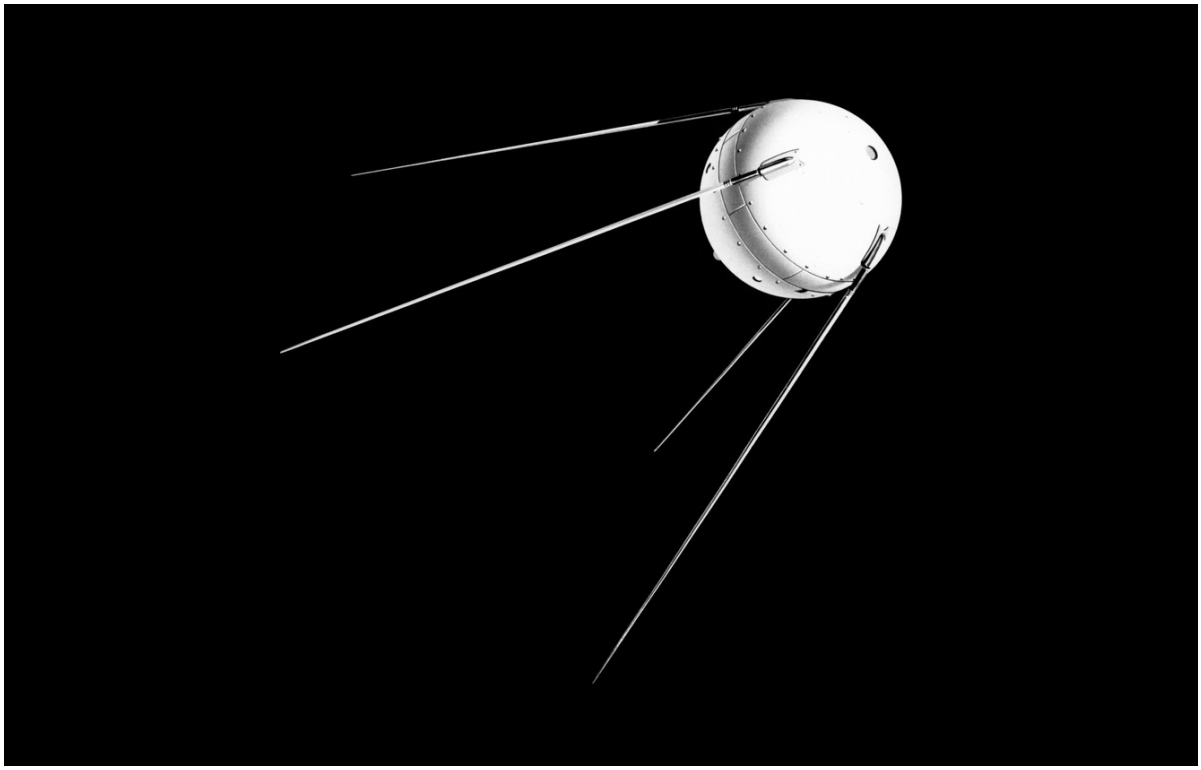
Hvordan kan vi få informasjon fra satelitten når all kommunikasjon er brutt?

Computational Essay skrevet av Domantas Sakalys

22. april 2021



Introduksjon

I denne Computational Essay skal vi ta for oss et sikkerhets tilfelle får et case scenario. Når vi sender satelitter opp til rommet, ønsker vi oss å holde på kommunikasjon mellom basene som er på jorda, og satelitten som er oppe i banen rundt jorda. Det er spesielt viktig å vite satelittens posisjon og hastighet under banets justeringer, derfor brutt på kommunikasjon kan føre til katastrofale konsekvenser.

I denne computational essay skal vi derfor fordype oss inn i og bygge et model som vi kan bruke i slike scenarier.

Vi skal begynne først med et enkel og tilnærmet perfekt system, og deretter skal vi ta for oss de begrensninger av antagelsene og bevege oss inn i mer og mer komplisert system som blir tilnærmet til realiteten. Vi skal se at ved bruk av noe matematiske metoder og fysikkens prinsipper, kan vi enkelt finne hastigheten til satelitten.

System 1: En kilde og en observatører i 1D

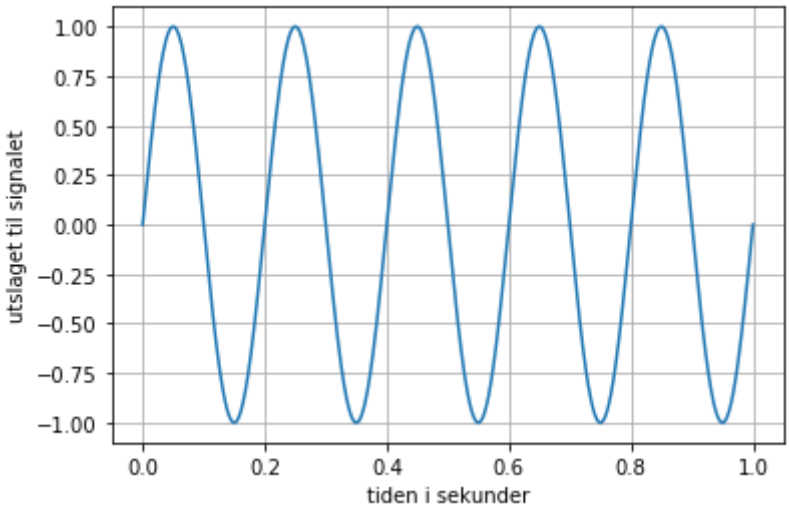
Vi kan først tenke oss og anta at vi befinner oss i en 1D plan. I denne planen finnes det to objekter. Vi kaller den ene for observatør, og den andre for kilde. Deretter tenker vi oss at kilden sender ut lysstråling i alle mulige retninger, og observatøren har et måleaparat som kan observere disse strålene.

Vi da modellerer et signal for denne lysstrålen. Vi antar at denne lysstrålet er på et enkel harmonisk form som er avhengig av tiden t. For nå antar vi at denne kilden stråler ut lys med frekvensen f på 5Hz.

$$\text{signal} = A\sin(2\pi ft)$$

A er amplituden til signalet.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

def sinus(t,f):
    """
    lager et enkel harmonisk signal på sinusform
    """
    return A*np.sin(2*np.pi*f*t)


A = 1 #definerer amplitude som 11
N = 1000 #Antall sted mellom i løpet av total tiden
T = 1 #tar opp signal i 1 sekund
"""
her setter vi N til å være 100, N bestemmer rett og slett hvor presis
vår data blir presentert. Vi kommer mer innpå etterpå hvor høy N
må være ved bruk av Nyquist regel.
"""
t = np.linspace(0,T,N) #definerer et tidsarray som går fra sekund 0 til 1 med


plt.plot(t,sinus(t, 5))
plt.xlabel("tiden i sekunder")
plt.ylabel("utslaget til signalet")
plt.grid(True)
plt.show()
```



Her antokk vi at observatøren tar målingene i 1 sekund.

Vi kan nå anta at kilden beveger seg relativt til oss. Hva kommer observatøren observere? Ifølge den Austriske fysikeren Christofer Doppler, kommer observatøren til å måle et bølge med høyere frekvens hvis kilden beveger seg mot obsevatøren, og lavere frekvens hvis kilden beveger seg fra obsevatøren.

Dette kan bli illustrert av .gif'en nedenfor.

Her kan vi se et bil som sender et konstant frekvens fra hans referanse system, men for oss (som ser dette fra side) ser vi akkurat samme frekvens i alle retningen når bilen står i ro, men med engang bilen begynner å bevege seg mot en retning, ser vi at det blir tettere mellom bølgetopper. Det vil si at jo kortere bølgelenge er, desto større er frekvensen.

Ved å vite frekvensen f_k som er sendt fra kildet, og observert frekvensen f_o, kan vi da finne ut hvor raskt kilden beveger seg relativt til oss!

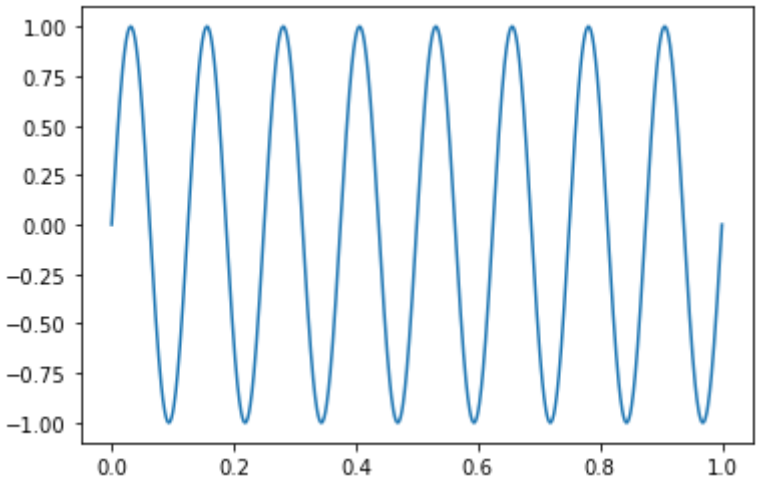
Relasjonen mellom hastigheten og frekvensene er oppgitt i uttrykk (1)

$$f_o = \frac{c}{c - v} f_k \tag{1}$$

Her er f_o frekvensen som observatør observerer, c er strålets hastighet, v er kildets hastighet relativt til observatør, og f_k er frekvensen som kildet sender ut.

For å teste dette ut, antar vi at strålen beveger seg med 100m/s, kildet sender fortsatt ut 5Hz, og at observatøren observerer dette signalet:

In [39]:
```
plt.plot(t, sinus(t,8))
plt.show()
```

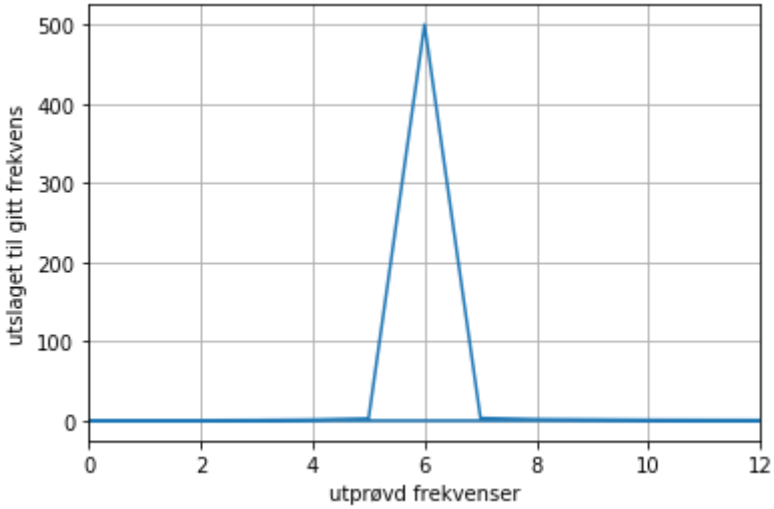


Hvordan kan vi bestemme frekvensen utifra dette signalet?

Hvis vi hadde for eksempel prøvd å multiplisere dette sinus funksjonen med annen sinus funksjon med annen frekvens, ville vi ikke få et stort utslag, men hvis vi multiplsierer dette sinus signalet med identisk sinus signal, ville vi ha fått stort utslag! Dette skal vi ikke teste her nå, men dette er noe som heter for Fourier transform! Vi rett og slett utfører flere slike multiplikasjon og tester ut forskjellige sinus signaler med forskjellige frekvenser, når vi prøver ut et frekvens som er identisk, vil vi få et stort utslag på det frekvensen. Vi rett og slett gjetter oss fram til riktig frekvens!

Vi har et innebygget modul i python som utfører fourier transform, vi velger å bruke denne.

In [40]:
```
#Vi begynner med å lage et eget array med alle signal punkter som vi kan samp
data = np.zeros(len(t))
for i in range(len(t)):
    data[i] = sinus(t[i], 6) #genererer et signal med frekvens på 6Hz

#og nå implementerer vi fourier transform:
dt = T/N
def fourier(data):
    data_fourier = np.fft.fft(data)
    freq = np.fft.fftfreq(N, dt) #Med dette lager vi bare et frekvens domen


    plt.plot(freq, abs(data_fourier)) #ved fourier bruker man komplekse tall,
    plt.xlabel("utprøvd frekvenser")
    plt.ylabel("utslaget til gitt frekvens")
    plt.xlim(xmin=0, xmax=12) #denne zoomer inn i plottet
    plt.grid(True)
    plt.show()
fourier(data)
```



Vi ser at utslaget er på 6Hz, dette må stemme siden vi har generert et signal på forhond som er på 6Hz, vi har derfor vist at fourier transform funker!

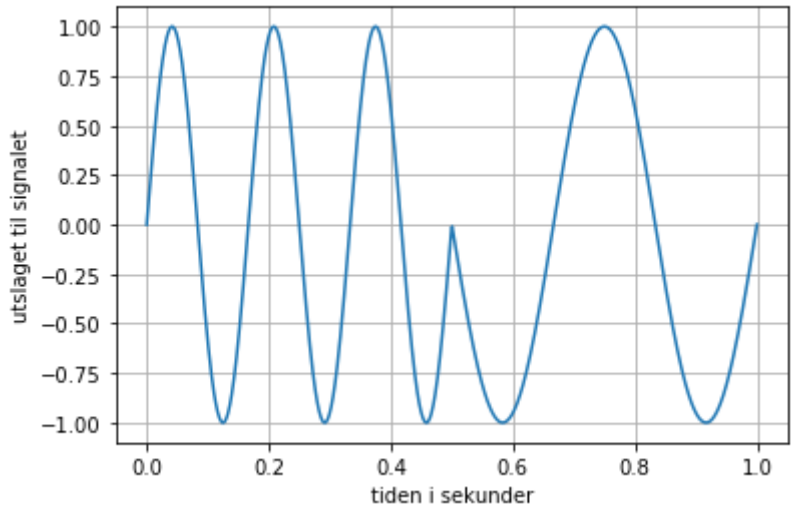
Etter at vi finner observert frekvens, ønsker vi å bruke det til å finne kildets hastighets verdi.

In [41]:
```
def vel(f_o, f_k):
    c = 100
    return abs(c * ((f_k/f_o) - 1))

print(vel(6, 5))
```

```
16.666666666666664
```

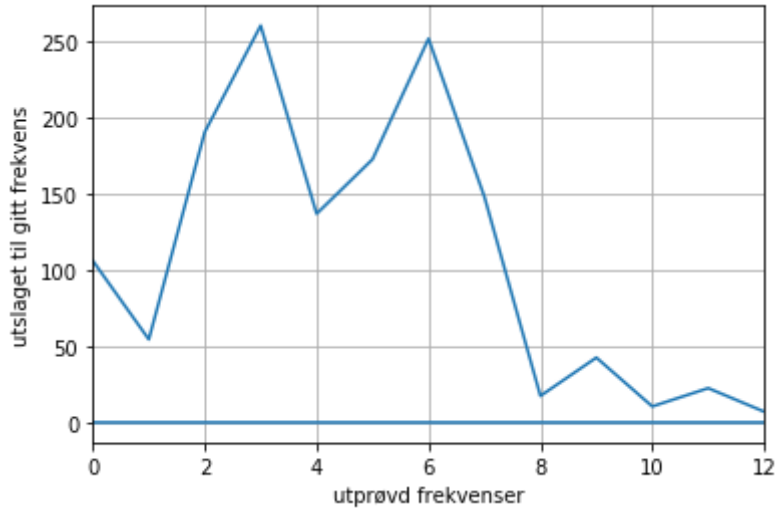
Resultatene sier at kildet beveger seg med ca 16.6 m/s! Vi ser at dette funker fint når kildet beveger seg med konstant hastighet, men hva ville skje hvis kildet hadde plutselig akselerert?! Hva hvis observatøren får inn et slik signal:

In [42]:
```
data1 = np.zeros(len(t))
for i in range(len(t)):
    """
    Later som at kilden akselererer momentant
    """
    if t[i] < 0.5:
        data[i] = sinus(t[i], 6)
    else:
        data[i] = sinus(t[i], 3)

plt.plot(t,data)
plt.xlabel("tiden i sekunder")
plt.ylabel("utslaget til signalet")
plt.grid(True)
plt.show()
```



Her ser vi at frekvensen plutselig mynker! Hva ser våres fourier transform?

In [43]:
```
fourier(data)
```



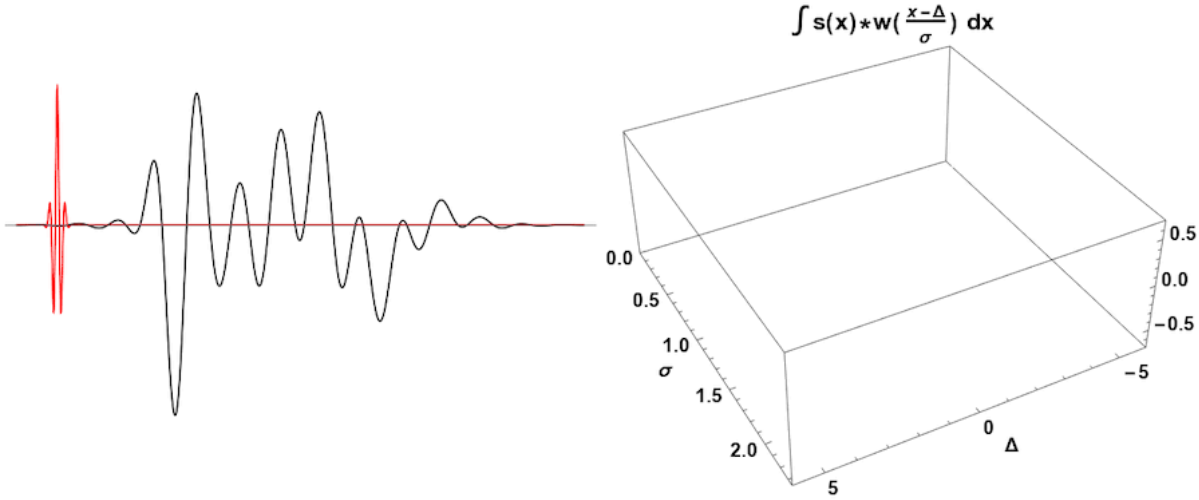
Her ser vi at vi har målt to frekvenser, frekvens på 3Hz og frekvens op 6Hz. Hva blir hastigheten?

In [44]:
```
print(f"vel_1: {vel(3,5)}, vel_2 = {vel(6,5)}")
```

```
vel_1: 66.66666666666667, vel_2 = 16.666666666666664
```

Her ser vi at under observasjonen har kilde hatt to forskjellige hastigheter, en på ca. 66.6m/s og en på ca 16.6 m/s. Her møter vi et begrensning av fourier transform, vi kan finne ut alle frekvenser og dermed alle hastigheter, men vi vet ikke når disse frekvensene har innslått, og dermed vet vi ikke om kilde har akselerert eller deakselerert relativt til oss.

For å kunne finne ut når visse frekvenser innslått i hvilke tidspunker, tar vi i bruk noe som heter wavelet-transform. Med wavelet transform, lager vi et wavelet med et viss bredde. Deretter velger vi et tilfeldig frekvens til denne waveletten, og lar den "scanne" seg gjennom hele signalet, vi registrerer da utslag i visse tider når wavelets frekvens er identisk til signalets frekvens. Etter at waveletten har scannet seg gjennom, endrer vi frekvenser til wavelet og gjør det samme. Gif'en nede illustrerer denne prossesen meget godt



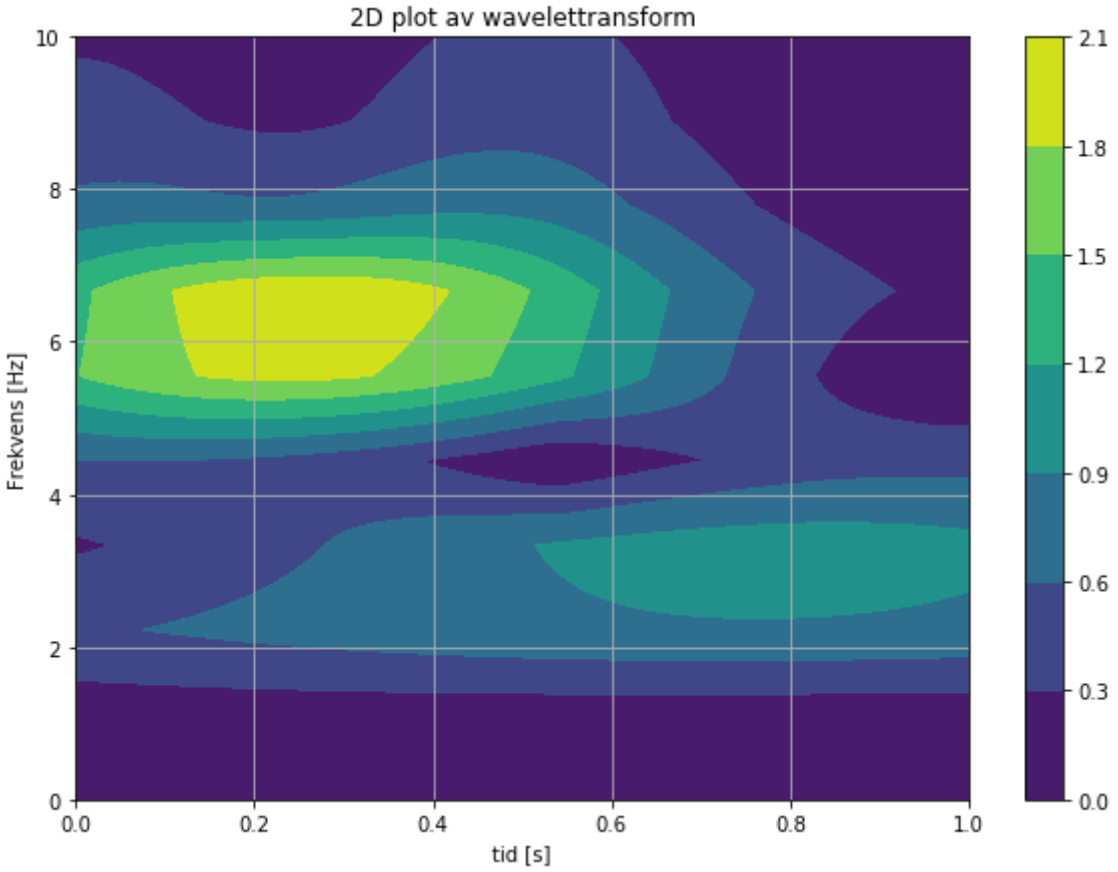
Dermed ved å lage to wavelets, en med frekvens på 3Hz, og annen med 8Hz, kan vi la dem scanne seg gjennom signalet of identifisere i hvilke tidspunker visse frekvenser gir utslag. Før vi skriver kode for dette, må vi innføre samplingsfrekvens. Samplingsfrekvens handler om hvor godt vi tilpasser signal inn i vår samplingen. Når vi får inn et digital signal, pleier vi å sette "punkter" der hvor signalet har verdier. Jo flere slike punkter det er, desto mer presis vi representerer signalet i vårt digital samplingen. Man kan tenke seg som om man ønsker å tegne et sirkel med punkter, og sette rette streker mellom disse punktene. Vi får meget unøyaktig sirkel ved å ha for eksempel 4 punker, men med flere punker vil våres sirkel se mer og mer presis ut. Samplingsfrekvens forteller dermed hvor stor mellomrom det er mellom disse punktene vi setter på signalet. Nyquist regel sier at samplingsfrekvens må være dobbelt så stor som frekvensen til signalet.

In [45]:

```
f_s = 5000 #samplingsfrekvens
"""
med samplingsfrekvens kan vi da finne antall punkter vi sampler med totalt he
med det igjen kan vi finne dt og utifra det bygger vi en ny tids array
"""
T = 1
N = f_s*T
dt = T/N
t = np.linspace(0,T,N)
print(N)

data = np.zeros(N)
for i in range(N):
    if t[i] < 1/2:
        data[i] = sinus(t[i], 6)
    else:
        data[i] = sinus(t[i], 3)



def wavelet_analyse(tmin, tmax, tsteps, fmin, fmax, fsteps):
        def wavelet(f_a, K, t_k, t_n):
            C = 0.798*2*np.pi*f_a/(f_s*K)
            eksp1 = -1j*2*np.pi*f_a*(t_n - t_k)
            eksp2 = -K**2
            eksp3 = (-2*np.pi*f_a**2)*((t_n - t_k)**2)/(2*K)**2
            return C*(np.exp(eksp1) - np.exp(eksp2))*np.exp(eksp3)

        t_ = np.linspace(tmin,tmax,tsteps)
        f_ = np.linspace(fmin,fmax,fsteps)
        T,F = np.meshgrid(t_,f_)
        K = 3
```

```
        sum = 0
        for i in range(N):
            sum = sum + np.conj(data[i]*(wavelet(F, K, T, t[i])))
        Z = abs(sum)
        plt.figure(figsize=([10,7]))
        plt.contourf(T,F,Z)
        plt.grid(True)
        plt.colorbar()
        plt.xlabel("tid [s]")
        plt.ylabel("Frekvens [Hz]")
        plt.title("2D plot av wavelettransform")

        plt.show()
wavelet_analyse(0, 1, 500, 0, 10, 10)
```

```
5000
```



Vi får denne 2D plotten og ser at vi starter med frekvens 6, og deretter går den over til frekvens 3. Det betyr at kildet har mynket hastigheten!

La oss nå rydde koden vårt slik at den blir enklere å bruke og at alt blir mer ryddig.

In [46]:

```
class Analyse:
    def __init__(self, data, T):
        """
        Input: data array og total analysens tiden T
        Output:
        """
        self.T = T
        self.N = data.shape[0]
        self.f_s = self.N/self.T
        self.dt = 1/self.f_s

        print(f"Data med {self.N} samplingspunkter er mottat...")
        print(f"Samplingsfrekvens er på: {self.f_s}Hz, data som har frekvens
```

```
        print(f"Varighet av data er på {self.T} sekunder")

        self.t = np.linspace(0,self.T,self.N)
        self.x_n = data


    def sampled_signal(self):
        plt.figure(figsize=([15,5]))
        plt.plot(self.t, self.x_n)
        plt.grid(True)
        plt.xlabel("tid [s]")
        plt.ylabel("amplitude")
        plt.title("Plot of sampled signal")
        plt.show()

    def fourier_transform(self, xmin, xmax):
        self.x_k = np.fft.fft(self.x_n)
        self.freq = np.fft.fftfreq(self.N, self.dt)
        plt.figure(figsize=([15,5]))
        plt.plot(self.freq, abs(self.x_k))
        plt.grid(True)
        plt.xlabel("frekvens [f]")
        plt.ylabel("amplitude")
        plt.title("Plot of fourier transform")
        plt.xlim(xmin=xmin, xmax=xmax)
        plt.show()

        return self.x_k
    def wavelet_analyse(self,tmin, tmax, fmin, fmax, steps):
        self.steps = steps
        def wavelet(f_a, K, t_k, t_n):
            C = 0.798*2*np.pi*f_a/(self.f_s*K)
            eksp1 = -1j*2*np.pi*f_a*(t_n - t_k)
            eksp2 = -K**2
            eksp3 = (-2*np.pi*f_a**2)*((t_n - t_k)**2)/(2*K)**2
            return C*(np.exp(eksp1) - np.exp(eksp2))*np.exp(eksp3)

        self.t_ = np.linspace(tmin,tmax,self.steps)
        self.f_ = np.linspace(fmin,fmax,self.steps)
        T,F = np.meshgrid(self.t_,self.f_)
        K = 10

        sum = 0
        for i in range(self.N):
            sum = sum + np.conj(self.x_n[i]*(wavelet(F, K, T, self.t[i])))
        self.Z = abs(sum)


        plt.figure(figsize=([10,7]))
        plt.contourf(T,F,self.Z)
        plt.grid(True)
        plt.colorbar()
        plt.xlabel("tid [s]")
        plt.ylabel("Frekvens [Hz]")
        plt.title("2D plot av wavelettransform")
        plt.show()

    def hastighet(self,f_o, c):
        frek = np.zeros(self.steps)
        tid = self.t_
        vel = np.zeros(self.steps)

        def v(f_k):
            return (c * ((f_k/f_o) - 1))
```

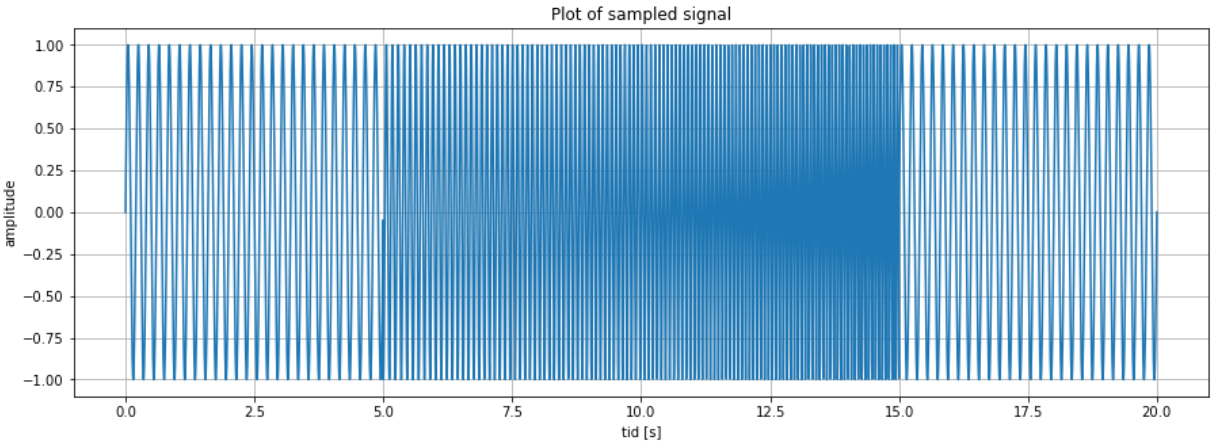
```
        for i in range(self.steps):
            index = np.where(self.Z == np.amax(self.Z[:,i]))
            monoindex = index[0]
            frek[i] = self.f_[monoindex]
            vel[i] = v(frek[i])
        plt.figure(figsize=([10,7]))
        plt.plot(tid, vel)
        plt.plot(tid, vel, 'o')
        plt.grid(True)
        plt.xlabel("tid [s]")
        plt.ylabel("Hastighet [m/s]")
        plt.title("Hastighetsplot")
        plt.show()
```

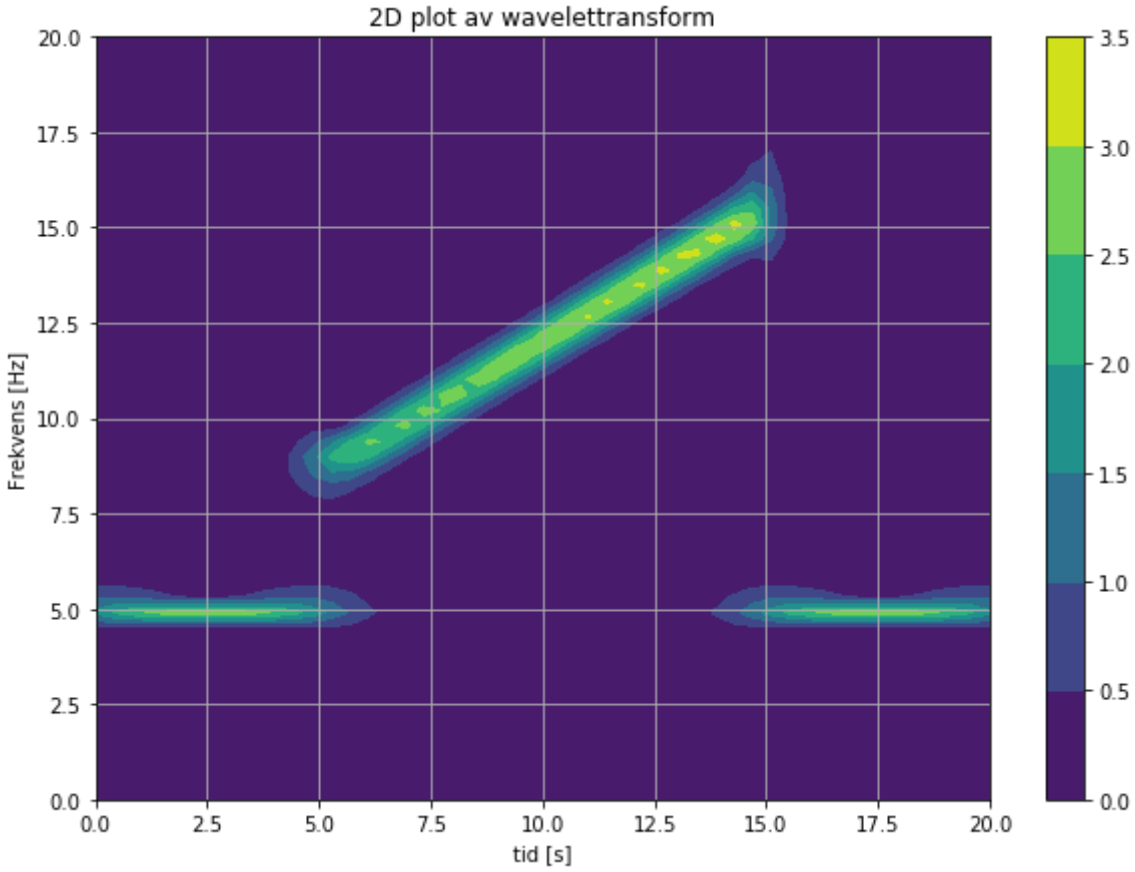
Da har vi skrevet en klasse som tar i mot observert data, og finner hastighets kurven til kilden som sender ut lyset! La oss prøve den ut ved å analysere litt mer komplisert signal.

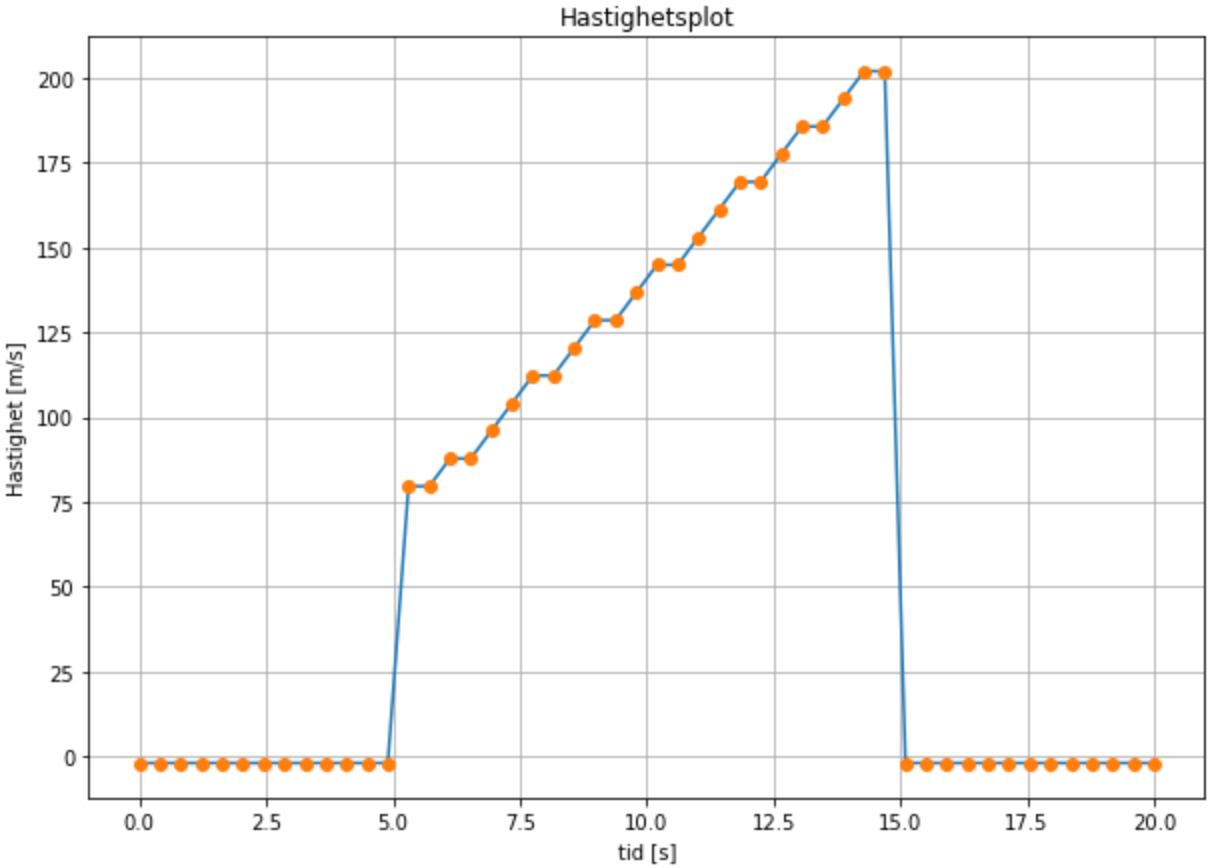
La oss tenke at kilden begynner fra ro, og begynner akselerer mot observatøren og plutselig brå bremser ned til ro igjen i løpet av 20 sekunder tidsperiode:

In [47]:
```
N = 10000
t = np.linspace(0,20,N)
data = np.zeros(N)
inc = 0
for i in range(N):
    if t[i] < 5:
        data[i] = sinus(t[i], 5)
    elif 5 < t[i] < 15:
        freq = 5 + inc
        data[i] = sinus(t[i], freq)
    else:
        data[i] = sinus(t[i], 5)
    inc = inc + 0.0007


inst = Analyse(data, 20)
inst.sampled_signal()
inst.wavelet_analyse(0, 20, 0, 20, 50)
inst.hastighet(5,100)
```

```
Data med 10000 samplingspunkter er mottat...
Samplingsfrekvens er på: 500.0Hz, data som har frekvens mindre enn 250.0Hz bl
ir ikke registrert
Varighet av data er på 20 sekunder
```


Plot of sampled signal
amplitude
tid [s]


2D plot av wavelettransform
Frekvens [Hz]
tid [s]



Her ser vi at vi får imponerende gode resultater! Verdiene i dette systemet er urealistiske, dette er på grunn av at vi nøyer oss på å få dette scripte til å funke. Vi ser i tillegg at verdiene er ikke akkurat på de verdiene vi har modellert. For eksempel hastighet er ikke akkurat null når signalet er på 5Hz, for å få det mer nøye må vi rett og slett ofre mer datakapasitet og legge til flere timesteps mellom aller verdiene som simulasjonen prøver ut.

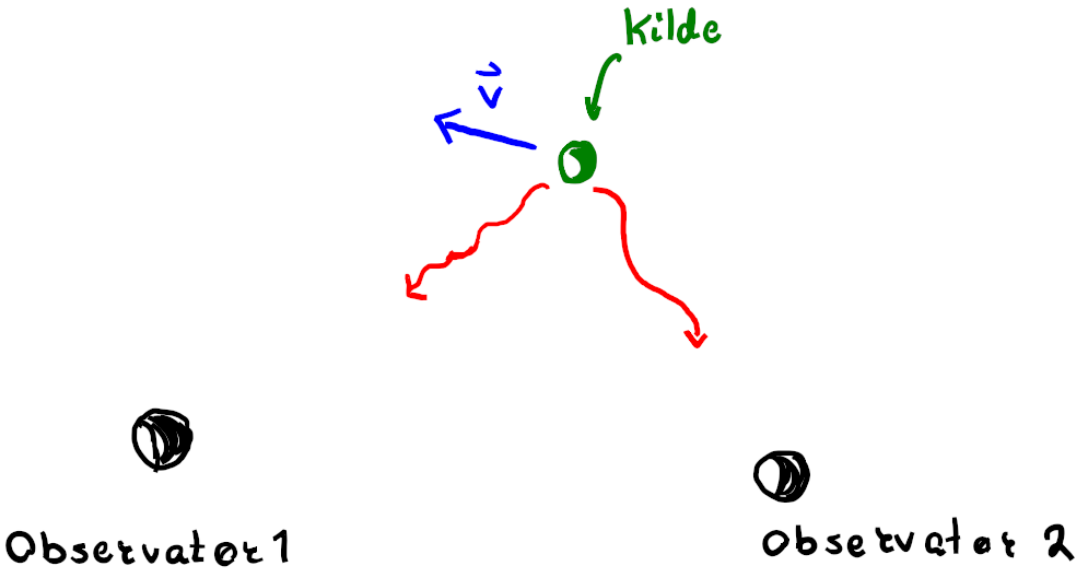
Det siste plottet i vår tilfelle er viktigst, på plottet har vi hastighetsutvikling gitt i tid. Utifra det plottet kan vi da finne akselerasjon til kildet ved å finne veksten til hastigheten per tidsenhet. I det siste signalet kunne vi for eksempel tatt linær regresjon mellom $t = 5$sek og $t = 15$sek og finne stigningstallet til denne linære regresjonen. Stigningstallet ville være da akselerasjon.

Vi kan i tillegg finne ut hvor strekning kildet har nådd ved å finne arealet under kurven, det kan vi oppnå ved å integrere over hele kurven! Dette er særlig viktig for våres problemstilling, siden vi ønsker å vite hvor vår satelitt befinner seg. Vi kan da for eksempel installere et system i satelitten som merker når kommunikasjon er brutt, når satelittens elektronikk ser det, kan den begynne å sende ut lysstråler, samtidig som at basene på jorda begynner å samle inn data. Slik kan basene begynne å "følge med" på hastighetsretningene for satelitten, og dermed vite hvor den er.

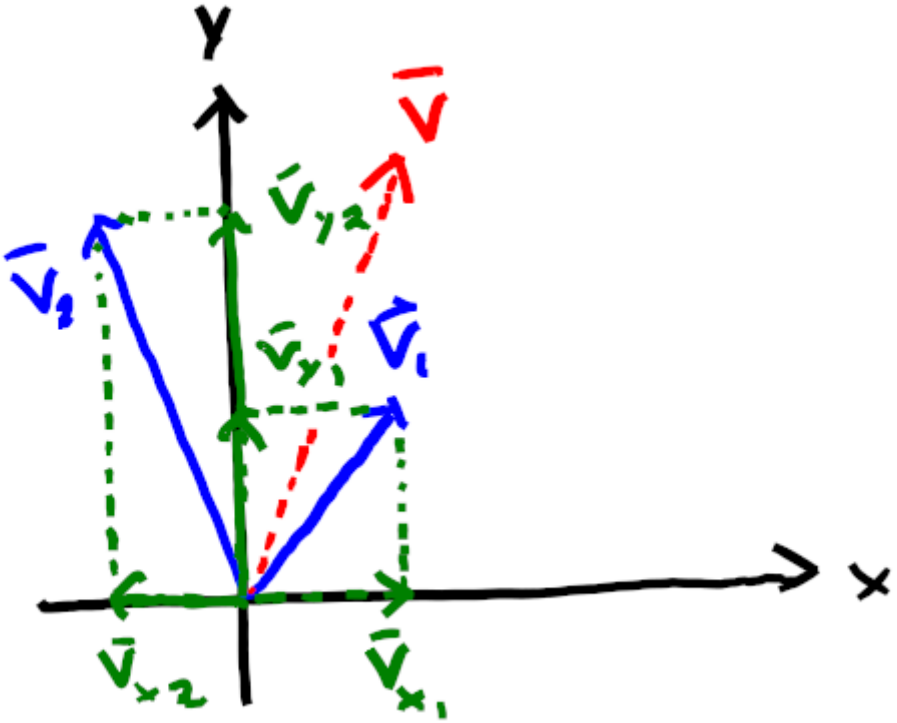
Vi skal nå se på et nytt system, et system som er mer komplisert en denne som vi har hatt nå. Vi skal nå flytte oss inn i 2D verden.

System 2: Et kilde og to observatører i 2D

Vi tenker oss at vi finnes i et 2D plan, og at her finnes det tre objekter, et kilde og to observatører. Kildet sender ut stråling, mens disse to observatører måler denne strålingen.

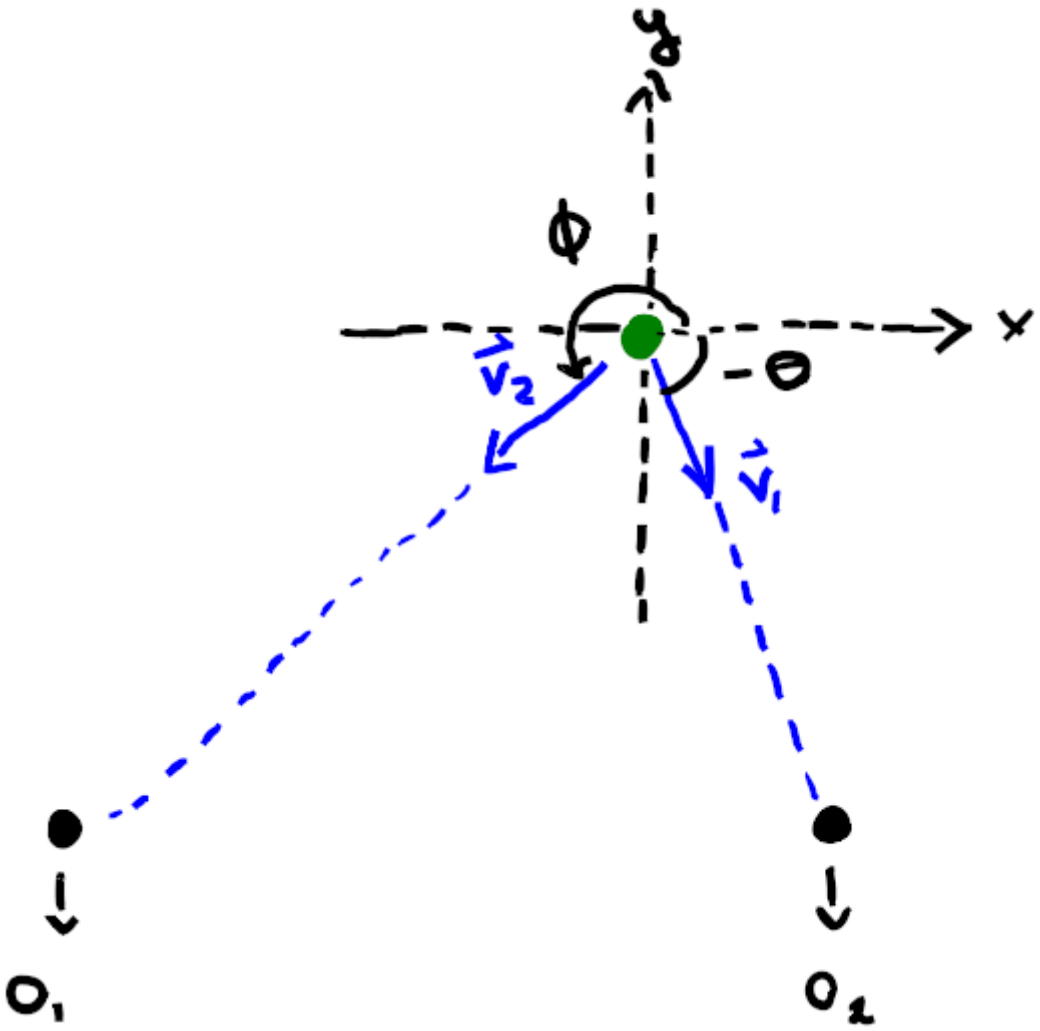


I dette systemet kan kildet ha hastighetsretning i uendelig mange retninger. Hvis vi hadde hatt bare en observatør som i det forrige systemet, ville vi ha visst hastighet har bare det radielle retningen for observatøren. Vi hadde altså bare visst hvor fort kildet beveger seg mot eller fra observatøren. Ved å ha to observatører, kan vi da måle to hastighetskomponenten, og utifra disse to kan vi da finne det riktige hastighetsretning til kildet ved bruk av vektorkalkulus.



Her antar vi at vi har et koordinatsystem, og måler to hastighetsvektorer (blått), dermed kan vi dekomponere disse to vektorene (grønt) og bygge det faktiske hastighetsvektor utifra de (rødt).

I vårt modell, antar vi at før kommunikasjonen er brutt mellom basen og satelitten, registrer vi hastighet, posisjon og vinkelen θ og ϕ slik figuren nedenfor viser. Når kommunikasjonen er plutselig brutt, bruker vi da sist registrerte verdiene for hastighet, posisjon og vinklene som initialbetingelse slik at vi kan "følge bevegelsen til satelitten".



Her definerer vi et kartesisk koordinatsystem hvor vi setter satelitten inn i origo. Vi kan nå modellere et signal og anta at vinklene var på $\theta = -30$ grader og $\phi = 190$grader når kommunikasjonen ble brutt.

Her skal vi ikke gjøre en full utregning til å finne løsningen siden da blir denne computational essay for lang. Hvis kildet hadde for eksempell hatt hastigheten $\vec{v} = 200\hat{i} + 100\hat{j}$ i kartesisk koordinat system. Hva vil observatorene våres observere da?

Siden vinkelen θ og ϕ oppdateres hele tiden, vil også koordinatsystem for observatørene endres hele tiden også. Dette kan vi finne ut ved bruk av linær algebra.

Basiser i kildets koordinatsystem (kartesiske) er

$$\vec{e_1} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

og

$$\vec{e_2} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

For å finne ut hva observatørene observerer, må vi bytte basis fra kildets til observatørens. For å gjøre det, skalerer vi bare kildets basiser:

$$\vec{b_1} = c_1\vec{e_1} + c_2\vec{e_2}$$

For c_1 og c_2 bruker vi enkel trigonmetri,

$$c_1 = \cos(\theta)$$

$$c_2 = \sin(\theta)$$

Tilsvarende for $\vec{b_2}$

$$\vec{b_2} = c_3\vec{e_1} + c_4\vec{e_2}$$

når

$$c_1 = \cos(\phi)$$

$$c_2 = \sin(\phi)$$

Dermed et vektor $\vec{v}$ i observatørenes koordinatsystem blir nå

$$\vec{v}_{observ.} = 200\vec{b_1} + 100\vec{b_2}$$

Dermed får vi,

$$\begin{bmatrix} 200\cos(-30) + 100\cos(190) \\ 200\sin(-30) + 100\sin(190) \end{bmatrix} = \begin{bmatrix} 75 \\ -117 \end{bmatrix}$$

Da ser vi at farta er på 75m/s langs $\vec{b_1}$ enhetsvektor og -117m/s langs $\vec{b_2}$ enhetsvektor. Det vil si at observatør 1 observerer 75m/s, og observatør 2 observerer -117m/s. Ved å få rå data fra disse observatørene, kan vi gjøre da alt dette baklens og finne hastighetsvektor i kartesiske koordinater når satelitt er i origo. Alt dette avhenger av hvordan vi selv ønsker å definere koordinatene. Det å bytte mellom basiser hadde vært kanskje litt for komplisert og for mye arbeid som da koster datakapasistet og tid. Det kunne for eksempel lønnet seg å sette jordas sentrum i orgio og dermed jobbe med de forskjellige vektorere og bruke vektorkalkulus til å finne hastighet til satelitten relativt til jorda.

Her velger vi å ikke skrive et script som løser alt dette. Meningen med dette var å starte et slags tanke for hvordan eventuell reel system som løser dette kunne fungere.

Signal som går gjennom to medier

Nå har vi sett på to systemer, den første var for å bygge et script og metode for å finne hastighets plot. Utifra denne plotten kan vi som nevnt finne både akselerasjon (ved å finne stigningene) og posisjon (ved å ta integralet). Når vi finner arealet under plottet, finner vi da hvor langt satelitten har beveget seg.

En annen ting er at signalet reiser mellom to medier. Det starter i vakuum og ender i luft. Heldigvis endres ikke frekvensen når elektromagnetisk bølge reiser mellom to medier. Allikevel, endres hastigheten. Dette skaper oss et problem, siden når kommunikasjonen blir brutt, vil det gå noe tid før signalet kommer til basene, og i denne tiden er vi i et slags blindsone. Dermed er det best å velge et elektromagnetisk bølge som et signal, siden den reiser med lyshastighet i vakuum, og tilnærmet lyshastighet i luft.

En annen ting som vi er heldig for, er at signalet blir går fra en medium med lav brytningsindeks, til høyrer brytningsindeks. Hvis det hadde vært omvendt, ville vi oppleve kritiske vinkler. Det vil

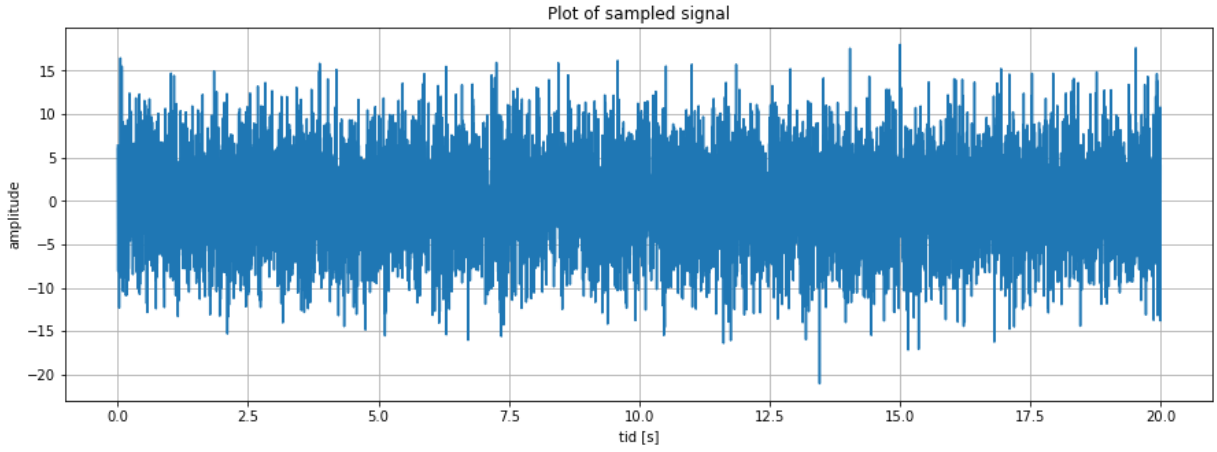
si at i visse vinkler, vil all signal bli reflektert, og dermed får vi ikke observere noen ting. Vi opplever ikke kritisk vinkel når vi går fra lav brytningsindeks til høy brytningsvinkel.

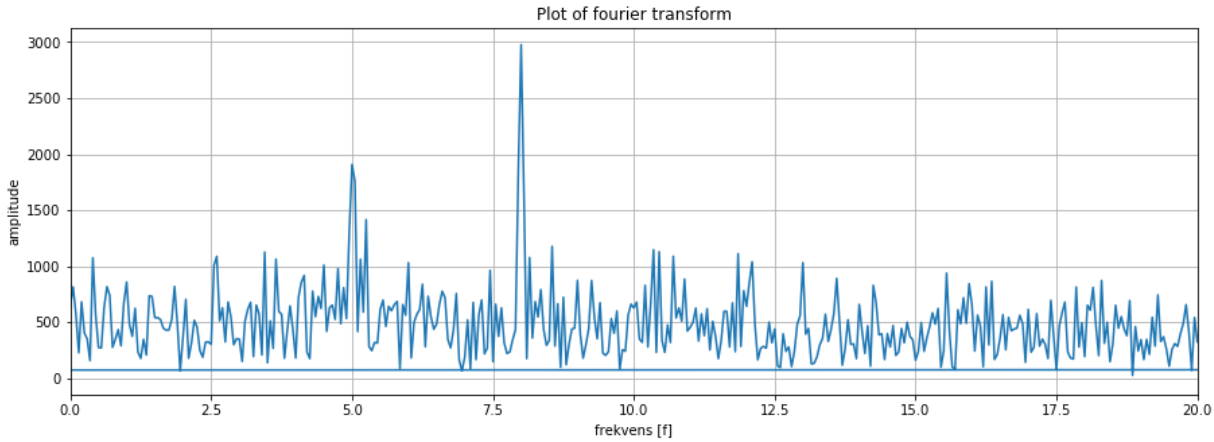
Støy

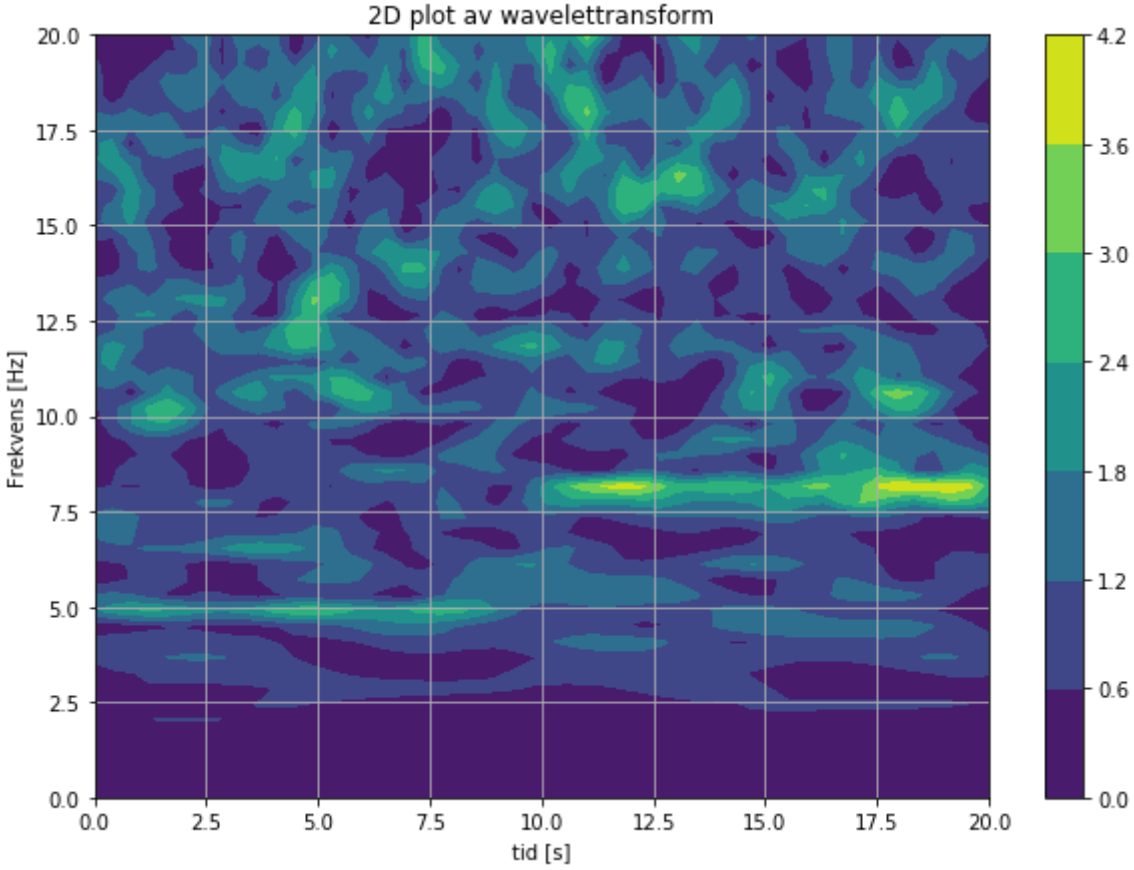
Når basene mottar signal, vil signalet mest sannsynlig være støyete, og dermed lønner det seg å lage et script som da fjerner denne søyen. Vi velger å generere samme signalet som vi har hatt i første systemt vårt, men denne gangen, legge inn litt gaussisk støy.

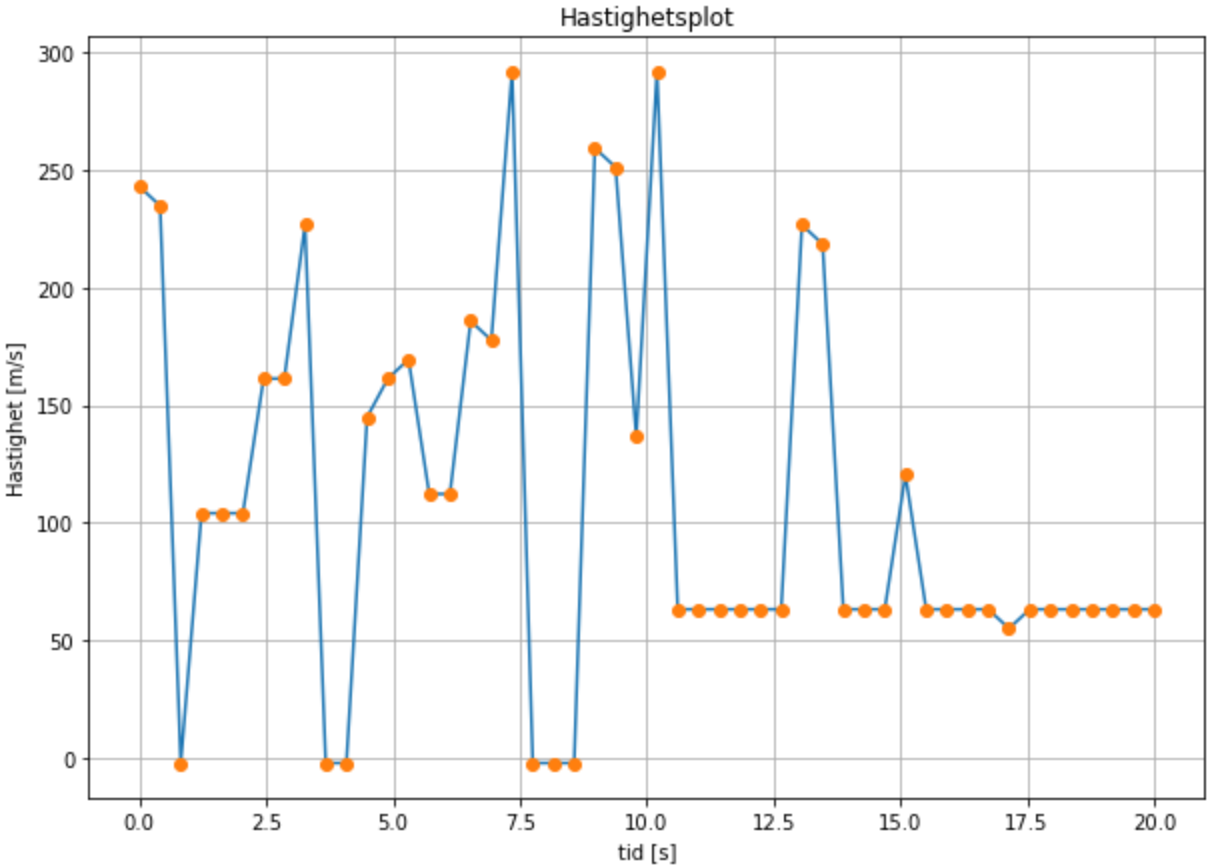
In [48]:
```
N = 10000
t = np.linspace(0,20,N)
data = np.zeros(N)
inc = 0
for i in range(N):
    if t[i] < 10:
        data[i] = sinus(t[i], 5)
    else:
        data[i] = sinus(t[i], 8)
    inc = inc + 0.0007

noise = np.random.normal(0,5,data.shape)
noisy_data = data + noise

inst = Analyseinspace(0,20,N)
data = np.zeros(N)
inc = 0
for i in range(N):
    if t[i] < 10:
        data[i] = (noisy_data, 20)
inst.sampled_signal()
inst.fourier_transform(0,20)
inst.wavelet_analyse(0,20,0,20,50)
inst.hastighet(5,100)
```

```
Data med 10000 samplingspunkter er mottat...
Samplingsfrekvens er på: 500.0Hz, data som har frekvens mindre enn 250.0Hz bl
ir ikke registrert
Varighet av data er på 20 sekunder
```




Plot of fourier transform
amplitude
frekvens [f]
0
500
1000
1500
2000
2500
3000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


2D plot av wavelettransform
Frekvens [Hz]
tid [s]
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0.0
0.6
1.2
1.8
2.4
3.0
3.6
4.2

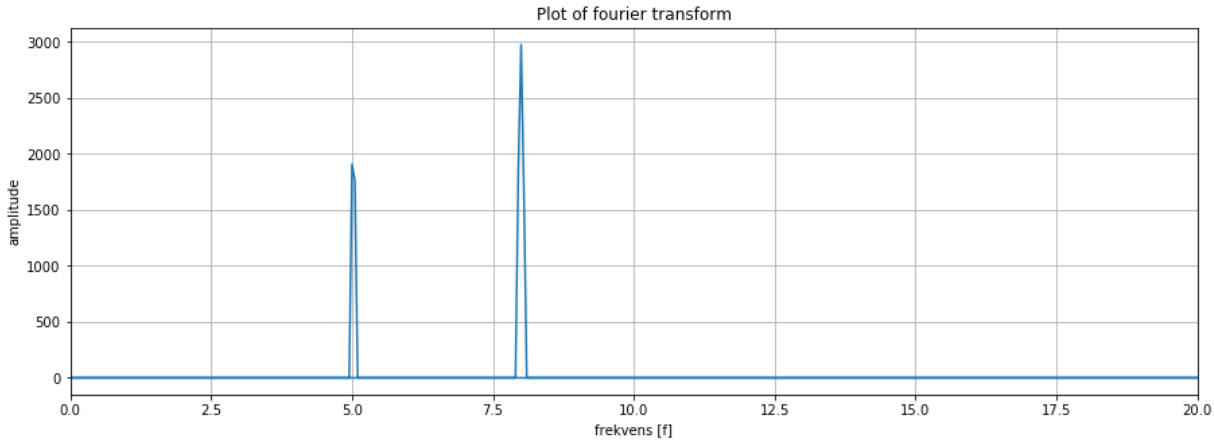


Her ser vi i våres plottene at vi ikke klarer å få ut noe nyttig informasjon. På wavelet transform plottet kan vi nesten se såvidt at de første 8 sekundene registrerer observatør 5Hz. Allikevel programmet vårt sliter med å lese det. Vi må dermed komme med et plan slik at vi kan rense bort all det støyet. Eneste plott som viser oss noe nyttig, er nemlig fourier transformasjons plottet. Der kan vi se at vi har to topper som er høyest. Nemlig en på 5Hz og annen på 8Hz.

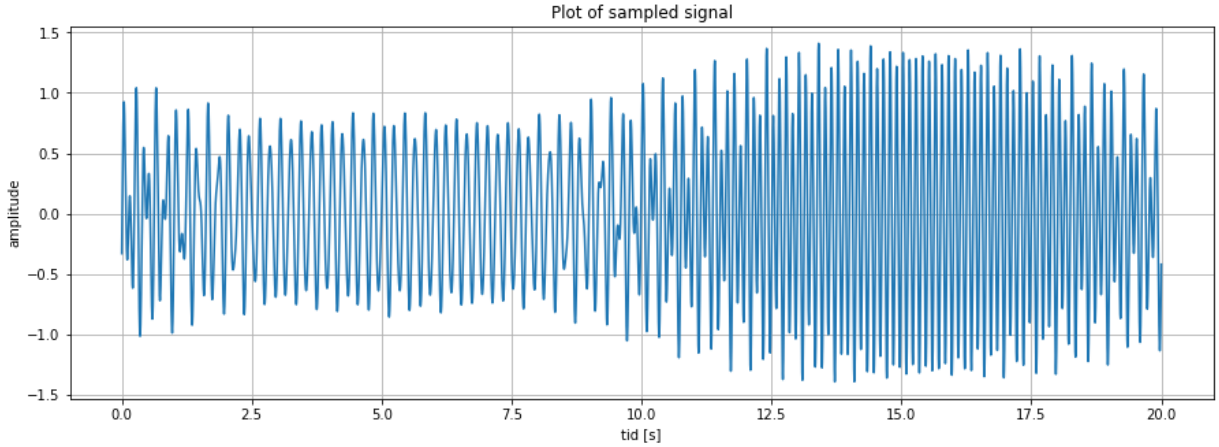
Hvis vi nå sier til programmet, finn disse to høye toppene, og fjern alle andre topper, også ta inverst fourier transform!

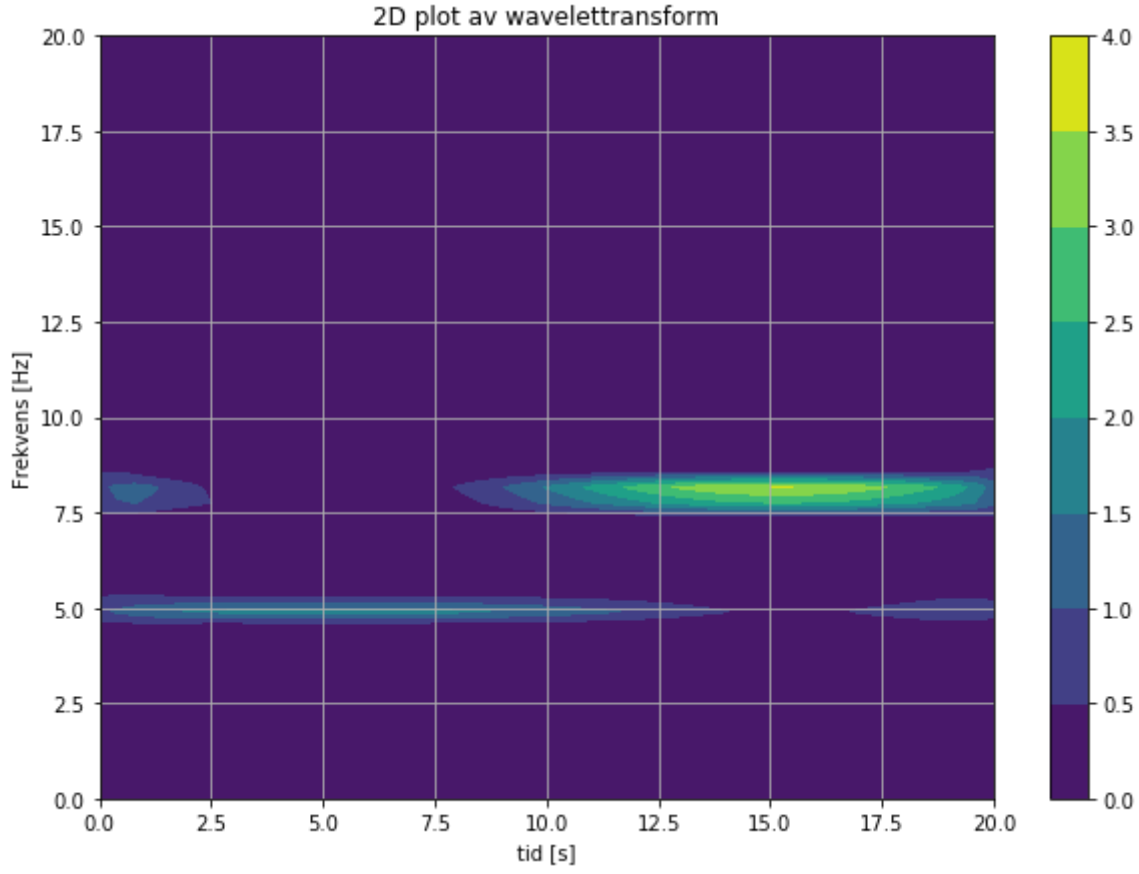
Hvis vi for eksempel velger å sette alle frekvenser lik null som har mindre amplitude enn 1500.

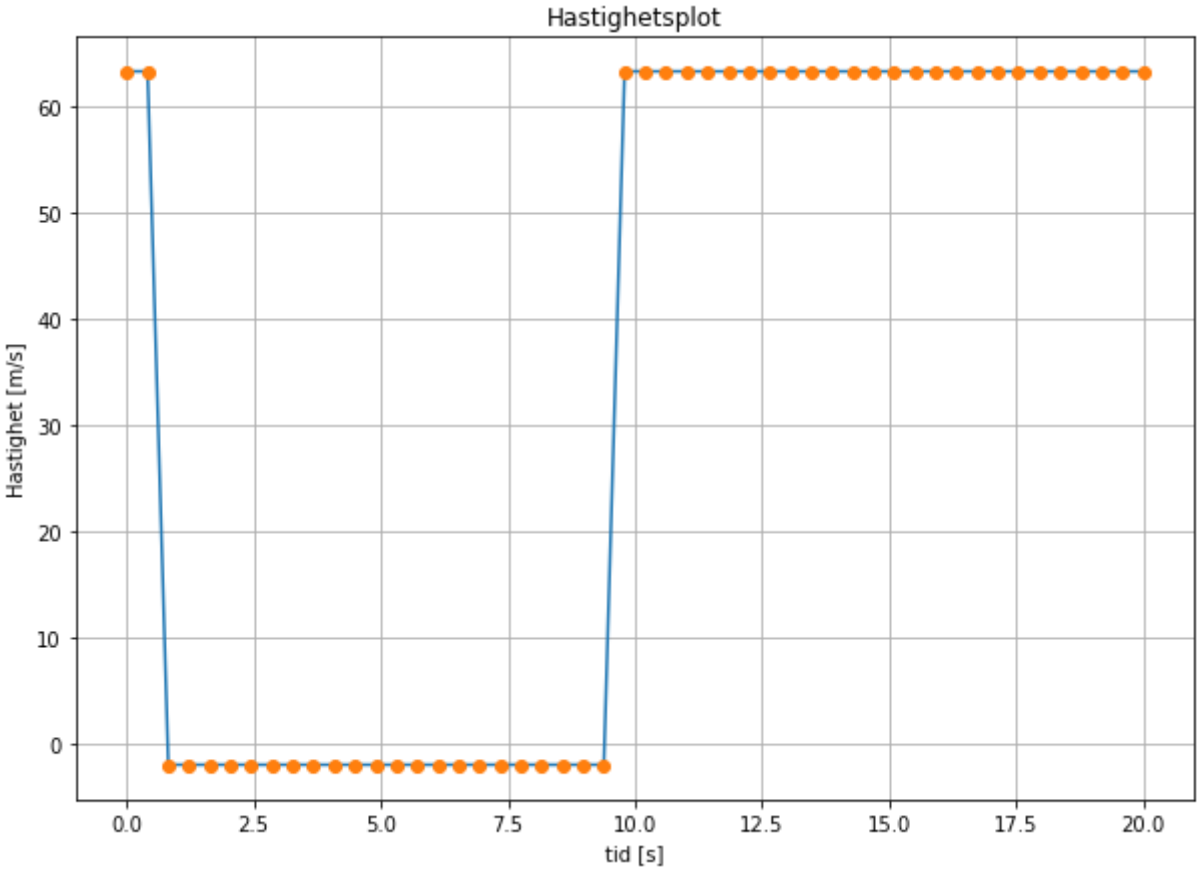
In [57]:
```
min_amp = 1500
x_k = inst.fourier_transform(0,20)

indices = abs(x_k) > min_amp
x_k_new = indices * x_k

n_x_new = np.fft.ifft(x_k_new)

inst = Analyse(n_x_new, 20)
inst.sampled_signal()
inst.wavelet_analyse(0,20,0,20,50)
inst.hastighet(5,100)
```




```
Data med 10000 samplingspunkter er mottat...
Samplingsfrekvens er på: 500.0Hz, data som har frekvens mindre enn 250.0Hz bl
ir ikke registrert
Varighet av data er på 20 sekunder
```







Da kan vi se at dette fungerte overaskende bra!

En annen ting vi burde tenke på, er hvordan vi skal kunne skylde det spesifike signalet fra alle andre signaler som er overalt. Løsningen til det kan da være for eksempel det at satelitten skal sende ut wavelets med bestemte mellomrom, eller et slags wavelets rekke mønster slik at vi kan da lettere skylde det fra alle andre signaler!

Konklusjon

Hvordan kan vi få informasjon fra satelitten når kommunikasjon mellom satelitten og basene på jorda er brutt? Vi har sett nå at ved å sende et elektromagnetisk signal på bestemt waveletform, kan vi bruke Doppler's skift til å bestemme hastighets retningene ved hjelp av Fourier analyse og litt linær algebra. Ved å vite hasitghetsutviklingen, kan vi deretter også finne ut hvor langt satelitten har reist etter at kommunikasjonen ble brutt. Itillegg kan vi finne ut akselerasjon. Vi har tatt for oss hvordan vi kan fjerne eventuell støy fra signalet, og tatt hensyn til signalet som går gjennom to medier.

Denne computational essay inneholder få betraktninger for våres problemstillingen, ved å fordype oss enda mer i denne modellen kan vi støtte på mange flere problemer som vi må løse. Det kan være for eksempel at vi kan miste signalet på veien, eller at vi ikke klarer å fjerne støy fra signalet, og dermed må implimentere et bedre metode for å fjerne støy. I tillegg ved store hastigheter må vi ta hensyn til relativitet og mye mer! Alt i alt, våres modell er langt fra til å være perfekt.