

```
%%html
<style>
  .output_scroll {
    overflow: visible !important;
    height: auto !important;
  }
</style>
```

```
import numpy as np
import scipy.signal as signal
import librosa
import librosa.display
import matplotlib.pyplot as plt
import sympy as sp
import requests
from io import BytesIO
```

```
##### GitLab link til vores lydfile #####
lyd_url = "https://gitlab.au.dk/au280502/dsb/-/raw/main/DSB%20MPC/EKGlyd.mp3?inline=false"
```

```
session = requests.Session()
session.headers.update({'User-Agent': 'Mozilla/5.0'})
```

```
lydfil = "EKGlyd.mp3"
##### Indlæsning af lydfil #####
try:
```

```
    # Hent filen
    response = session.get(lyd_url)
    response.raise_for_status() # Tjek for fejl
```

```
    # Gem midlertidigt i hukommelsen
    audio_data = BytesIO(response.content)
```

```
    # Indlæs med librosa og print bekræftelse samt varigheden
    y, fs = librosa.load(audio_data, sr=1000)
    lydfil = {"signal": y, "samplerate": fs}
    print(f"✅ lydfil indlæst succesfuldt!")
    print(f"Varighed: {len(y)/fs:.2f} sekunder\n")
    print("samplerate:", fs)
```

```
except Exception as e:
    print(f"❌ Fejl ved indlæsning af: {str(e)}")
    print("Mulige årsager:")
    print("- Filen kræver login (AU GitLab godkendelse)")
    print("- Adgangsrestriktioner på filen")
    print("- Filen er korrupt\n")
```

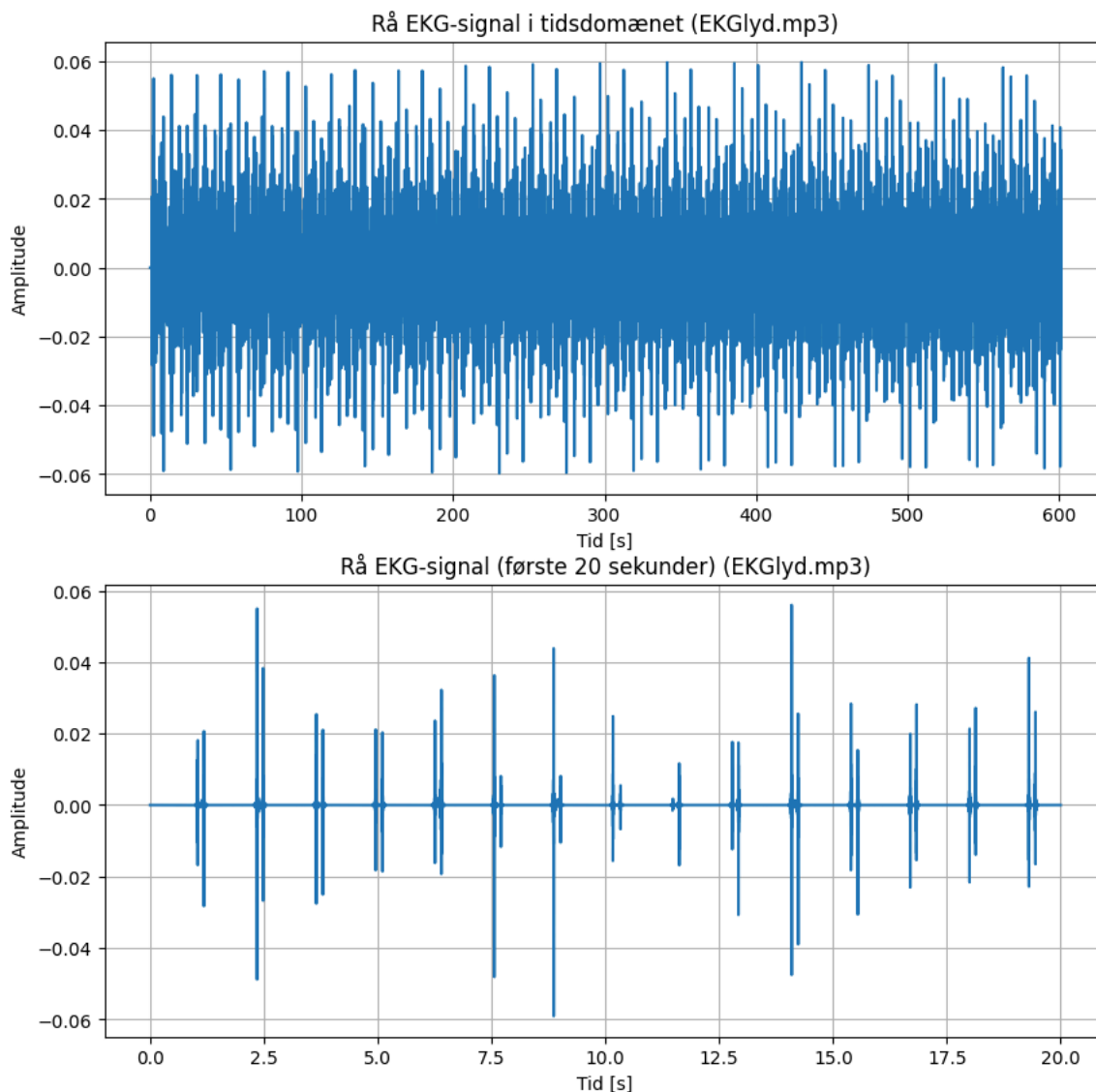
```
✅ lydfil indlæst succesfuldt!
Varighed: 601.05 sekunder
```

```
samplerate: 1000
```

```
##### Plotting af EKGlyd.mp3 #####
# Først plottes hele signalet
plt.figure(figsize=(10, 10))
plt.subplot(2, 1, 1)
plt.plot(np.linspace(0, len(y)/fs, len(y)), y)
plt.title("Rå EKG-signal i tidsdomænet (EKGlyd.mp3)")
plt.xlabel("Tid [s]")
plt.ylabel("Amplitude")
plt.grid()
print("Plottet af hele signalet for EKGlyd.mp3\n")
```

```
# Her tager vi de første 10 sec
plt.subplot(2, 1, 2)
plt.plot(np.linspace(0, len(y)/fs, len(y))[:fs*20], y[:fs*20])
plt.title("Rå EKG-signal (første 20 sekunder) (EKGlyd.mp3)")
plt.xlabel("Tid [s]")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
print("Plottet af første 20 sekunder for EKGlyd.mp3\n")
```

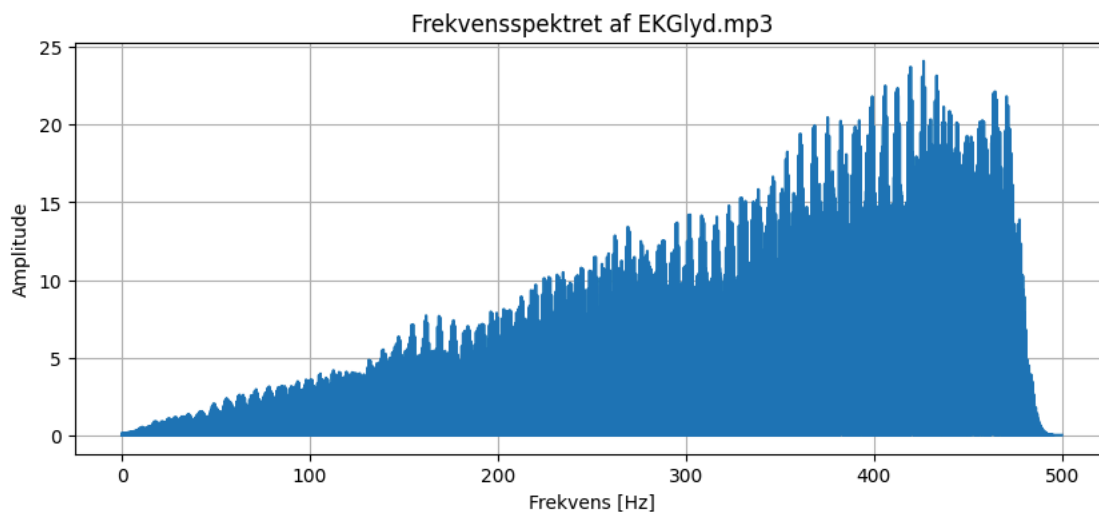
Plottet af hele signalet for EKGlyd.mp3



Plottet af første 20 sekunder for EKGlyd.mp3

```
#Signalet plottes i frekvensdomænet for at danne os et lille indledende
#overblik i, hvordan det ser ud.
freqs = np.fft.rfftfreq(len(y), 1/fs)
ekg_fft = np.abs(np.fft.rfft(y))

plt.figure(figsize=(10, 4))
plt.plot(freqs, ekg_fft)
plt.title("Frekvensspektret af EKGlyd.mp3")
plt.xlabel("Frekvens [Hz]")
plt.ylabel("Amplitude")
plt.grid()
plt.show()
```



##### Forstyrrelserne filtreres #####

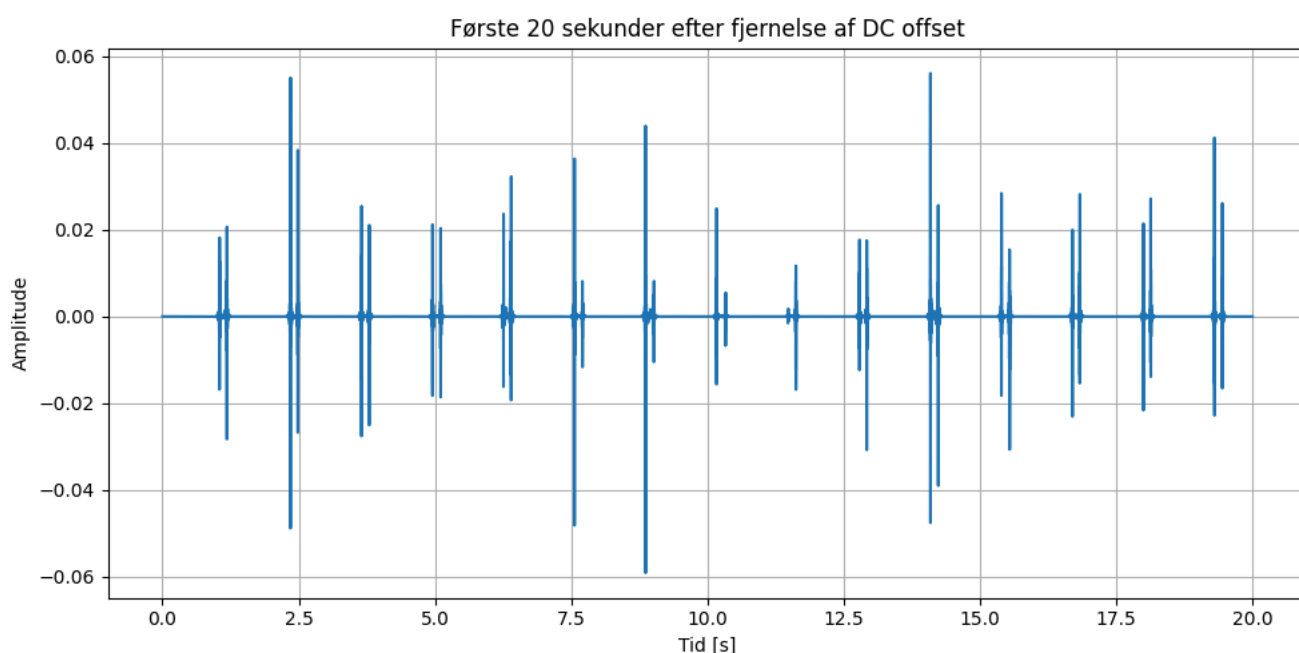
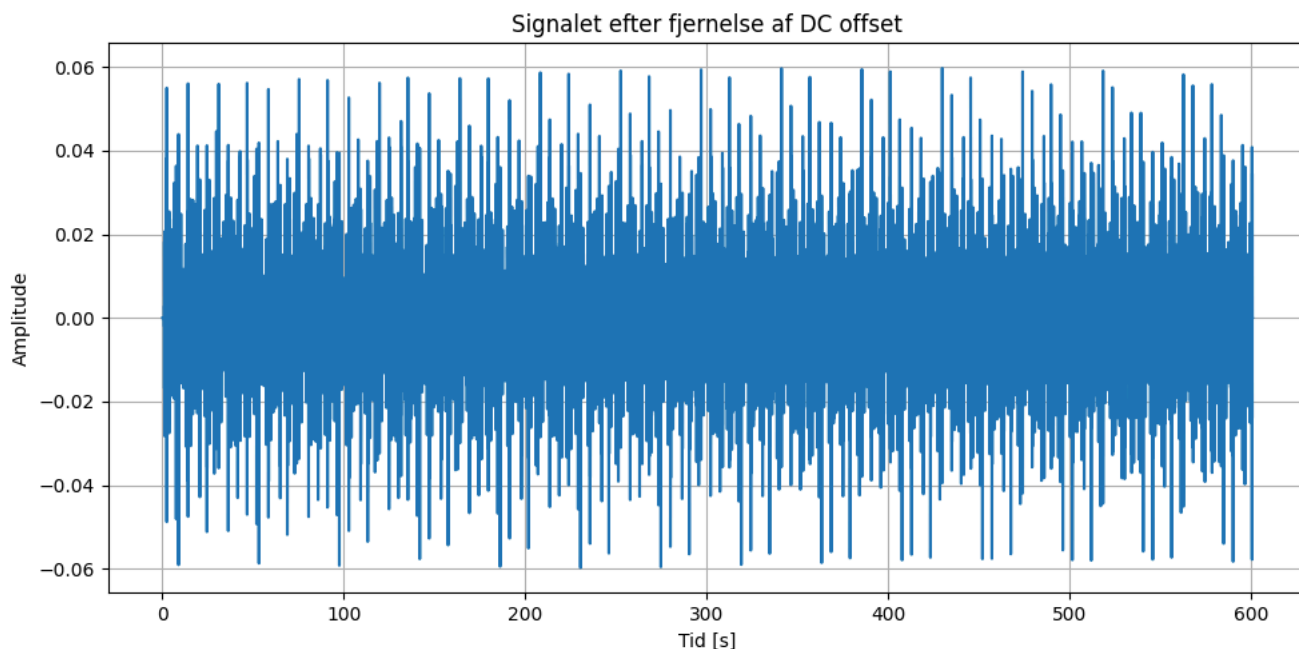
```
#DC offset
#fjerner mean værdien
y_filtered = y-np.mean(y)
```

```
#De nye graffer plottes:
plt.figure(figsize=(10, 10))
# Hele signalet
plt.subplot(2, 1, 1)
plt.plot(np.linspace(0, len(y_filtered)/fs, len(y_filtered)), y_filtered)
plt.title("Signalet efter fjernelse af DC offset")
plt.xlabel("Tid [s]")
plt.ylabel("Amplitude")
plt.grid()
```

```
#Første 20 sekunder
plt.subplot(2, 1, 2)
plt.plot(np.linspace(0, len(y_filtered)/fs, len(y_filtered))[ :fs*20], y_filtered[:fs*20])
plt.title("Første 20 sekunder efter fjernelse af DC offset")
plt.xlabel("Tid [s]")
plt.ylabel("Amplitude")
plt.grid()
```

```
plt.tight_layout()
plt.show()
```

```
print("Plottet af både hele signalet og første 20 sekunder efter fjernelse af DC offset\n")
```



Plottet af både hele signalet og første 20 sekunder efter fjernelse af DC offset

```
#Filtrering af baseline drift
#i design af vores FIR highpass filter vil vi lave en cutoff på 0.25Hz
N = 10001 # Antal taps
cutoff = 0.25 # Hz
f_pass = cutoff+0.5 # Nedre grænse for passbåndet (Hz)

# Opløsningen er fs / N, frekvensakse: intervallet 0 til 1000 Hz
f = np.linspace(0, fs, N, endpoint=False)

# H_desired er:
# 0 for frekvenser under cutoff
# 1 for frekvenser over 0.75 Hz (passbånd)
# i området mellem cutoff og passbånd lineær overgang fra 0 til 1.
H_desired = np.zeros(N, dtype=float)

# Da et FIR-filter med reelt impulsvar kræver Hermitisk symmetri, definerer vi først værdierne
# for frekvenser fra 0 til Nyquist (fs/2) og spejler dem så.
half_N = N // 2 + 1 # frekvenserne fra 0 op til og med Nyquist

for k in range(half_N):
    # Stopbånd: Under 0.25 Hz
    if f[k] < cutoff:
        H_desired[k] = 0.0
    # Passbånd: Over 0.75 Hz
    elif f[k] > f_pass:
        H_desired[k] = 1.0
```

```
else:
    # transitionområdet
    H_desired[k] = (f[k] - cutoff) / (f_pass - cutoff)

# Spejling for frekvenserne fra Nyquist til fs:
for k in range(half_N, N):
    H_desired[k] = H_desired[N - k]

# Den inverse FFT (np.fft.iff) konverterer H_desired til tidsdomænet, dvs. vi får
# filterets impulssvar h[n]. Vi tager den reelle del for at fjerne eventuelle små imaginære værdier.
h = np.fft.iff(H_desired)
h = np.real(h)
# fftshift centrerer impulssvaret.
h = np.fft.fftshift(h)

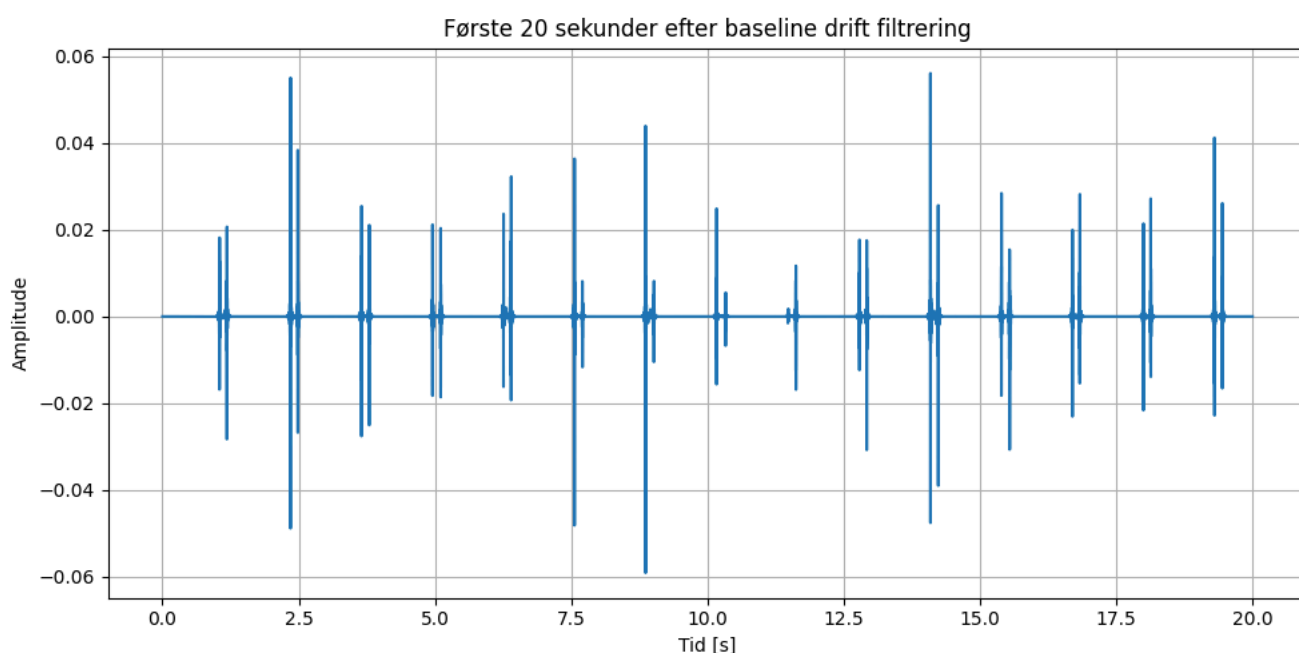
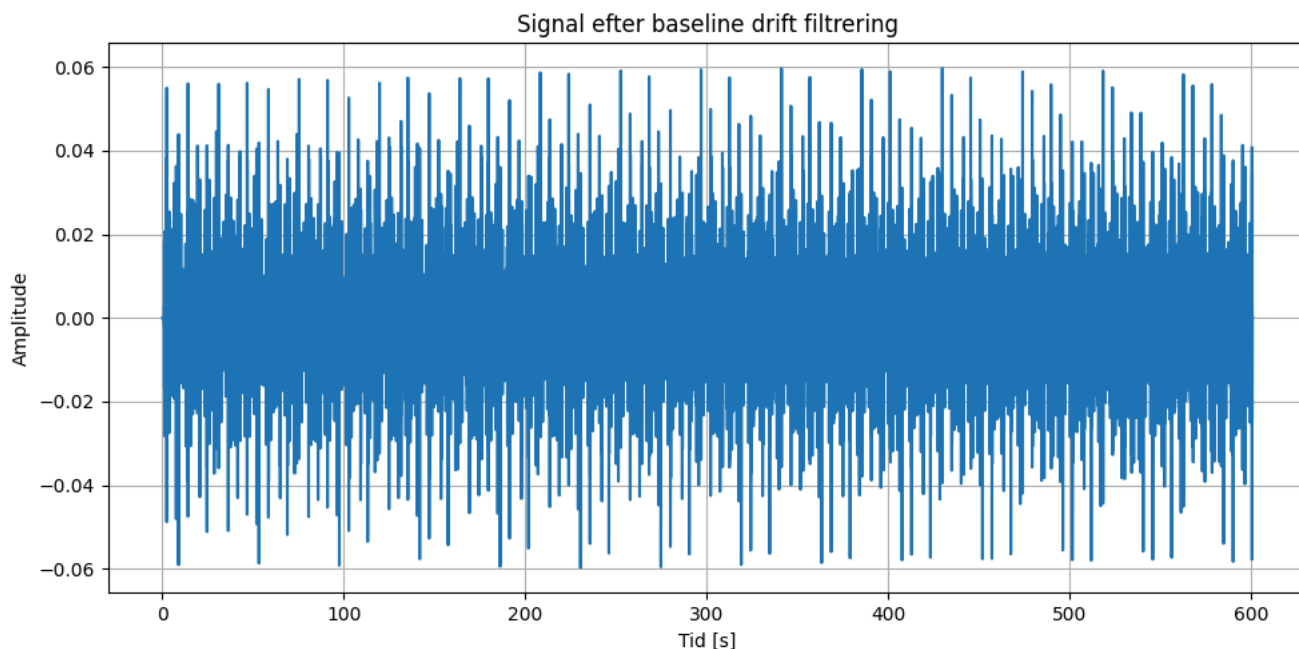
# Filteret anvendes
y_filteredb = signal.filtfilt(h, [1.0], y_filtered)

#De nye graffer plottes:
plt.figure(figsize=(10, 10))
# Hele signalet
plt.subplot(2, 1, 1)
plt.plot(np.linspace(0, len(y_filteredb)/fs, len(y_filteredb)), y_filteredb)
plt.title("Signal efter baseline drift filtrering")
plt.xlabel("Tid [s]")
plt.ylabel("Amplitude")
plt.grid()

#Første 20 sekunder
plt.subplot(2, 1, 2)
plt.plot(np.linspace(0, len(y_filteredb)/fs, len(y_filteredb))[:fs*20], y_filteredb[:fs*20])
plt.title("Første 20 sekunder efter baseline drift filtrering")
plt.xlabel("Tid [s]")
plt.ylabel("Amplitude")
plt.grid()

plt.tight_layout()
plt.show()

print("Plottet af både hele signalet og første 20 sekunder efter fjernelse af baseline drift\n")
```



Plottet af både hele signalet og første 20 sekunder efter fjernelse af baseline drift

```
#Notch filter til fjernelse af netværkstøj 50Hz og 100Hz
f0_50 = 50.0 # centrering omkring 50Hz
f0_100 = 100.0 # centrering omkring 100Hz
Q = 30 #Q faktoren er sat til 30, hvilket eliminerer signalet i
#en meget snæver område omkring de uønskede frekvenser

#iirnotch funktionen til isolation af netstøj:
b50, a50 = signal.iirnotch(f0_50, Q, fs)
b100, a100 = signal.iirnotch(f0_100, Q, fs)

#vi fjerner 1 komponent ad gangen
y_filteredn = signal.filtfilt(b50, a50, y_filteredb)
y_filtereds = signal.filtfilt(b100, a100, y_filteredn)

#De nye graffer plottes:
plt.figure(figsize=(10, 10))
# Hele signalet
plt.subplot(2, 1, 1)
plt.plot(np.linspace(0, len(y_filtereds)/fs, len(y_filtereds)), y_filtereds)
plt.title("Signal efter netstøj filtrering")
plt.xlabel("Tid [s]")
plt.ylabel("Amplitude")
plt.grid()

#Første 20 sekunder
```

```

plt.subplot(2, 1, 2)
plt.plot(np.linspace(0, len(y_filtereds)/fs, len(y_filtereds))[:fs*20], y_filtereds[:fs*20])
plt.title("Første 20 sekunder efter netstøj filtrering")
plt.xlabel("Tid [s]")
plt.ylabel("Amplitude")
plt.grid()

plt.tight_layout()
plt.show()

print("Plottet af både hele signalet og første 20 sekunder efter fjernelse af 50Hz og 100Hz netstøj\n")

#opskrivning af differensligningerne med korrekte koefficienter:
#først definerer vi de symbolske variabler vha sympy
n = sp.symbols('n', integer=True)
x = sp.Function('x')
y = sp.Function('y')

#vi sætter sympy til at udskrive med 2 decimaler
sp.init_printing(precision=2)

def diff_eq_sympy(b, a):

    b_round = [sp.Float(round(coef, 2)) for coef in b]
    a_round = [sp.Float(round(coef, 2)) for coef in a]

    # Bygger summen for inputs
    feedforward = sum(b_round[i] * x(n - i) for i in range(len(b_round)))

    # Bygger summen for outputs med 2 decimaler
    feedback = sum(a_round[j] * y(n - j) for j in range(1, len(a_round)))

    # Den symboliske ligning:  $y(n) = \text{feedforward} - \text{feedback}$ 
    eq = sp.Eq(y(n), feedforward - feedback)
    return eq

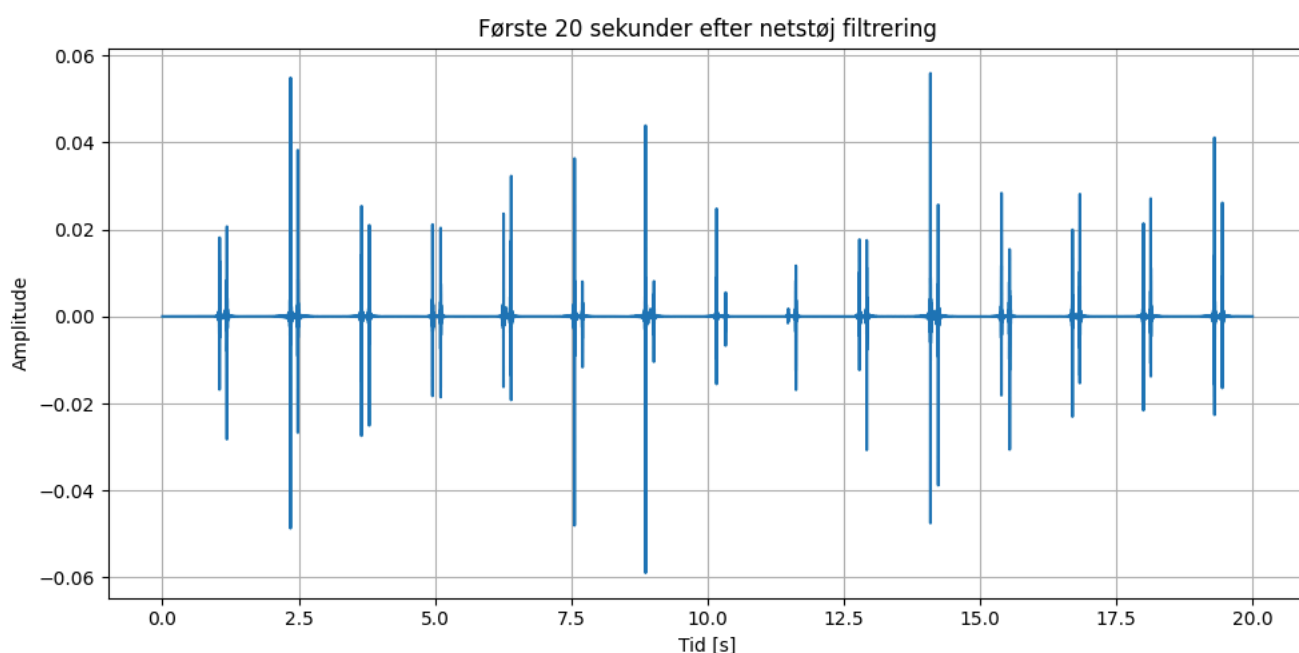
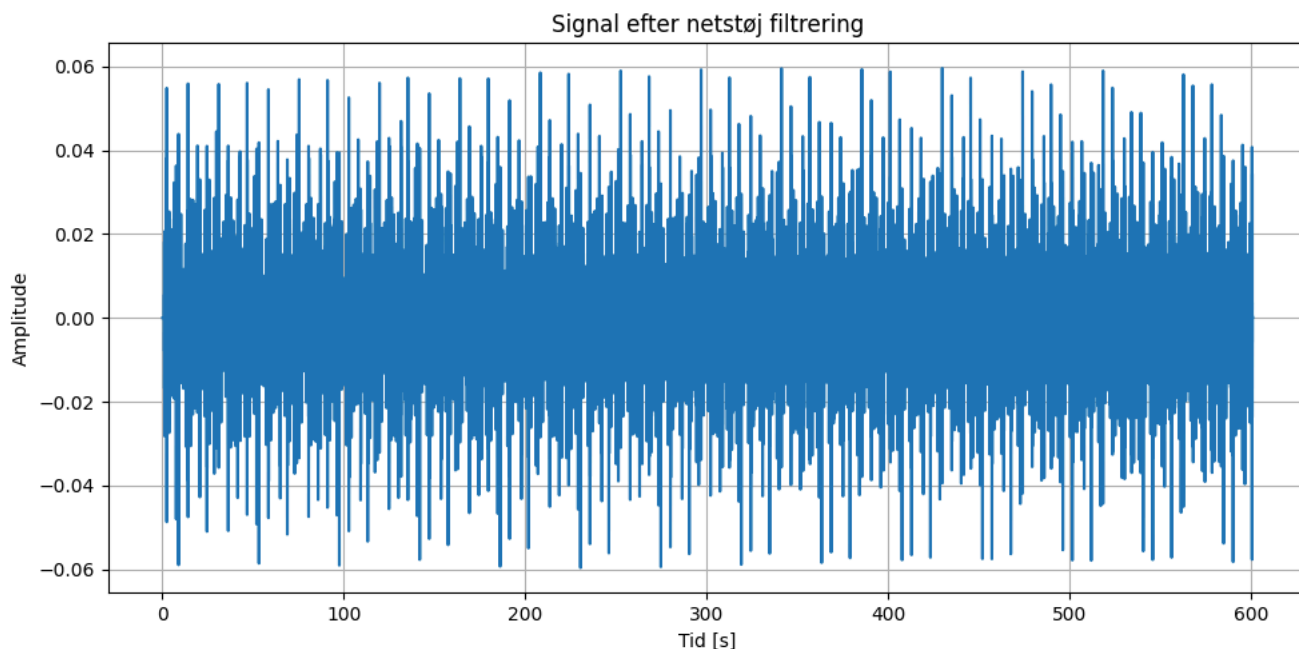
eq_50 = diff_eq_sympy(b50, a50)
eq_100 = diff_eq_sympy(b100, a100)

# Pprint ligningerne
print("50 Hz filter differensligning:")
sp.pprint(eq_50)
print("\n100 Hz filter differensligning:")
sp.pprint(eq_100)

#vi anvender group_delay funktionen til at udlede et array
#med frekvenspunkter w og group delay gd
w, gd = signal.group_delay((b50, a50), fs=fs)

# Plot group delay
plt.figure(figsize=(10, 4))
plt.plot(w, gd, 'b-', linewidth=2)
plt.xlabel('Frekvens [Hz]')
plt.ylabel('Gruppelay [samples]')
plt.title('Group Delay for Notch-filter ved 50 Hz')
plt.grid()
plt.xlim(0, 100)
plt.show()

```



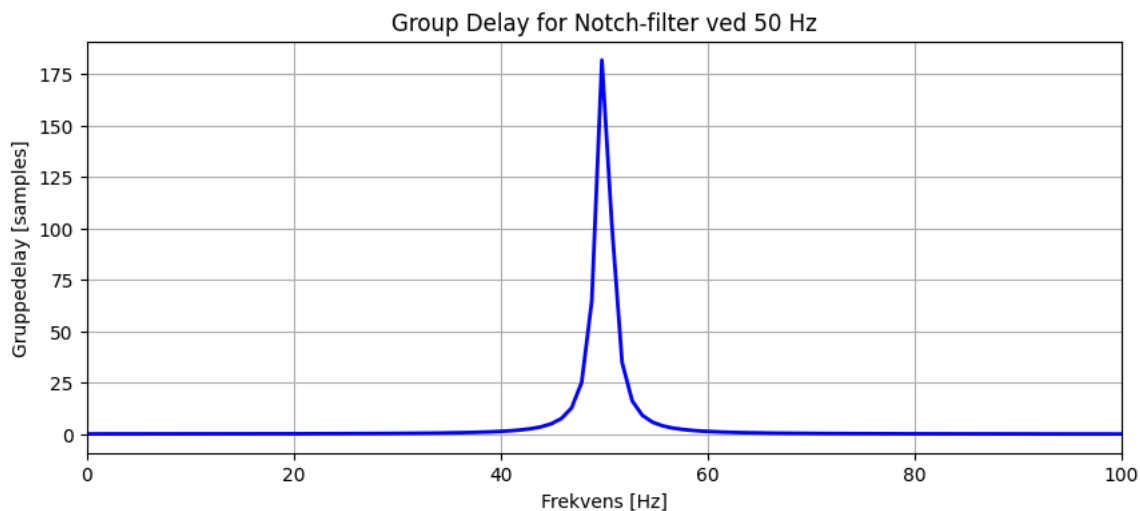
Plottet af både hele signalet og første 20 sekunder efter fjernelse af 50Hz og 100Hz netstøj

50 Hz filter differensligning:

$$y(n) = 0.99 \cdot x(n) + 0.99 \cdot x(n-2) - 1.89 \cdot x(n-1) - 0.99 \cdot y(n-2) + 1.89 \cdot y(n-1)$$

100 Hz filter differensligning:

$$y(n) = 0.99 \cdot x(n) + 0.99 \cdot x(n-2) - 1.6 \cdot x(n-1) - 0.98 \cdot y(n-2) + 1.6 \cdot y(n-1)$$





```

#HF filtrering
cutoff = 40                # Øvre grænse for relevante frekvenser
f_stop = 45                # Nedre grænse for stopbåndet
orders = [201, 401, 601, 801] # eksperimentert med forskellige filterordener
passbandRipple = 0.5 #dB
stopbandAttenuation = 60 #dB

#beregning af ds og dp
dp = 10**((passbandRipple/20)-1)    #formel 7.51 i lærerbogen
ds = 10**(-(stopbandAttenuation/20)) #formel 7.52 i lærerbogen
print ("DP ",dp," og DS ", ds)

#beregning af fraction form formel 7.53
frac = dp/ds

#Vægtstop for stopbåndet, og enhedsjustering i passbåndet
weights = [frac, 1]

# ønsket signalvægtning (0 i stopbåndet, 1 i passbåndet)
desired = [1,0]

#båndene deles op:
#0-cutoff, cutoff-passbånd, passbånd til nyquist (fs/2)
bands = [0, cutoff, f_stop, fs/2]

#vi vil samle vores signaler og ordner
filtered_signals_orders = {}
filters_orders = {}

#De nye graffer plottes:
filtered_signals_orders = {}
filters_orders = {}

for order in orders:
    # filter vha. ParksMcClellan (og Remez) med filterlængde = order
    b_order = signal.remez(order, bands, desired, weight=weights, fs=fs)
    filters_orders[order] = b_order

    # signalet filtreres med filtfilt
    y_order = signal.filtfilt(b_order, [1.0], y_filtereds)
    filtered_signals_orders[order] = y_order

    #vi udskriver en bekræftelse
    print(f"Filter med orden {order} designet og anvendt.")

# Hele signalet
t_full = np.linspace(0, len(y_filtereds)/fs, len(y_filtereds))
plt.figure(figsize=(12, 10))
for order in orders:
    plt.plot(t_full, filtered_signals_orders[order], label=f"Order {order}")
plt.title("Signal efter HF filtrering med forskellige filterordener")
plt.xlabel("Tid [s]")
plt.ylabel("Amplitude")
plt.grid(True)
plt.legend()
plt.show()

#vi vælger at gå videre med denne for mere præcision
y_filteredh = filtered_signals_orders[801]

#Første 20 sekunder
plt.figure(figsize=(10,4))
plt.plot(np.linspace(0, len(y_filteredh)/fs, len(y_filteredh))[:fs*20], y_filteredh[:fs*20])
plt.title("Første 20 sekunder efter HF filtrering")
plt.xlabel("Tid [s]")
plt.ylabel("Amplitude")
plt.grid()
plt.show()

# Visualiser impulsresponsen (h(n))
# Da vi her anvender Parks McClellan metode til at designe filtret, er
# vores koefficienter impulsresponsen.
plt.figure(figsize=(10, 4))
plt.plot(b)
plt.title("Impulsrespons for HF FIR-filtret")
plt.xlabel("n (samples)")
plt.ylabel("Amplitude")
plt.grid()
plt.show()

# Beregn og plot overføringskarakteristik vha FFT
# impulsresponsen (b) med zero-padding for bedre opløsning

```

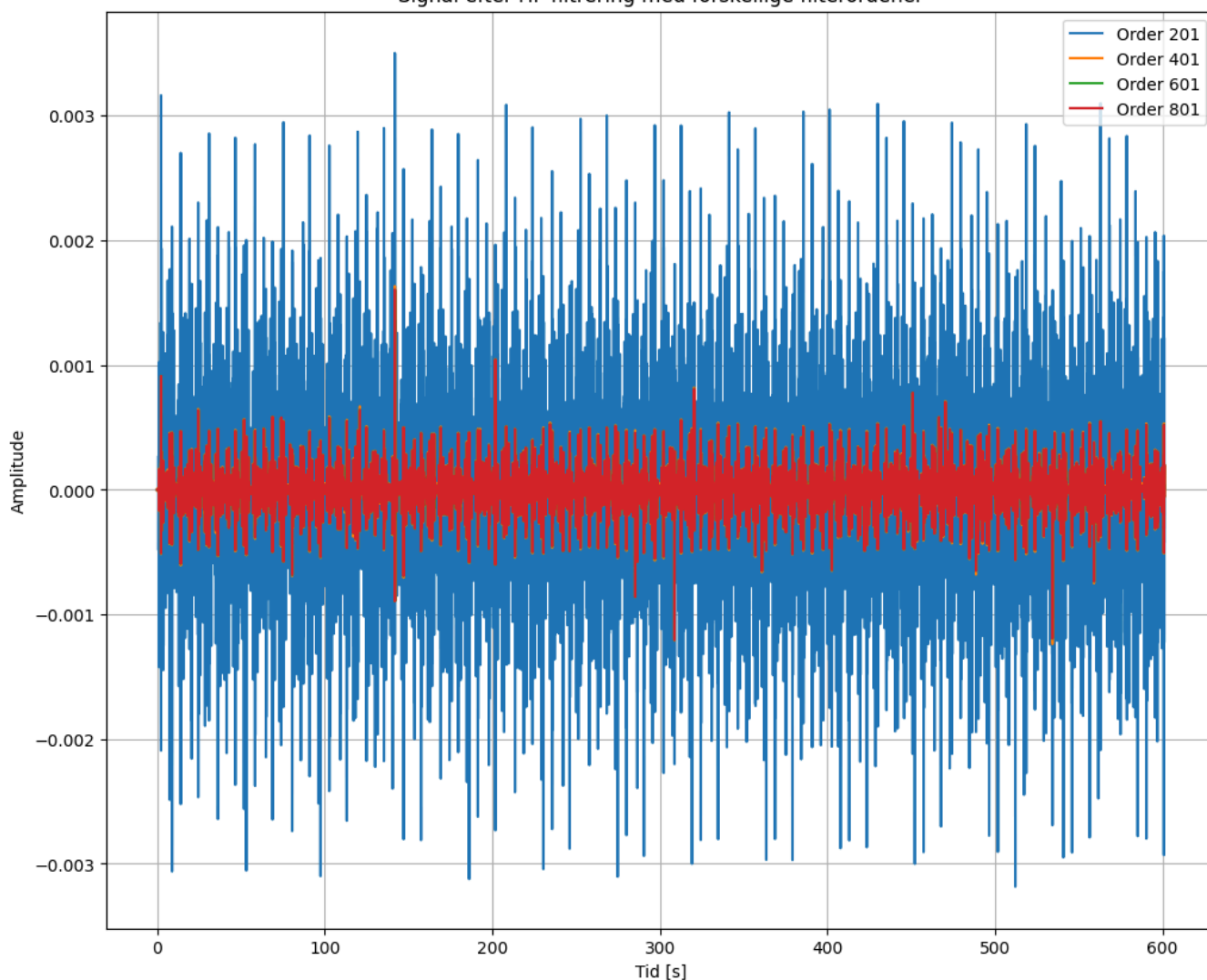
```
H = np.fft.fft(b, 4096)

# Frekvensakse (0 til fs/2)
freq = np.fft.fftfreq(4096, 1/fs)[:4096//2]
H_magnitude = np.abs(H[:4096//2])
H_db = 20 * np.log10(H_magnitude + 1e-10) # Konverter til dB

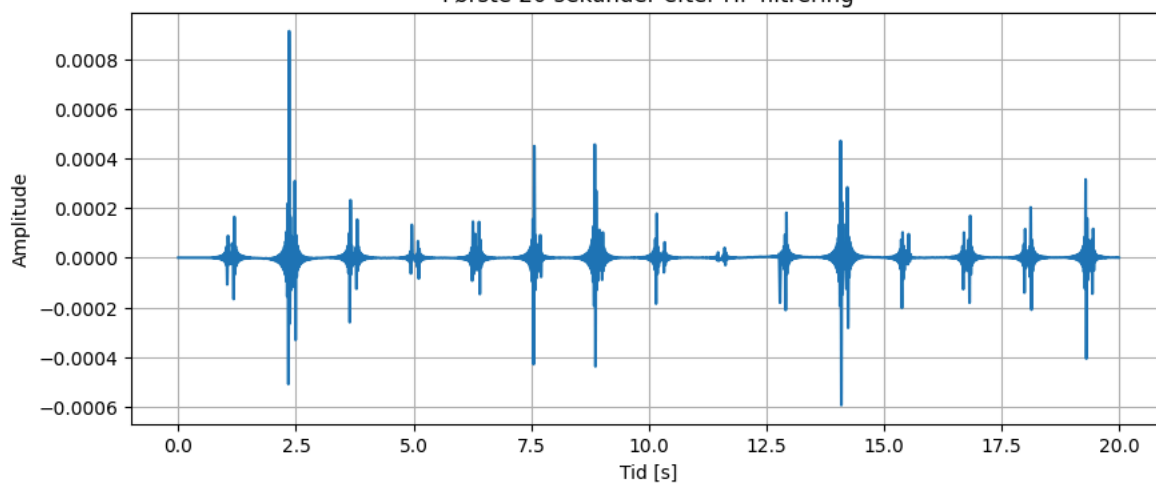
#dette plottes
plt.figure(figsize=(10, 4))
plt.plot(freq, H_db)
plt.title("Filterets overføringskarakteristik i dB")
plt.xlabel("Frekvens [Hz]")
plt.ylabel("Amplitude [dB]")
plt.xlim(0, 100)
plt.grid()
plt.show()
```

DP 0.05925372517728885 og DS 0.001  
Filter med orden 201 designet og anvendt.  
Filter med orden 401 designet og anvendt.  
Filter med orden 601 designet og anvendt.  
Filter med orden 801 designet og anvendt.

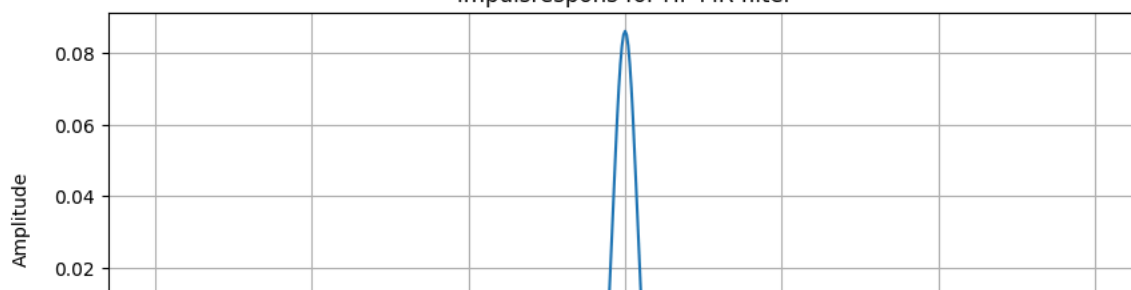
Signal efter HF filtrering med forskellige filterordener

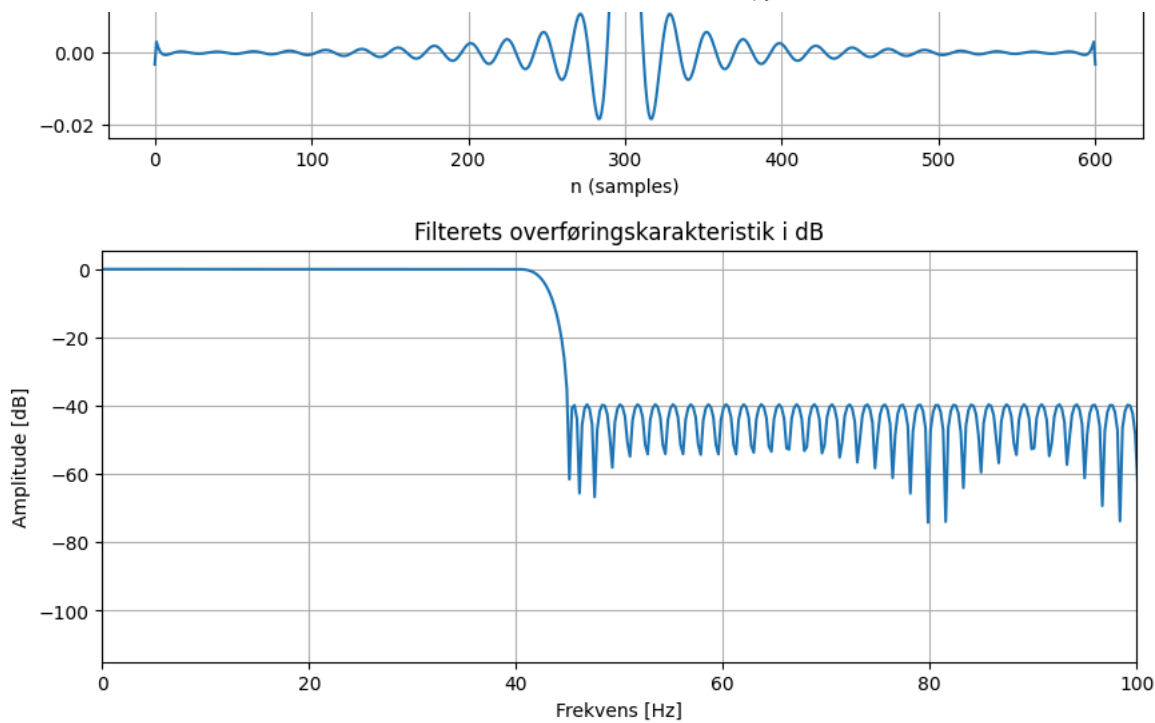


Første 20 sekunder efter HF filtrering



Impulsrespons for HF FIR-filter





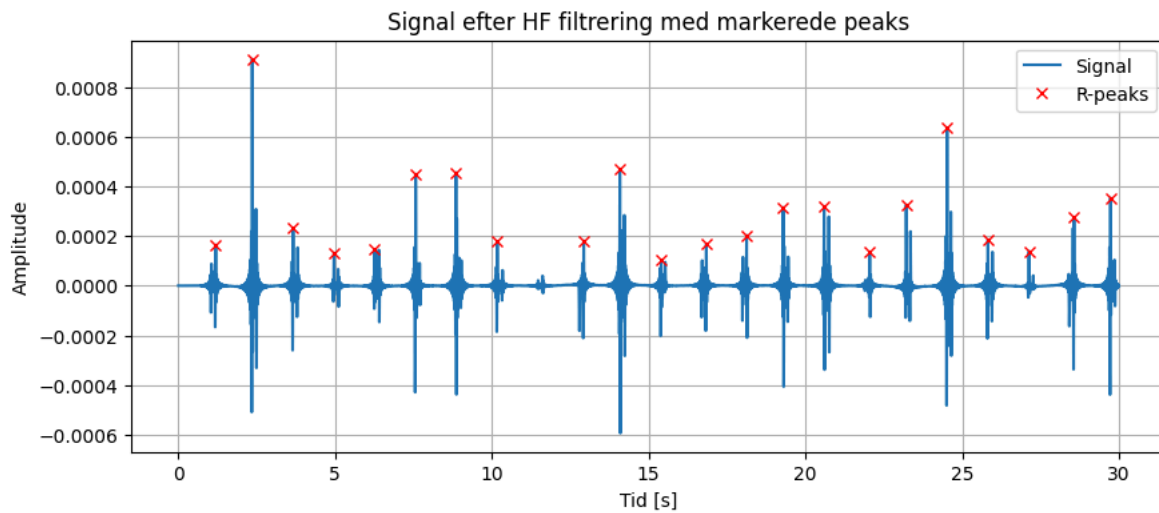
```
#For at undersøge hjerterytmen og eventuelt variation vil vi prøve at zoome ud
#og markere peaks med scipy find peaks
segment = y_filteredh[:fs*30]
t_segment = np.linspace(0, 30, len(segment))

# Find peaks med:
# minimum prominence på 0.00007(ca aflæst på grafen)
# distance på ca fs *0.5 = 500ms, så vi undgå for hyppige peakmarkeringer.
# denne distance er også aflæst på grafen
peaks, properties = signal.find_peaks(segment, prominence=0.00007, distance=fs*0.5)

plt.figure(figsize=(10, 4))
plt.plot(t_segment, segment, label="Signal")
plt.plot(t_segment[peaks], segment[peaks], "rx", label="R-peaks")
plt.title("Signal efter HF filtrering med markerede peaks")
plt.xlabel("Tid [s]")
plt.ylabel("Amplitude")
plt.grid()
plt.legend()
plt.show()

#gennemsnitlig puls BPM
peak_times = peaks / fs
intervals = np.diff(peak_times)
puls = 60 / np.mean(intervals)
print("Gennemsnitlig puls:", puls, "BPM")

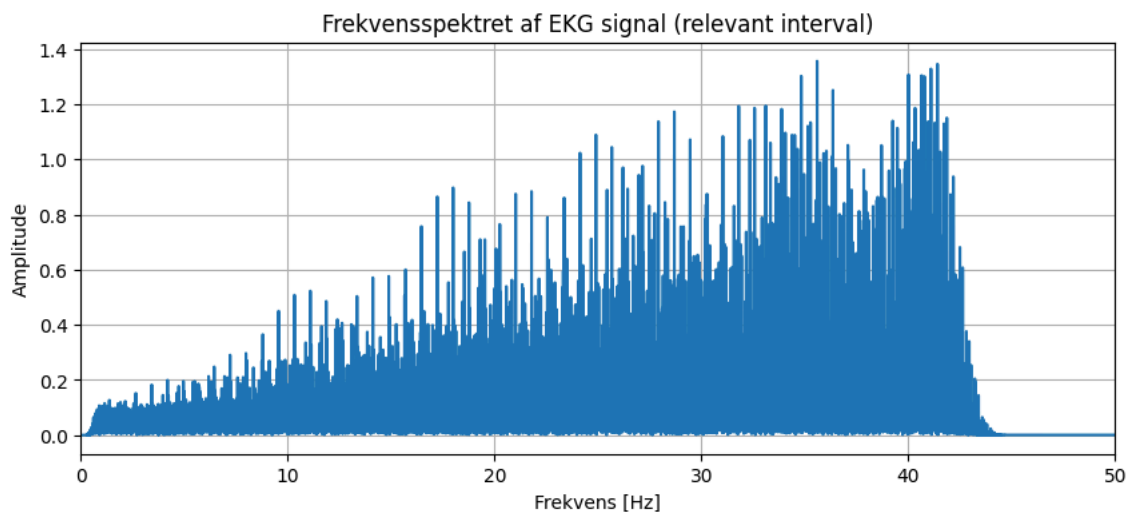
intervals_ms = intervals * 1000 # vi omregner først intervallerne til ms
diffs = np.diff(intervals_ms) # Forskelle mellem på hinanden følgende RR-intervaller
HRV = np.sqrt(np.mean(diffs**2))
print(f"HRV: {HRV:.2f} ms")
```



Gennemsnitlig puls: 44.14392320358757 BPM  
HRV: 499.13 ms

```
#plotning af filtreret kode i frekvensdomænet
freqs = np.fft.rfftfreq(len(y_filteredh), 1/fs)
ekg_fft = np.abs(np.fft.rfft(y_filteredh))

plt.figure(figsize=(10, 4))
plt.plot(freqs, ekg_fft)
plt.title("Frekvensspektret af EKG signal (relevant interval)")
plt.xlabel("Frekvens [Hz]")
plt.ylabel("Amplitude")
plt.xlim(0, 50) #fokuser på frekvenser fra 0 til 50 Hz
plt.grid()
plt.show()
```



```
#DFT af signalet for at analysere dominerende freks (spektral multiplication)
Ns = len(segment) # antal samples i segmentet
```

```
# Udfør FFT (DFT beregnes vha. den hurtige FFT-algoritme)
freqs = np.fft.rfftfreq(Ns, d=1/fs)
spectrum = np.fft.rfft(segment)
spectrum_magnitude = np.abs(spectrum)
```

```
# Her eksperimenterer vi med 6 forskellige passband cutoffs udvalgt
```

```
# for at se, om vi kan få et tydeligere pulsmåling frem
```

```
passbands = [
    (0.05, 5),
    (5, 15),
    (15, 25),
    (25, 35),
    (35, 45),
    (45, 50)
```

```
]
```

```
#arrays til at gemme signalerne og spektre
```

```

filtered_time_signals = []
filtered_spektre = []

# loop over de forskellige passbands og udfør spektral multiplikation
for f_low, f_high in passbands:
    # Konstruér et bandpassfilter: 1 inden for intervallet, 0 ellers
    band_mask = (freqs >= f_low) & (freqs <= f_high)
    spectral_filter = band_mask.astype(float)

    # Udfør spektral multiplikation (elementvis multiplikation i frekvensdomænet)
    filtered_spectrum = spectrum * spectral_filter
    filtered_spektre.append(filtered_spectrum)

    # Transformér det filtrerede spektrum tilbage til tidsdomænet
    filtered_time_signal = np.fft.irfft(filtered_spectrum, n=Ns)
    filtered_time_signals.append(filtered_time_signal)

# Plot det filtrerede spektrum for det aktuelle passband
plt.figure(figsize=(6, 2))
plt.plot(freqs, np.abs(filtered_spectrum), label=f"Filtered Spectrum ({f_low}-{f_high} Hz)")

plt.plot(freqs, spectral_filter * np.max(spectrum_magnitude), '--', label="Spectral Filter")
plt.title(f"Spektral Multiplikation: {f_low} - {f_high} Hz")
plt.xlabel("Frekvens [Hz]")
plt.ylabel("Amplitude")
plt.xlim(0, 120)
plt.grid(True)
plt.legend()
plt.show()

# Plot de filtrerede tidsdomænesignaler for alle passbands i subplots
t = np.linspace(0, 30, Ns) # Tidsvektor for et 30 sekunders segment
fig, axs = plt.subplots(3, 2, figsize=(14, 10))
axs = axs.flatten()

for idx, (f_low, f_high) in enumerate(passbands):
    axs[idx].plot(t, filtered_time_signals[idx])
    axs[idx].set_title(f"Filtered Signal ({f_low}-{f_high} Hz)")
    axs[idx].set_xlabel("Tid [s]")
    axs[idx].set_ylabel("Amplitude")
    axs[idx].grid(True)

plt.tight_layout()
plt.show()

```

