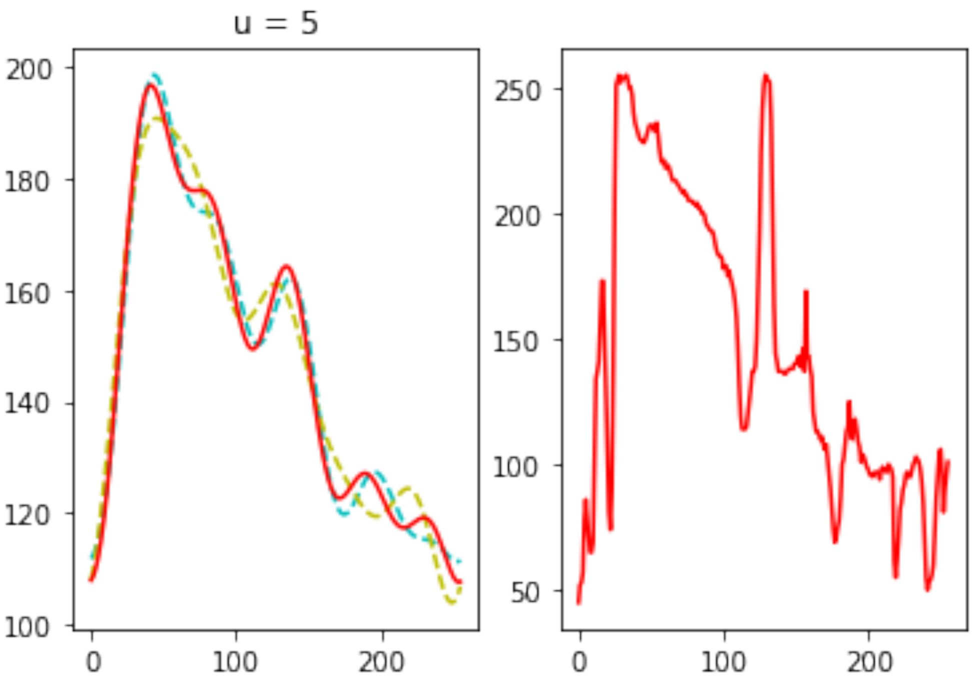


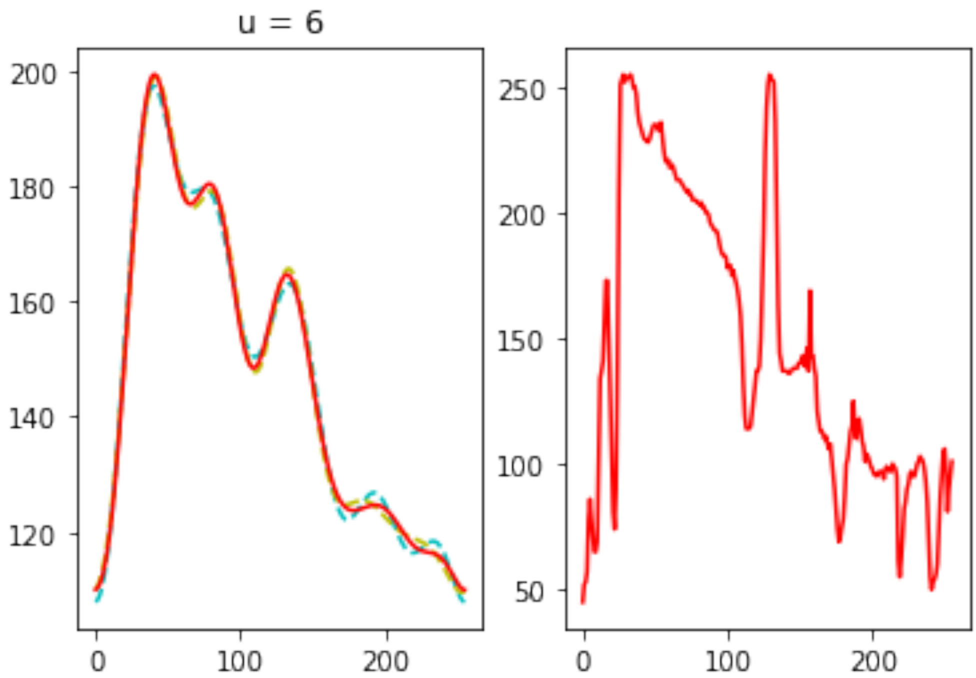
u = 0
A_sin=0, A_cos=146.8671875
154
152
150
148
146
144
142
140
250
200
150
100
50
1.04
1.02
1.00
0.98
0.96
0
100
200
u = 1
A_sin=31.0,A_cos=-11.0
180
170
160
150
140
130
120
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

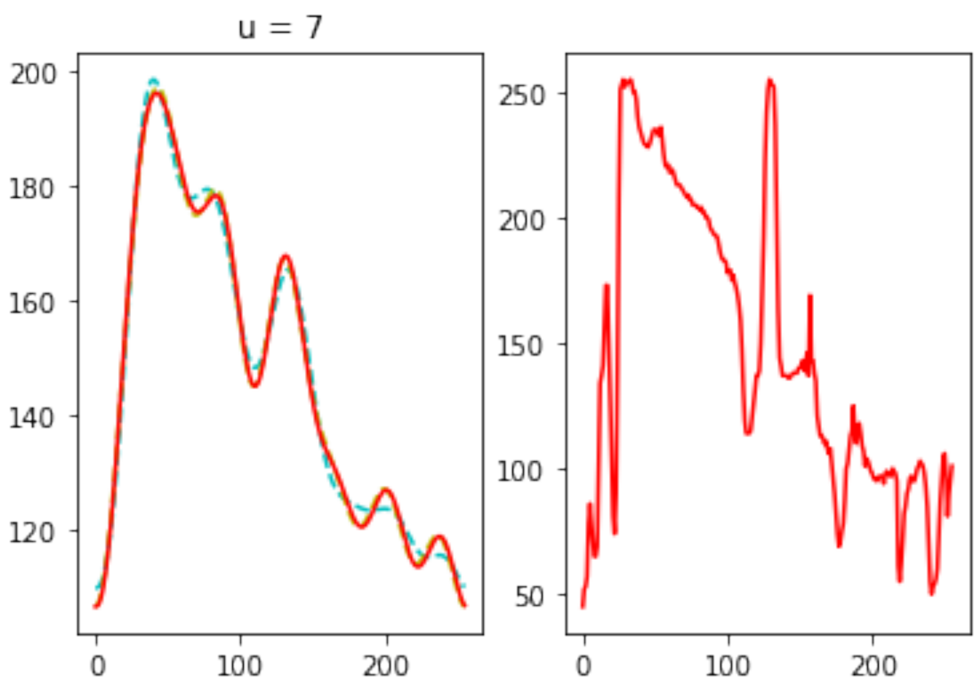

u = 2
A_sin=9.0,A_cos=-9.0

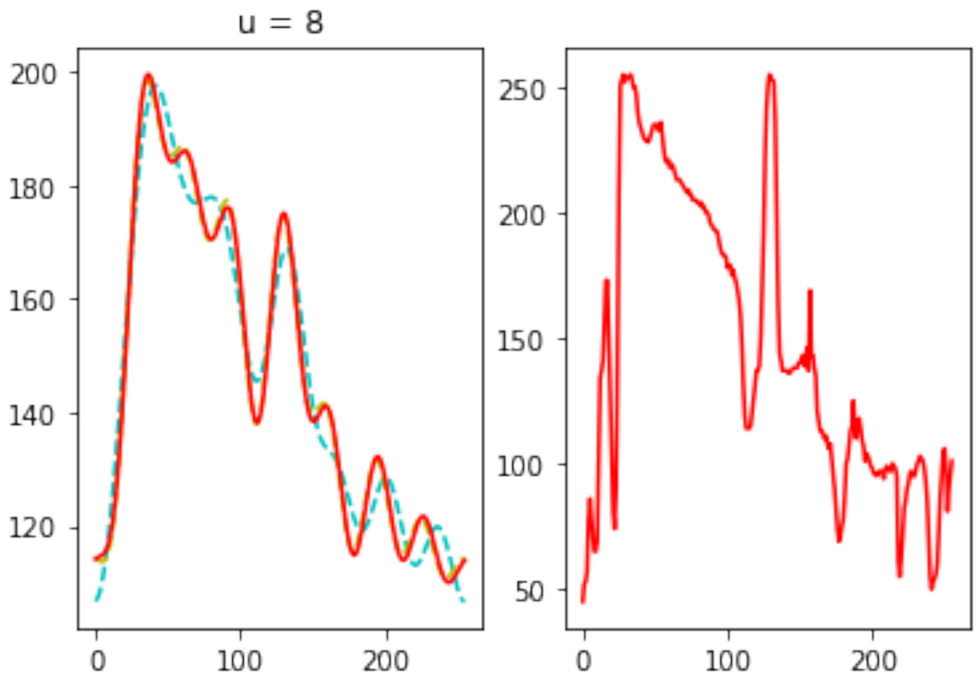

u = 3

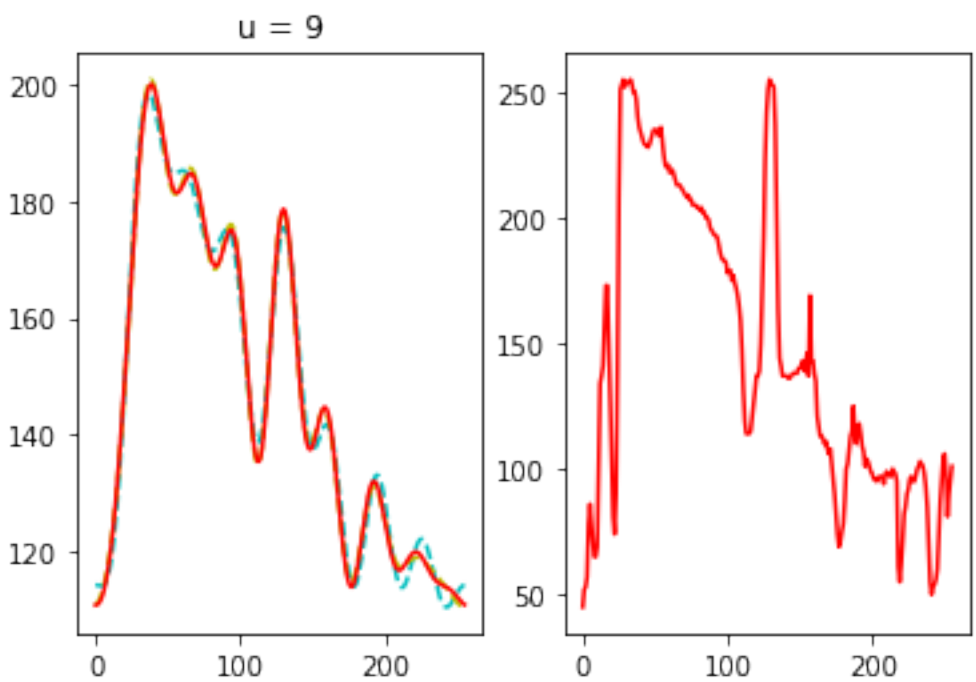

u = 4
200
180
160
140
120
0
100
200
250
200
150
100
50
0
100
200

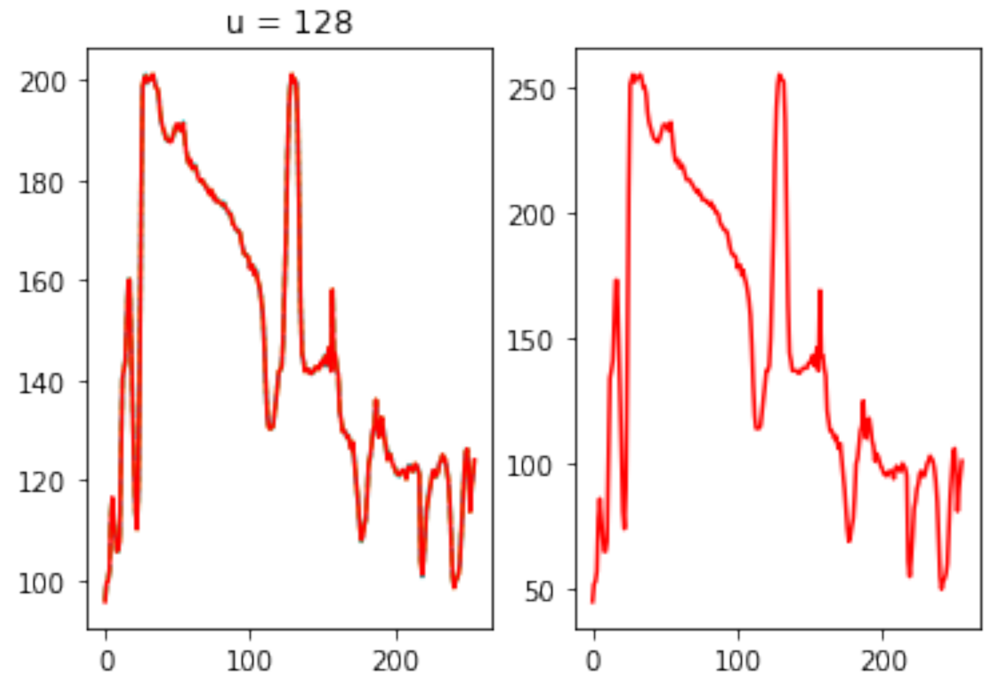

u = 5
200
180
160
140
120
100
0
100
200
250
200
150
100
50
0
100
200

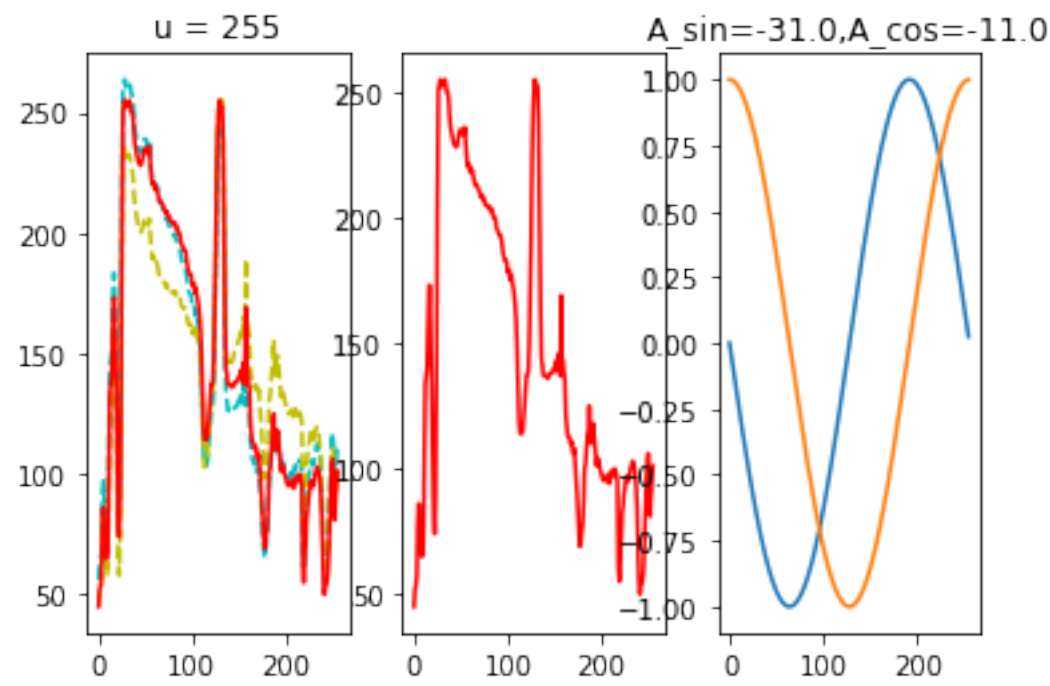

u = 6
200
180
160
140
120
0
100
200
250
200
150
100
50
0
100
200


u = 7
200
180
160
140
120
0
100
200
250
200
150
100
50
0
100
200


u = 8
200
180
160
140
120
0
100
200
250
200
150
100
50
0
100
200


u = 9
200
180
160
140
120
0
100
200
250
200
150
100
50
0
100
200





Noen ting å merke seg:

- Ingen informasjon går tapt i dikré Fourier transformasjon: Før og etter er identisk
- Amplitude for cosinus er lik for u = 1 som u = 255
- Amplitude for sinus er akkurat motsatt
- u = 128 ser formlikt ut som u = 255, men skalert forskjellig

Den største mulige frekvensen er på cirka u,v=N/2. Denne avkuttingen gjør livet enkelt for oss når en kontinuerlig Fourier transformasjon ville fortsatt uendelig lenge.

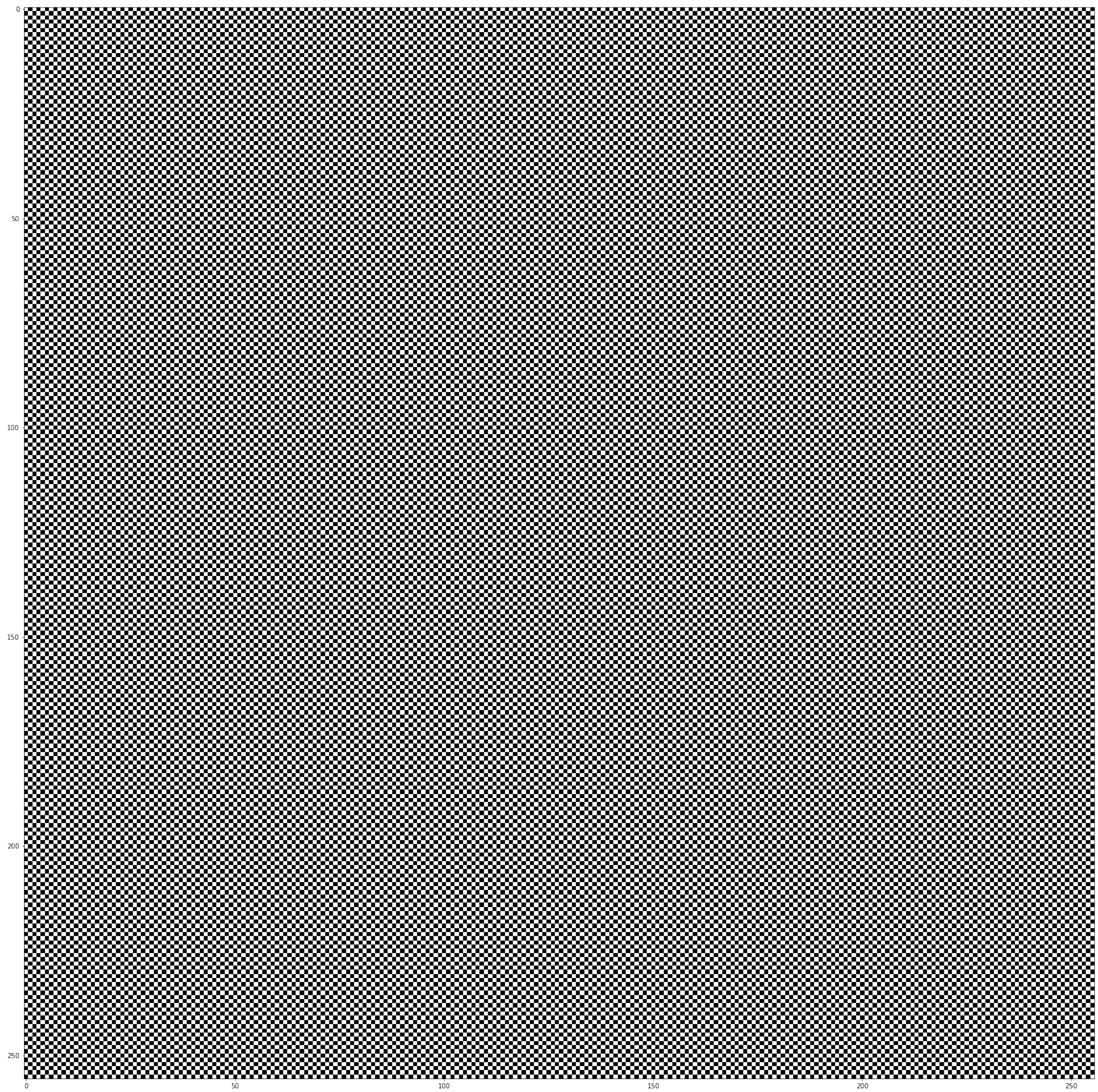
Dette minner om Nyquist: $f_{sample} > 2f_{max}$. Vi trenger å sample en topp og en dal og helst litt mer, så i et bilde med NxN piksler, kan vi ikke ha frekvenser med større hyppighet enn dette. Ettersom vi mapper x og y til et større og større sinus intervall, går vi forbi punktet vi kan sample med de skrittene vi tar på $\frac{x}{M}$, og gjennom en aliasing effekt går co-/sinusbildene våre tilbake til lave frekvenser.

In [214]:

```
cosinus2D = lambda x,y,u,v: np.cos((((x*u*2*np.pi)/(M))+((y*v*2*np.pi)/(N))))
plt.figure()
f, ax = plt.subplots(1, 1, figsize = (30, 30))
plt.imshow([[cosinus2D(i,j,M//2,N//2) for i in range(M)] for j in range(N)], cmap="gray")
```

Out[214]:

```
<matplotlib.image.AxesImage at 0x1c1e6b1250>

<Figure size 432x288 with 0 Axes>
```


0
50
100
150
200
250
0
50
100
150
200
250

Men hvordan fant jeg amplitudene?

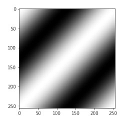
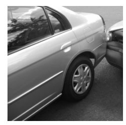
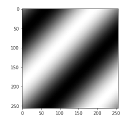
På forelesningen snakket Kristine om indreprodukt. Jeg vil her visualisere det litt mer.

Dette er formelen for DFT fra foilene:

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) * e^{-2\pi j(\frac{ux}{M} + \frac{vy}{N})}$$

Det kan være nyttig og tenke på den slik:

$$F(u, v) = sum(bilde * cosinus\_bilde(u, v), sum(bilde * sinus\_bilde(u, v)$$

$F(u, v) = sum\Big($  $*$  $\Big), sum\Big($  $*$  $\Big)$

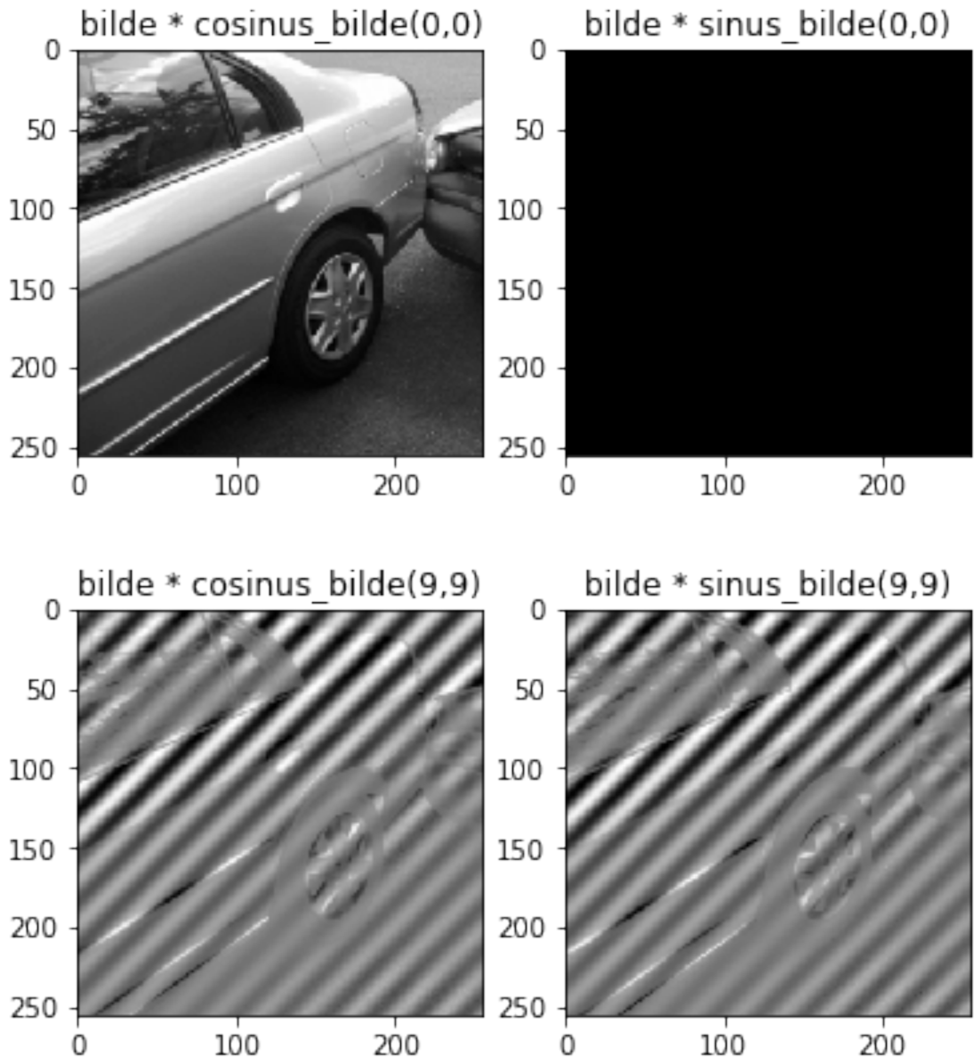
Under finner jeg F(u,v) for de 20x20 første frekvensene:

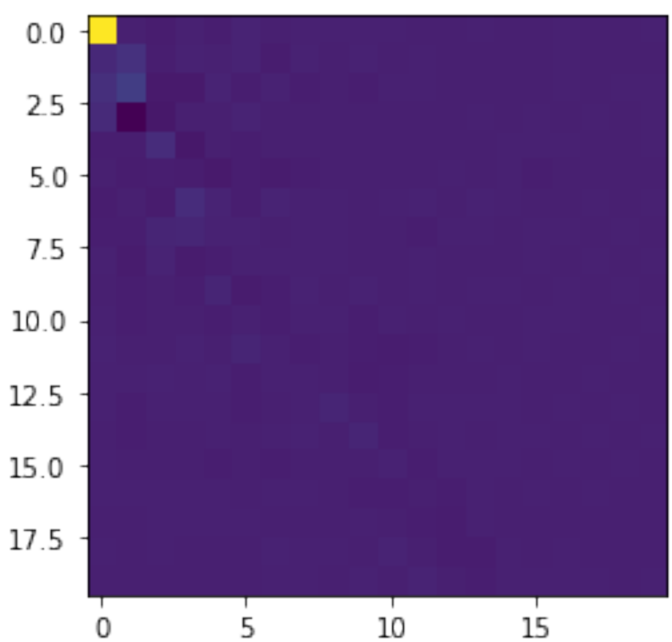
In [258]:

```python
cosinus_bilde = lambda u, v: [[cosinus2D(i,j,u,v) for i in range(M)] for j in range(N)]
sinus_bilde = lambda u, v: [[sinus2D(i,j,u,v) for i in range(M)] for j in range(N)]

n = m = 20
F_bad = np.zeros((n,m,2))

for u in range(m):
    for v in range(n):
        cosContribution = car * cosinus_bilde(u,v)
        sinContribution = car * sinus_bilde(u,v)
        if (u == 0 and v == 0) or (u == 9 and v == 9):
            fig = plt.figure()
            fig.add_subplot(1,2,1)
            plt.imshow(cosContribution, cmap="gray")
            plt.title("bilde * cosinus_bilde("+str(u)+","+str(v)+")")
            fig.add_subplot(1,2,2)
            plt.imshow(sinContribution, cmap="gray")
            plt.title("bilde * sinus_bilde("+str(u)+","+str(v)+")")
        F_bad[v,u,0] = np.sum(cosContribution)
        F_bad[v,u,1] = np.sum(sinContribution)

plt.figure()
showReal = [[F_bad[i][j][0] for j in range(n)] for i in range(n)]
plt.imshow(showReal)
```

Out[258]:

```
<matplotlib.image.AxesImage at 0x1c1d5d04d0>
```


bilde * cosinus_bilde(0,0)
bilde * sinus_bilde(0,0)
bilde * cosinus_bilde(9,9)
bilde * sinus_bilde(9,9)



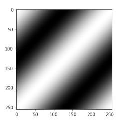
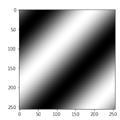
Ser at (0,0) er veldig lys. Her er alle pikslene summert, fordi, som kan sees, bilde * cosinus_bilde(0,0) er bare bildet selv. Dette er fordi cosinus(0) = 1.

Når vi setter dette sammen igjen bruker vi formelen for iDFT, invers Fourier:

$$f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{2\pi j(\frac{ux}{M} + \frac{vy}{N})}$$

som igjen kan tenkes på som

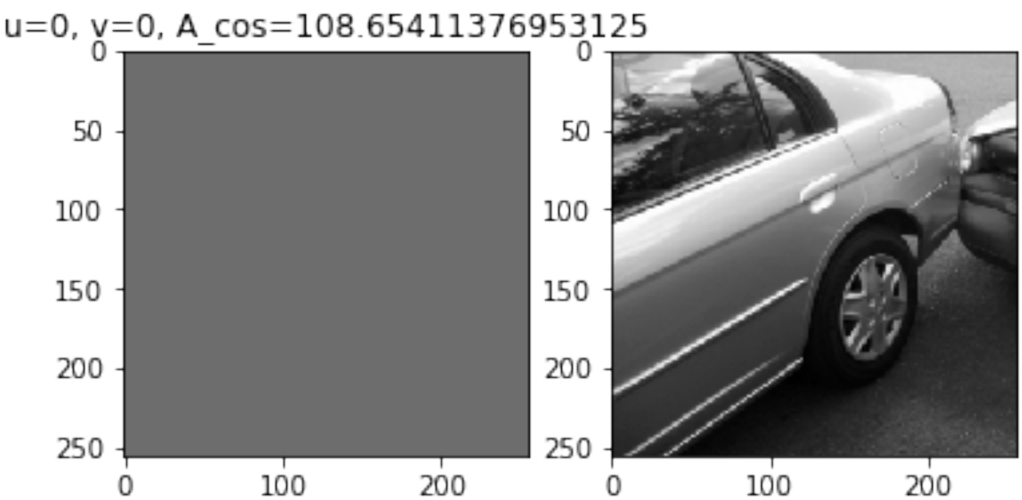
$$f = \sum_{u=0}^{M} \sum_{v=0}^{N} amplitude\_cos(u, v) * cos\_bilde(u, v) + amplitude\_sin(u, v) * si$$

$f = \sum_{u=0}^{M} \sum_{v=0}^{N} A_{cos} *$  $+ A_{sin} *$ 

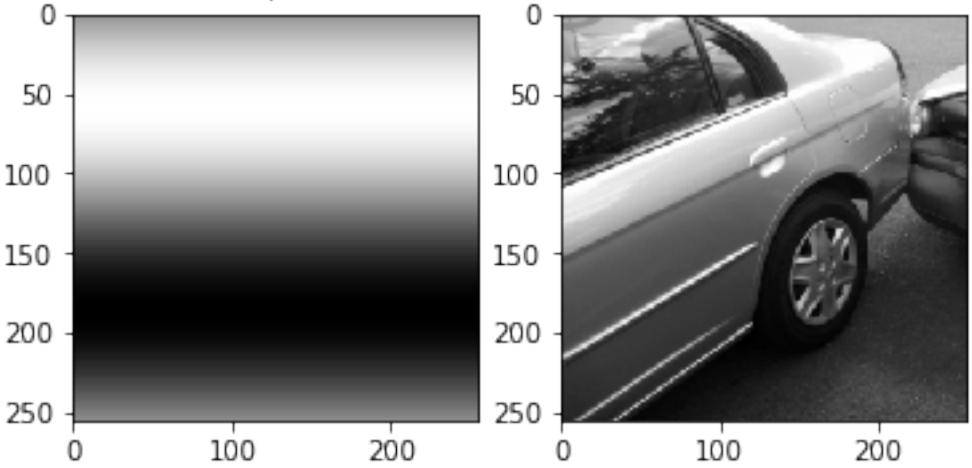
der amplituden finnes ved å ta F(u,v), realdelen for cos, imaginærdelen for sin, og dele på antall piksler i bildet. Dette er en slags middelverdi av contribution.

In [259]:

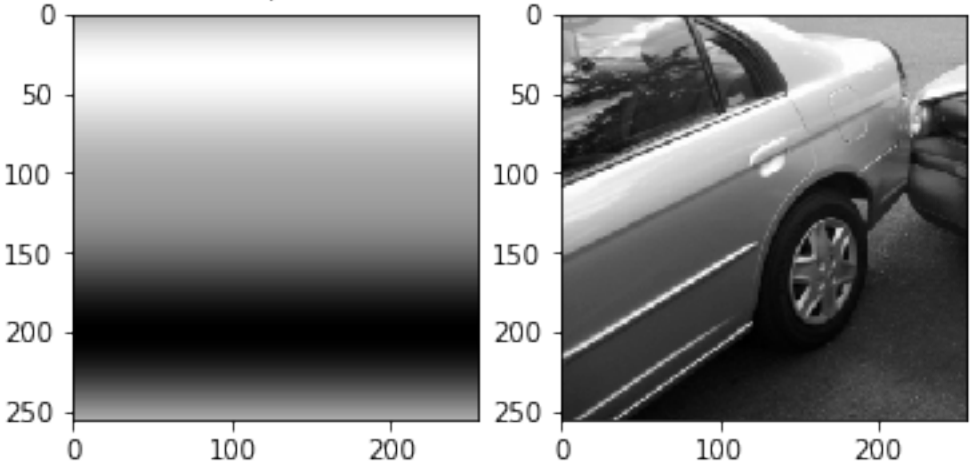
```
car_bad = np.zeros((N,M))

for u in range(m):
    for v in range(n):
        a_cos = F_bad[v,u,0] / (N*M)
        a_sin = F_bad[v,u,1] / (N*M)
        car_bad += (a_cos * np.array(cosinus_bilde(u,v)))
        car_bad += (a_sin * np.array(sinus_bilde(u,v)))

        if (u < 3 and v < 3) or (u == m-1 and v == n-1):
            fig = plt.figure()
            fig.add_subplot(1,2,1)
            if u == 0 and v == 0:
                plt.title("u="+str(u)+", v="+str(v)+", A_cos="+str(a_cos))
                plt.imshow(car_bad, cmap="gray", vmin=0, vmax=255)
            else:
                plt.imshow(car_bad, cmap="gray")
                plt.title("u="+str(u)+", v="+str(v))
            fig.add_subplot(1,2,2)
            plt.imshow(car, cmap="gray")
```



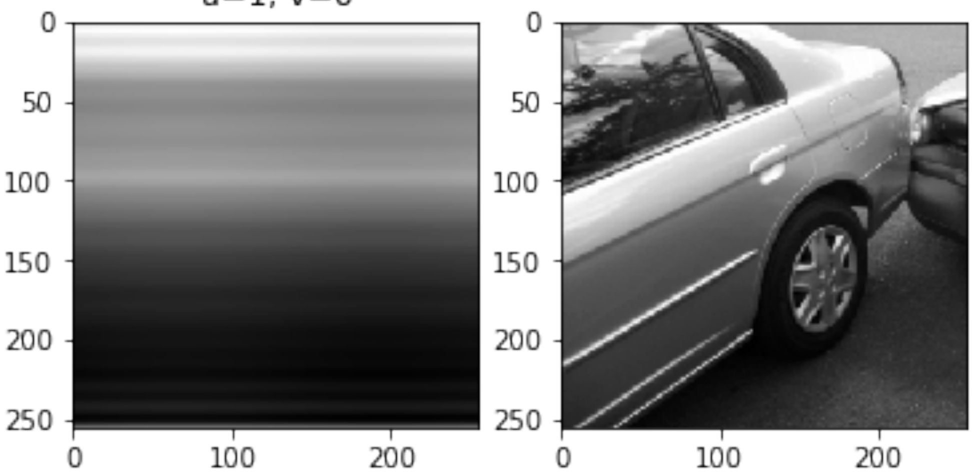
u=0, v=1

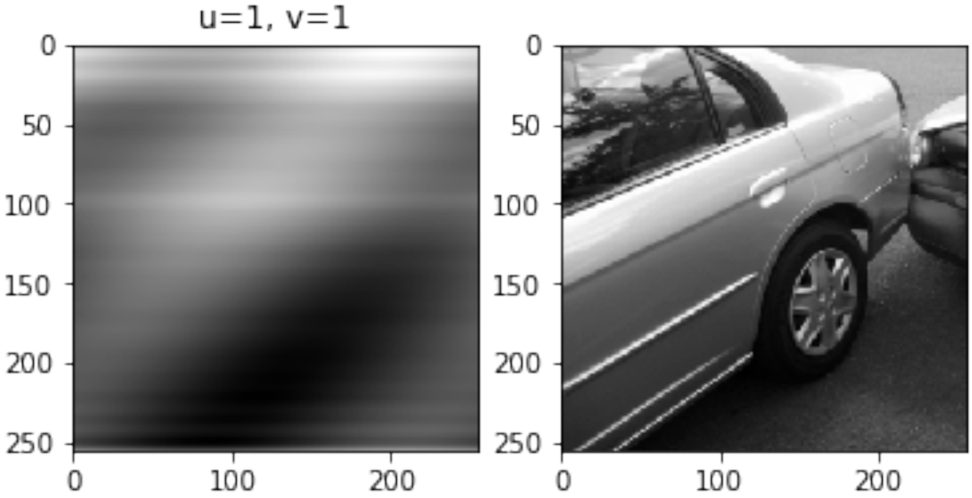


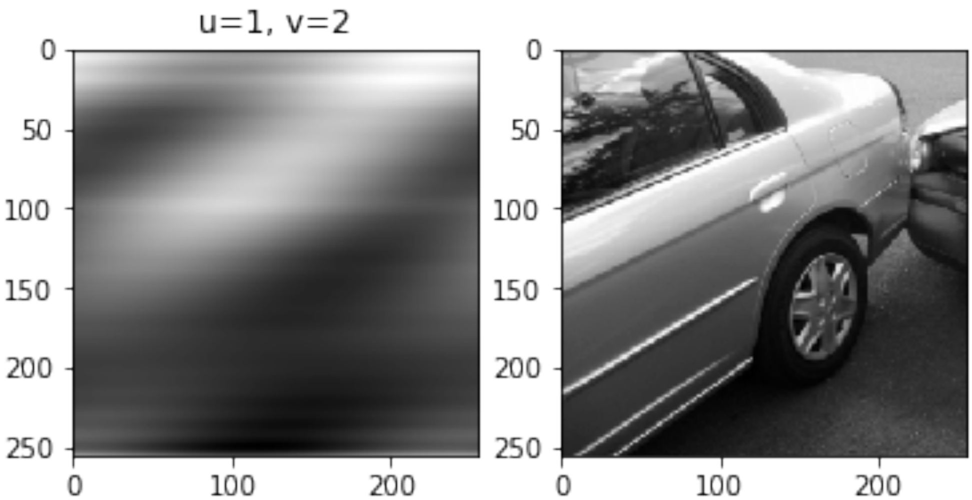
u=0, v=2

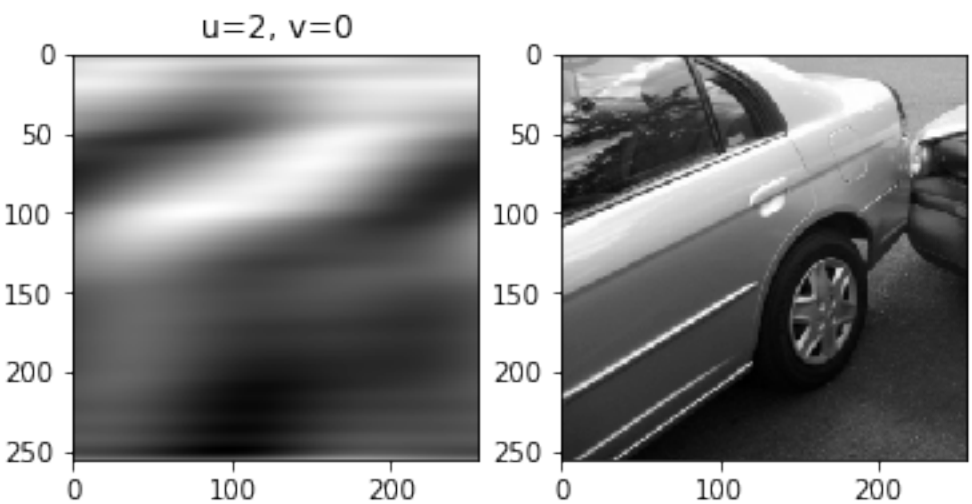


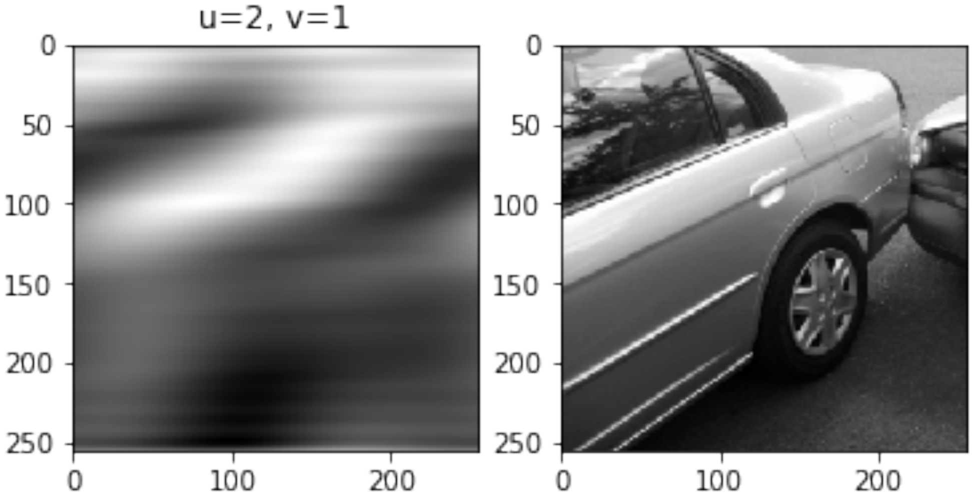
u=1, v=0

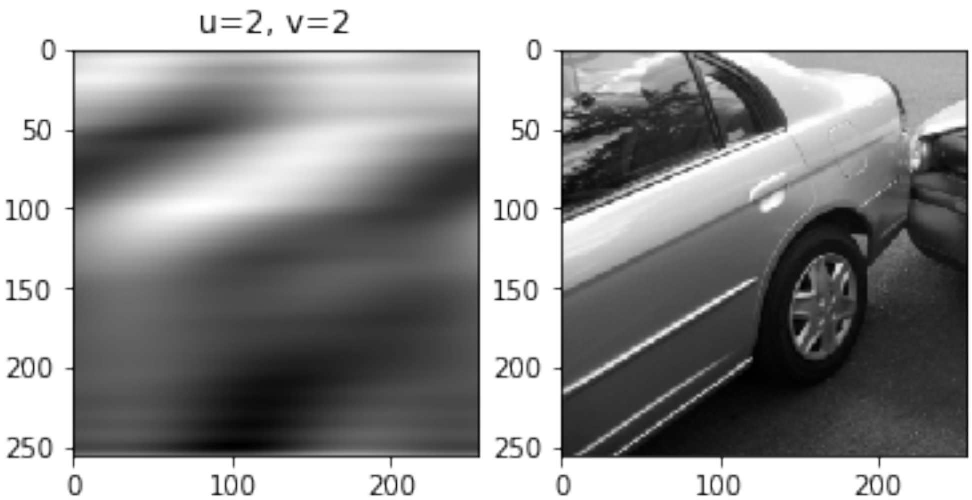


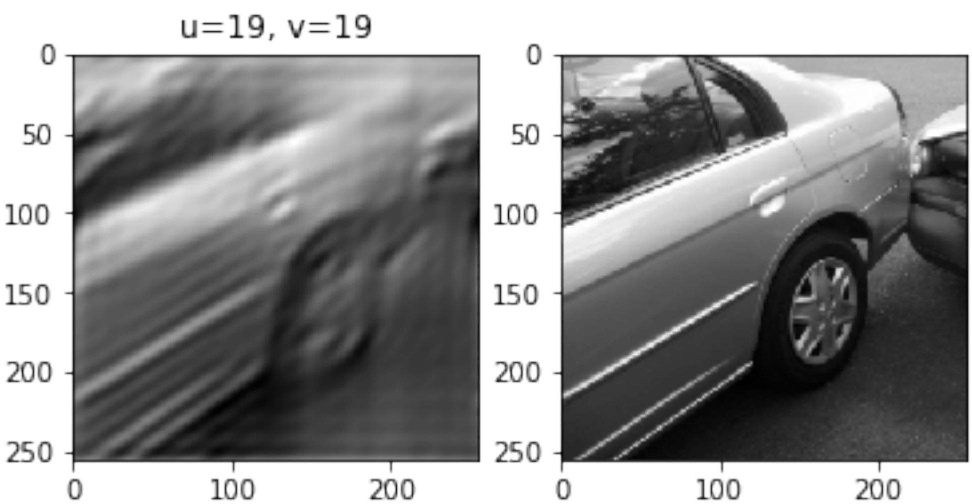

u=1, v=1


u=1, v=2


u=2, v=0







Og det var det :)