Hjemmeopgave 4

34210 Digital Kommunikation

Daniel Brasholt s214676

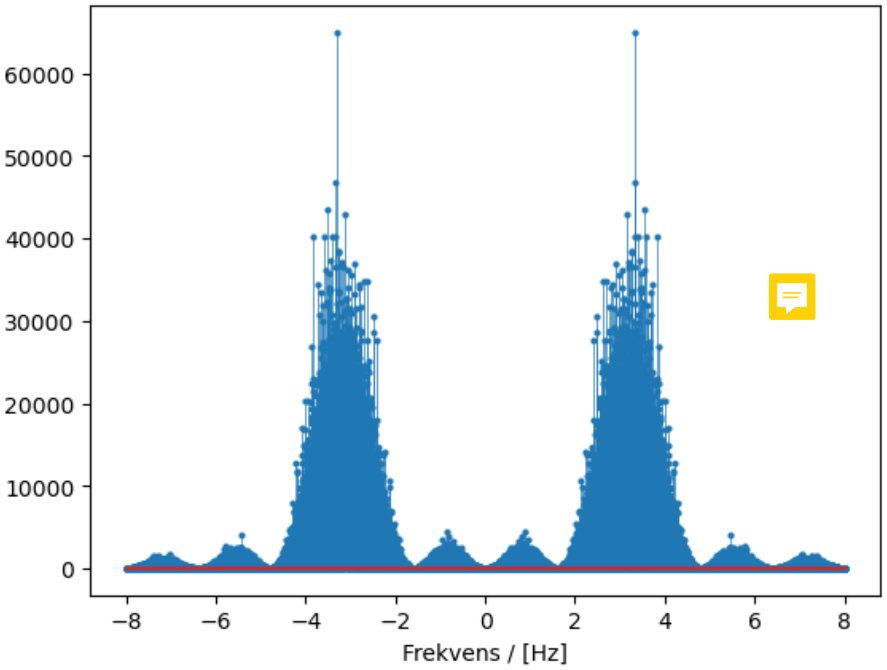
Maj 2023

Indhold

Spørgsmål 1 Ortogonale signaler

Ifølge formel (6.9), skal den givne sum være 0 for alle $n \neq p$. Kun for $n = p$ skal summen give noget forskelligt fra 0. Som det fremgår af bilagene, er dette netop opfyldt. Summen giver ikke præcis 0, men da både realdel og imaginærdel af tallet er af størrelsen 10^{-15}, kan det siges at skyldes afrundingsfejl i værktøjet.

Spørgsmål 2 Frembringning af signalet v



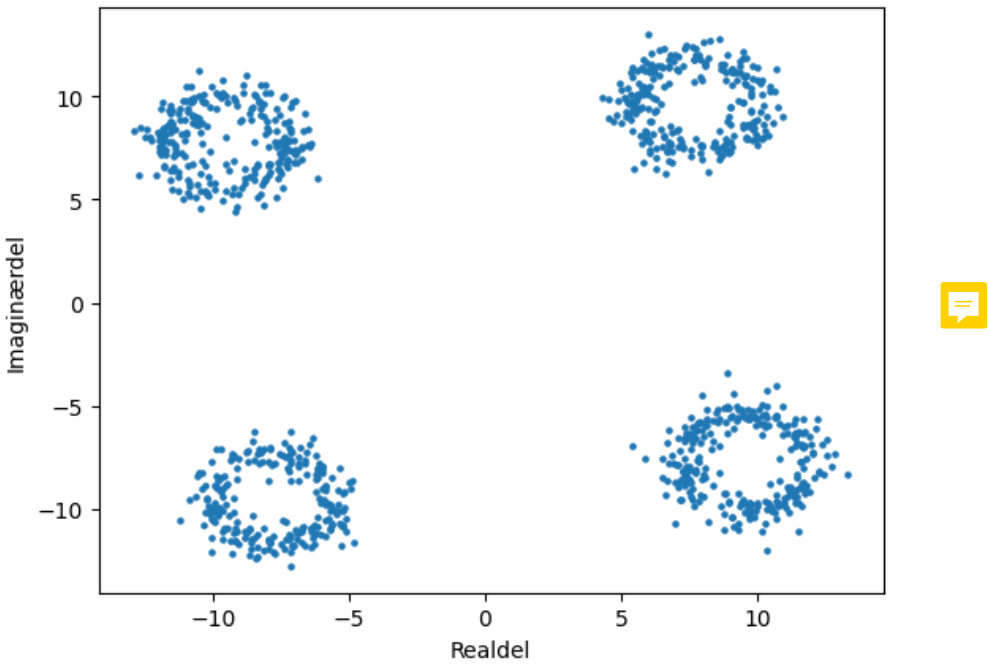
Figur 1: *Energispektret af v*

Ovenstående figur 1 viser energispektret for v. Her er givet 2 toppe, hvilket giver mening, da vi modulerer rundt om en bærebølge. De to toppe er centreret omkring ± 3. Derudover indeholder signalet mange mindre bidrag helt til ± 8, hvilket giver mening, da Fourier-serien for en firkantet puls vil være uendeligt lang med bidrag, der bliver uendeligt små.

Spørgsmål 3 Modtageren

Spm 3.1: Display af 1 kanal

Her er det valgt at vise kanal nr. 5. De andre ligner dog denne:



Figur 2: *Kanal 5 med realdel på x-aksen og imaginærdel på y-aksen*

Her viser der sig 4 konstellationer, hvilket passer med en sædvanlig QPSK-konstellation. Mærkværdigt er det, at konstellationerne ikke er perfekte punkter som forventet i en støjfri transmission, men derimod egentlige konstellationer. Dette kan muligvis skyldes mærkelige afrundinger i de komplekse tal eller det, at NumPy alligevel giver værdier til 0-bidrag, som nævnt i spørgsmål 1. Dog gør det her ikke en forskel, da vi blot kan kigge på fortegnet af real- og imaginærdel og på den måde gennemskue det symbol, der blev sendt.

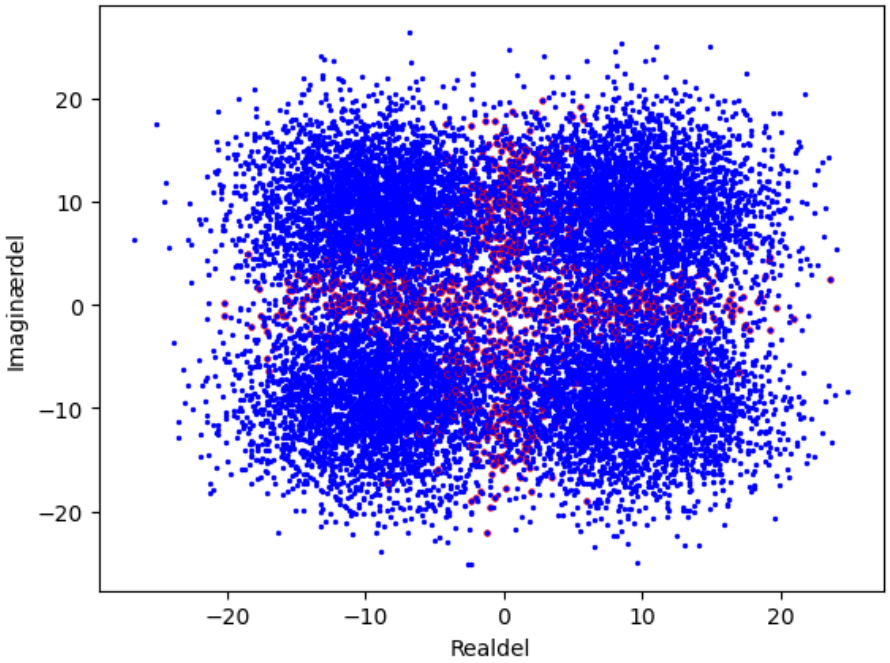
Spm 3.2: Gendannelse uden fejl

I bilagene er det vist, hvordan man, ved kun at kigge på fortegnet for realdel og imaginærdel af hver kanal for hver k, kan gendanne `a_k`. Dette er gjort ved at tælle antallet af steder, hvor der er forskel på den genskabte og den sendte.

Spørgsmål 4 Signalet med støj

Spm 4.1: Addering med støj

Nedenstående figur 3 viser konstellationerne for de modtagne symboler, når der er adderet med normalfordelt støj, $\sigma = \sqrt{2}$.



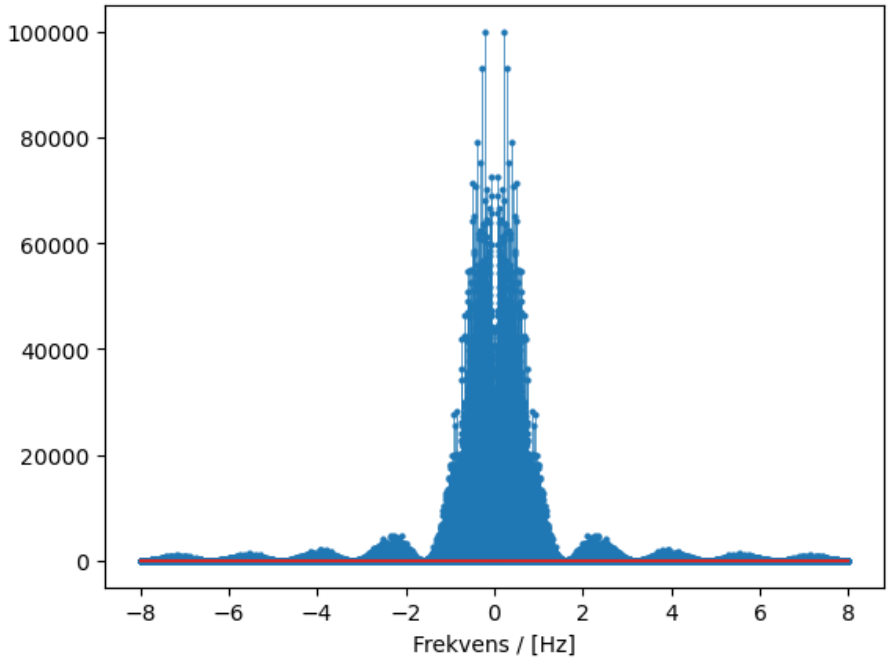
Figur 3: *Konstellationer for alle kanaler med de forkerte markeret med rødt*

Spm 4.2: Antal fejl med støj

Som i spørgsmål 3.2 er antallet af fejl talt med NumPy. Dette har givet antallet 1020, hvilket er en faktor 2.5 højere end det forventede, $1000 \cdot 16 \cdot 2 \cdot \texttt{qfunc}\left(\sqrt{\frac{10}{2}}\right) \approx 406$. Forklaringen på dette kan muligvis være den samme forklaring som i spørgsmål 3.1, hvor konstellationerne ikke var perfekt centrerede på trods af støjfri transmission. Der gjorde det ingen forskel, men da punkterne er kommet væsentligt tættere på hinanden, kan det have gjort en forskel nu.

Spørgsmål 5 Simulering af ADSL

Spm 5.1: Energispektrum for v



Figur 4: *Energispektrum for v*

Ovenstående figur 4 viser energispektret for den nye v. Her kan ses størst bidrag omkring $\pm 2Hz$, dog igen med bidrag, der bliver uendeligt små, men fortsætter uendeligt længe, på grund af den firkantede pulsform.

Hjemmeopgave 4

May 3, 2023

[84]:
```
import numpy as np
from matplotlib import pyplot as plt
from scipy.stats import norm
```

1 Spørgsmål 1

[85]:
```
T = 1
L = 16
q = np.arange(0,L)
p = q
m = 10
Ts = T/L
t = q*Ts
```

[86]:
```
s = np.zeros((L,L), dtype=np.cdouble)
for i in range(L):
    s[i] = np.exp(1j*2*np.pi*(i/T)*t)
```

[87]:
```
n = 8
for p in range(L):
    print(sum([Ts*gnk*np.conj(gpk) for gnk,gpk in zip(s[n],s[p])]))
```

```
2.83276944882399e-16j
(-9.71445146547012e-17+9.71445146547012e-17j)
(1.0408340855860843e-16+1.5265566588595902e-16j)
(-1.97758476261356e-16+6.522560269672795e-16j)
(-3.8353333219367193e-16+4.163336342344337e-17j)
(-4.198030811863873e-16-6.938893903907228e-16j)
(-1.1796119636642288e-16-2.0122792321330962e-16j)
(-1.5959455978986625e-16-5.898059818321144e-17j)
(1+0j)
(2.220446049250313e-16+7.632783294297951e-17j)
(-8.187894806610529e-16-3.400058012914542e-16j)
(-1.5265566588595902e-16-7.355227538141662e-16j)
(1.2887400986479702e-15-5.551115123125783e-17j)
(-1.9081958235744878e-16+1.4988010832439613e-15j)
(-2.498001805406602e-16+5.204170427930421e-16j)
(5.342948306008566e-16+2.185751579730777e-16j)
```

2 Spørgsmål 2

[88]:
```
symbol_set = [-1,1]
a_k = np.zeros((L,1000), dtype=np.cdouble)
x = np.random.choice(symbol_set, size=(L,1000), replace=True)
y = 1j * np.random.choice(symbol_set, size=(L,1000), replace=True)
a_k[::] = x+y
```

[89]:
```
y_k = np.zeros(L*1000, dtype=np.cdouble)
for k in range(a_k.shape[1]):
    y = np.sqrt(L)*np.fft.ifft(a_k[:,k].T)
    y_k[k*L:(k+1)*L] = y
```

[90]:
```
m = 10
#h = np.array([0,0,0,0,1,1,1,1,0,0,0,0])
h = np.ones(10)
```

[91]:
```
i0 = np.zeros(y_k.size*m)
y_real = np.real(y_k)
i0[::m] = y_real
i = np.convolve(h,i0)

q0 = np.zeros(y_k.size*m)
y_imag = np.imag(y_k)
q0[::m] = y_imag
q = np.convolve(h,q0)
```

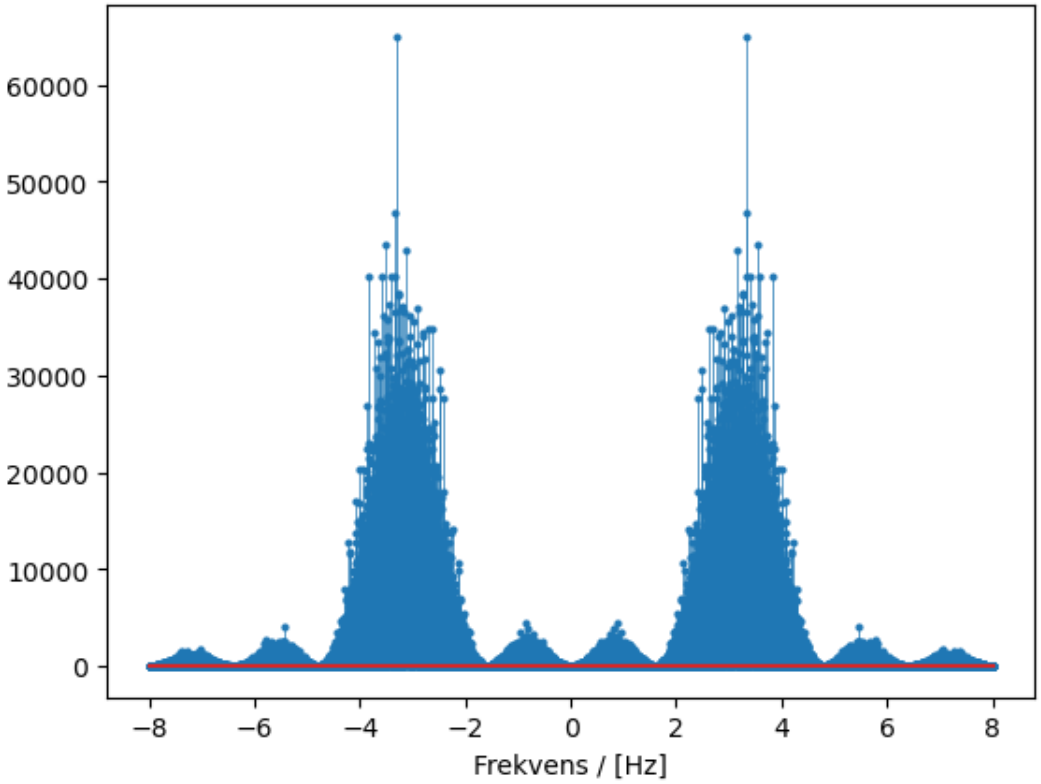
[92]:
```
fc = 32/T
wc = 2*np.pi*fc
N = q.size
t = np.linspace(0, 1000*T*L, num=N, endpoint=True)
print(min(t), max(t))
#t = np.arange(0, 1000*T*m, )

i_mod = i * np.sqrt(2)*np.cos(wc*t)
q_mod = q * np.sqrt(2)*np.sin(wc*t)
v = i_mod + q_mod
```

```
0.0 16000.0
```

[93]:
```
v_fft = np.fft.fft(v) * Ts
v_energy = np.abs(v_fft)**2
#f_axis = np.linspace(-1/(2*Ts), 1/(2*Ts), num=N)
f_axis = np.arange(-1/(2*Ts), 1/(2*Ts), 1/Ts/v.shape[0])
markers,stems,base = plt.stem(f_axis, np.fft.fftshift(v_energy))
plt.setp(stems, 'linewidth', 0.5)
plt.setp(markers, markersize=2)
plt.xlabel('Frekvens / [Hz]')
```

```
plt.show()
```

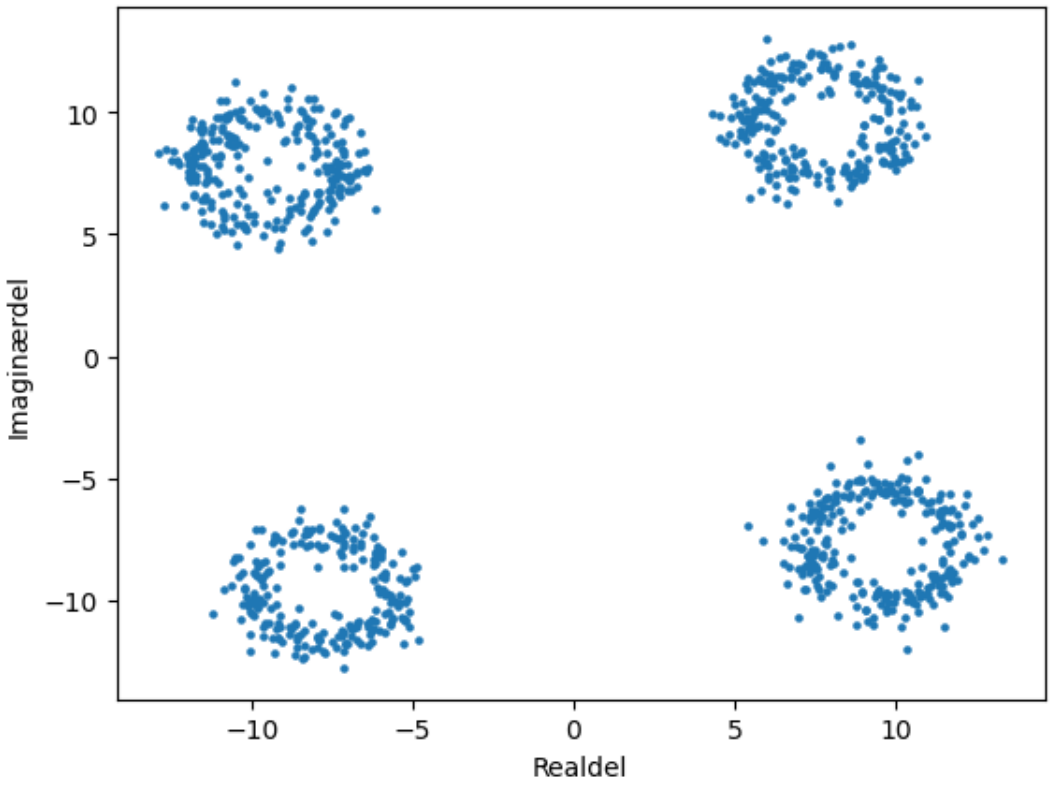


3 Spørgsmål 3

[95]:
```
v_cos = v * np.sqrt(2)*np.cos(wc*t)
v_sin = v * np.sqrt(2)*np.sin(wc*t)
v_cos_filter = np.convolve(h, v_cos)
v_sin_filter = np.convolve(h, v_sin)
received = v_cos_filter + 1j*v_sin_filter
y_k_sampled = received[m:-m:m].reshape(1000,16).T
```

[96]:
```
R = np.zeros((L,1000), dtype=np.cdouble)
for n in range(y_k_sampled.shape[1]):
    R_n = 1/np.sqrt(L)*np.fft.fft(y_k_sampled[:,n])
    R[:,n] = R_n
```

[103]:
```
plt.scatter(np.real(R[5]), np.imag(R[5]), s=5)
plt.xlabel('Realdel')
plt.ylabel('Imaginærdel')
plt.show()
```



[98]:
```
real_sign = np.sign(np.real(R))
imag_sign = np.sign(np.imag(R))
sum(sum(a_k != real_sign + 1j*imag_sign))
```

[98]: 0

4 Spørgsmål 4

[100]:
```
#noise
sigma = np.sqrt(2)
noise = np.random.normal(loc=0, scale=sigma, size=v.size)
v_noise = v + noise

#receiver magic
v_noise_cos = v_noise * np.sqrt(2)*np.cos(wc*t)
v_noise_sin = v_noise * np.sqrt(2)*np.sin(wc*t)
v_noise_cos_filter = np.convolve(h, v_noise_cos)
v_noise_sin_filter = np.convolve(h, v_noise_sin)
received_noise = v_noise_cos_filter + 1j*v_noise_sin_filter
y_k_noise_sampled = received_noise[m:-m:m].reshape(1000,16).T
```

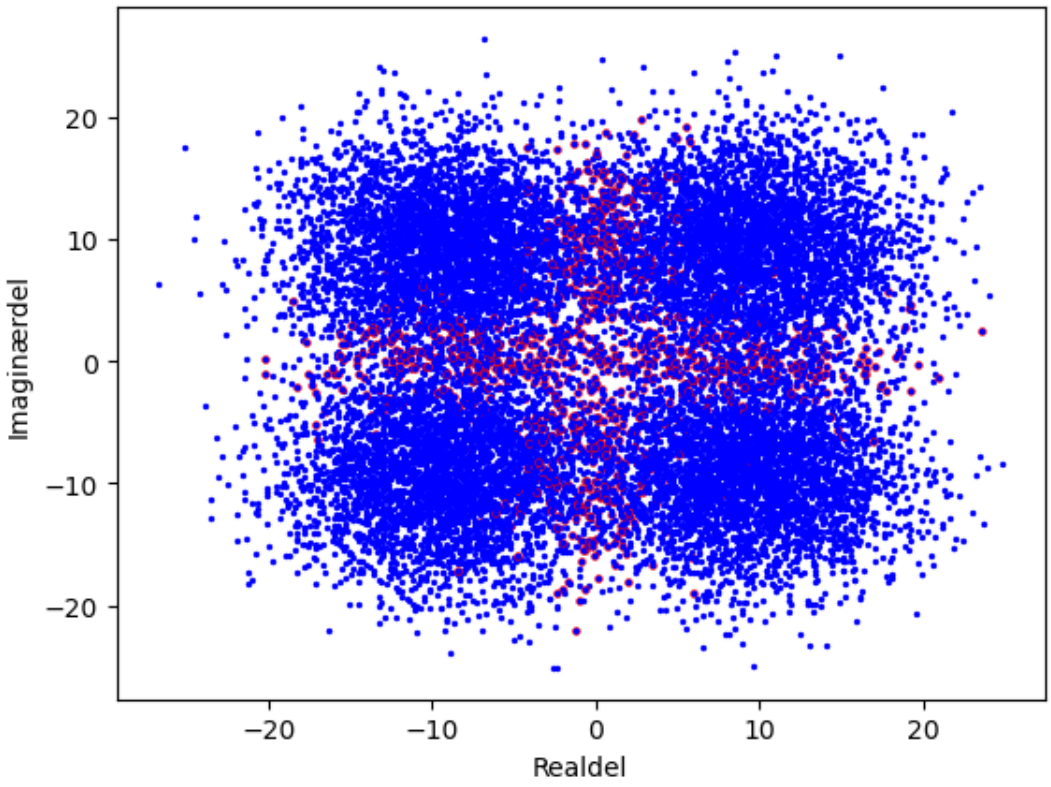
```
R_noise = np.zeros((L,1000), dtype=np.cdouble)
for n in range(y_k_noise_sampled.shape[1]):
    R_n_noise = 1/np.sqrt(L)*np.fft.fft(y_k_noise_sampled[:,n])
    R_noise[:,n] = R_n_noise

#errors
real_sign = np.sign(np.real(R_noise))
imag_sign = np.sign(np.imag(R_noise))
print('Errors:', sum(sum(a_k != real_sign + 1j*imag_sign)))

tot = 0
for channel in range(L):
    #wrong points
    coords = np.nonzero(a_k[channel] != (real_sign[channel] +␣
 ↪1j*imag_sign[channel]))
    plt.scatter(np.real(R_noise[channel][coords]), np.
 ↪imag(R_noise[channel][coords]), c='r', s=5)
    print(f'Errors in channel {channel}: {len(coords[0])}')
    tot += len(coords[0])

    #constellations
    plt.scatter(np.real(R_noise[channel]), np.imag(R_noise[channel]), c='b',␣
 ↪s=2)
print(tot)
plt.xlabel('Realdel')
plt.ylabel('Imaginærdel')
plt.show()
```

```
Errors: 1020
Errors in channel 0: 34
Errors in channel 1: 37
Errors in channel 2: 27
Errors in channel 3: 52
Errors in channel 4: 58
Errors in channel 5: 54
Errors in channel 6: 96
Errors in channel 7: 96
Errors in channel 8: 102
Errors in channel 9: 114
Errors in channel 10: 93
Errors in channel 11: 75
Errors in channel 12: 57
Errors in channel 13: 49
Errors in channel 14: 49
Errors in channel 15: 27
1020
```



[101]:
```
def qfunc(x):
    return 1-norm.cdf(x)
```

[102]:
```
1000*L*2*qfunc(np.sqrt(m/sigma**2))
```

[102]: 405.55709883949123

5 Spørgsmål 5

[113]:
```
symbol_set = [-1,1]
a_k = np.random.choice(symbol_set, size=(L,1000), replace=True)
a_k = a_k.astype(np.cdouble)
a_k.shape
```

[113]: (16, 1000)

[114]:
```
lower_rows = a_k[1:L//2,:]
```

[115]:
```
for l in range(1, lower_rows.shape[0]+1):
    imaginary_part = a_k[-l,:] * 1j
```

```
    lower_rows[l-1] += imaginary_part
```

[119]:
```
flipped = np.conj(np.flip(lower_rows, axis=0))
```

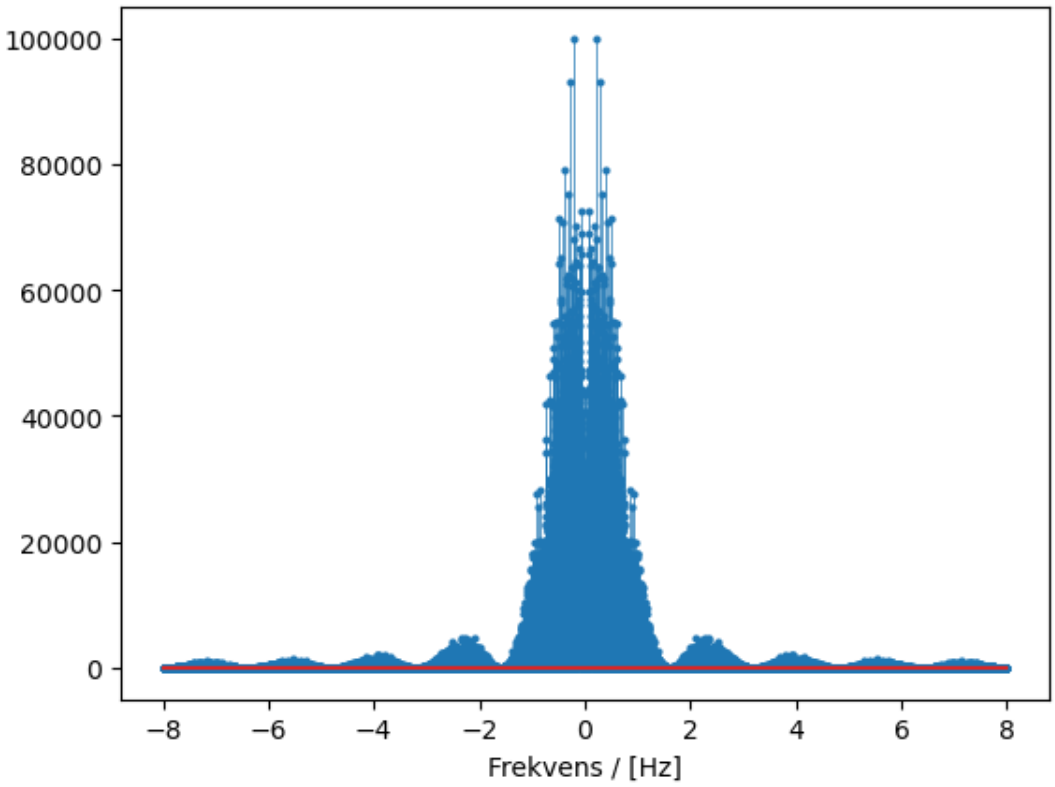
[121]:
```
a_k[1:L//2,:] = lower_rows
a_k[L//2+1:,:] = flipped
```

[126]:
```
y_k = np.zeros(L*1000, dtype=np.cdouble)
for k in range(a_k.shape[1]):
    y = np.sqrt(L)*np.fft.ifft(a_k[:,k].T)
    y_k[k*L:(k+1)*L] = y
```

[129]:
```
all(np.imag(y_k) == np.zeros(L*1000))
```

[129]: True

[131]:
```
h = np.ones(10)
y_k0 = np.zeros(m*y_k.size)
y_k0[::m] = np.real(y_k)
v = np.convolve(h, y_k0)
v_fft = np.fft.fft(v) * Ts
v_energy = np.abs(v_fft)**2
f_axis = np.arange(-1/(2*Ts), 1/(2*Ts), 1/Ts/v.shape[0])
markers,stems,base = plt.stem(f_axis, np.fft.fftshift(v_energy))
plt.setp(stems, 'linewidth', 0.5)
plt.setp(markers, markersize=2)
plt.xlabel('Frekvens / [Hz]')
plt.show()
```



[132]:
```
r_real = np.real(v)
r_imag = np.imag(v)
r_real_filter = np.convolve(h, r_real)
r_imag_filter = np.convolve(h, r_imag)
received = r_real_filter + 1j*r_imag_filter
y_k_sampled = received[m:-m:m].reshape(1000,16).T
```