ISS Projekt 2023/24

Honza Pavlus, Honza Brukner a Honza Černocký, ÚPGM FIT VUT

6.11. 2023

1. Úvod

V projektu budete pracovat se biomedicínskými signály a to konkrétně se signálem elektrokardiogramu EKG. Vyzkoušíte si filtraci tohoto druhu signálu, abyste dostali krásné EKG křivky, které můžete vidět ve filmech. Dále si zkusíte vybudovat jednoduchý, ale účinný detektor QRS a ti, kteří se vrhnou i na bonusový úkol, si zkusí odhalit srdeční patologii. K dispozici dostanete každý 3 nahrávky jednokanálového EKG signálu, jeden zdravý a dva s různými patologiemi.

Projekt je nejlépe možno řešit v Python-u a to přímo v dodaném Python notebooku, který si můžete zkopírovat do vlastního Google Colabu. Projekt je také možno řešit v Matlab-u, Octave, Julii, jazyce C, Java nebo v libovolném jiném programovacím či skriptovacím jazyce. Je možné použít libovolné knihovny. Projekt se nezaměřuje na "krásu programování", není tedy nutné mít vše úhledně zabalené do okomentovaných funkcí (samozřejmě se ale okomentovaný kód lépe opravuje a to hlavně v případě podivných výsledků), ošetřené všechny chybové stavy, atd. Důležitý je výsledek.

Vaši práci odevzdáváte vyexportovanou do dvou souborů: (1) do PDF souboru login.pdf, (2) do Python notebooku login.ipynb. PDF musí obsahovat výsledky prokazatelně vytvořené Vaším kódem. V případě řešení projektu v jiném jazyce nebo prostředí než v dodaném Python notebooku, je prvním souborem protokol v PDF, druhý soubor je archiv s Vaším kódem. Ten musí být spustitelný na standardní fakultní distribuci Windows nebo Linuxu.

3. Vstup

Pro řešení projektu má každý student/ka k disposici osobní soubor se zdravým signálem (sinusovým rytmem): ***login.wav***, kde login je váš xlogin popřípadě VUT číslo (pro studenty FSI). Dále jsou k disposici ještě další dva signály: ***FIS.wav*** a ***KES.wav***. První signál obsahuje fibrilaci a druhý komorovou extrasystolu. Tyhle dva soubory jsou pro všechny společné a využijete je při řešení bonusového úkolu.

In [ ]:
```
#Načtení Vašeho signálu - xlogin99 nahraďte Vaším loginem
import soundfile as sf

!wget https://www.fit.vutbr.cz/study/courses/ISS/public/proj2023-24/xsalta01.wav
!wget https://www.fit.vutbr.cz/study/courses/ISS/public/proj2023-24/FIB.wav
!wget https://www.fit.vutbr.cz/study/courses/ISS/public/proj2023-24/KES.wav

x, fs = sf.read("xsalta01.wav")
```

```
--2023-12-17 11:31:45--  https://www.fit.vutbr.cz/study/courses/ISS/public/proj202
3-24/xsalta01.wav
Resolving www.fit.vutbr.cz (www.fit.vutbr.cz)... 147.229.9.23, 2001:67c:1220:809::
93e5:917
Connecting to www.fit.vutbr.cz (www.fit.vutbr.cz)|147.229.9.23|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10044 (9.8K) [audio/x-wav]
Saving to: ‘xsalta01.wav’

xsalta01.wav        100%[===================>]   9.81K  --.-KB/s    in 0s

2023-12-17 11:31:47 (200 MB/s) - ‘xsalta01.wav’ saved [10044/10044]

--2023-12-17 11:31:47--  https://www.fit.vutbr.cz/study/courses/ISS/public/proj202
3-24/FIB.wav
Resolving www.fit.vutbr.cz (www.fit.vutbr.cz)... 147.229.9.23, 2001:67c:1220:809::
93e5:917
Connecting to www.fit.vutbr.cz (www.fit.vutbr.cz)|147.229.9.23|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10044 (9.8K) [audio/x-wav]
Saving to: ‘FIB.wav’

FIB.wav             100%[===================>]   9.81K  --.-KB/s    in 0s

2023-12-17 11:31:48 (164 MB/s) - ‘FIB.wav’ saved [10044/10044]

--2023-12-17 11:31:48--  https://www.fit.vutbr.cz/study/courses/ISS/public/proj202
3-24/KES.wav
Resolving www.fit.vutbr.cz (www.fit.vutbr.cz)... 147.229.9.23, 2001:67c:1220:809::
93e5:917
Connecting to www.fit.vutbr.cz (www.fit.vutbr.cz)|147.229.9.23|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10044 (9.8K) [audio/x-wav]
Saving to: ‘KES.wav’

KES.wav             100%[===================>]   9.81K  --.-KB/s    in 0s

2023-12-17 11:31:48 (195 MB/s) - ‘KES.wav’ saved [10044/10044]
```

4. Úkoly

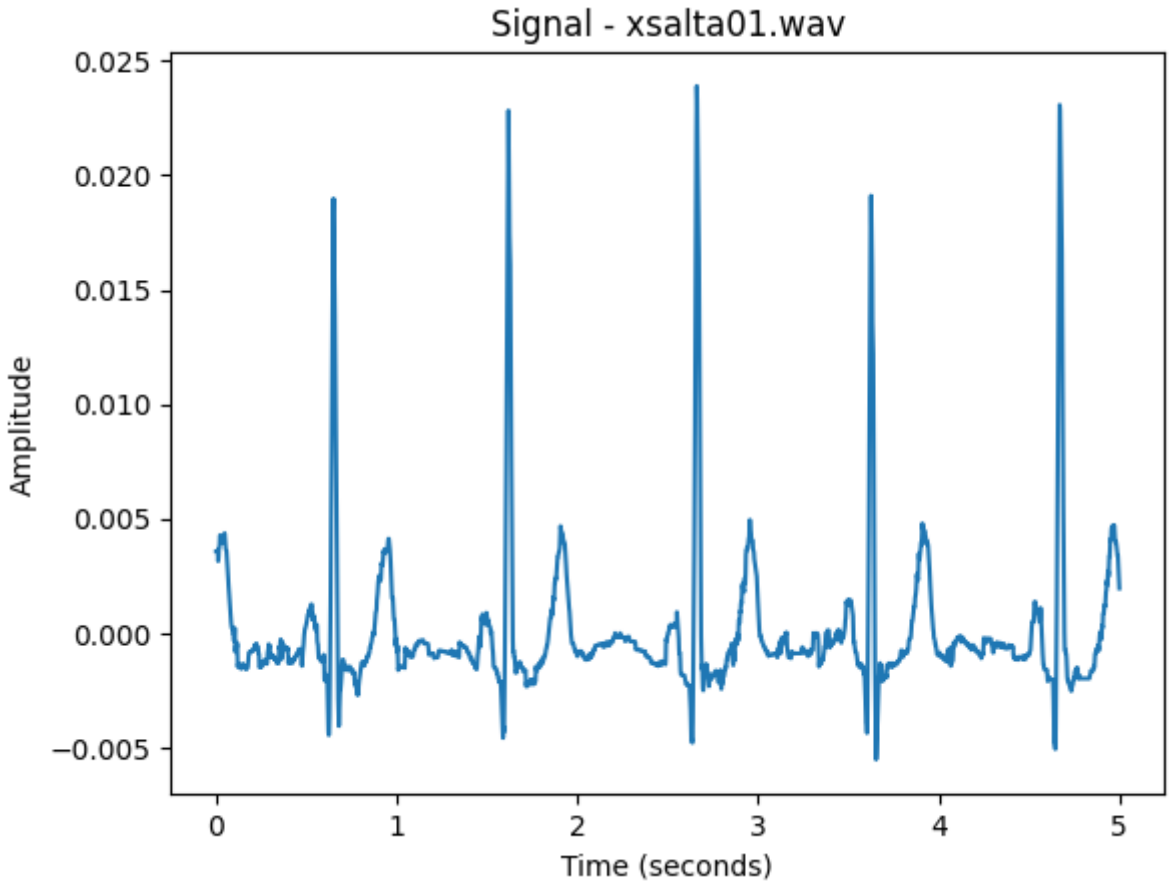
4.1. [2.5b] Nahrání a zobrazení EKG signálu

Nezapomeňte na popisy os u jednotlivých grafů.

a) [1b] Nahrajte EKG signál login.wav, vyberte 5-sekundový úsek a zobrazte ho v časové doméně. Pro nahrání signálu použijte knihovny numpy a soundfile.

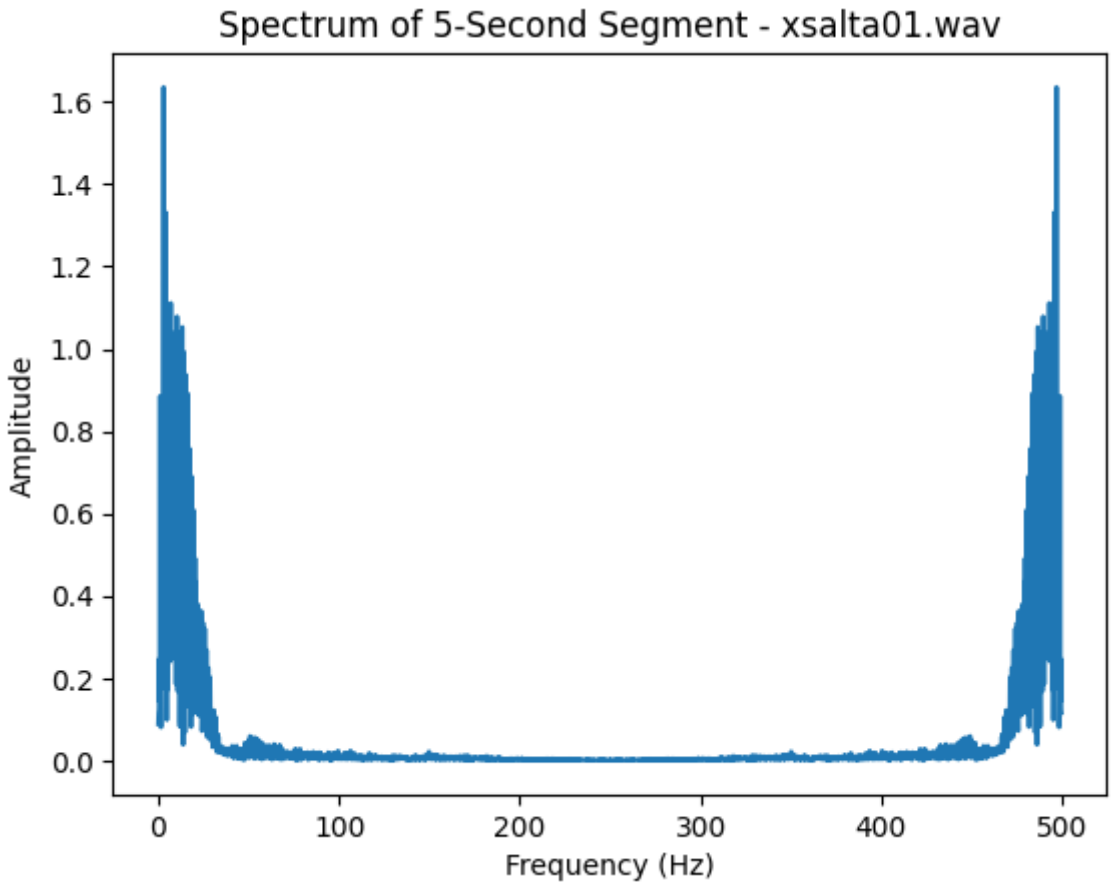
In [ ]:
```python
import numpy as np
import soundfile as sf
import matplotlib.pyplot as plt

start_time = 0
duration = 5
end_time = start_time + duration

start_idx = int(start_time * fs)
end_idx = int(end_time * fs)
segment_signal  = x[start_idx:end_idx]
```

```
time = np.linspace(start_time, end_time, len(segment_signal))

plt.plot(time, segment_signal)
plt.title('Signal - xsalta01.wav')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.show()
```



b) [1b] Spočítejte spektrum z 5 sekundového úseku nahraného signálu a zobrazte jej.

In [ ]:
```
# Calculate the FFT to obtain the spectrum
X = np.fft.fft(segment_signal)
N = len(segment_signal)
kall = np.arange(0,N) # here we run till N-1
Xmag = np.abs(X)
Xphase = np.angle(X)

f = kall / N * fs

# Plot the spectrum
plt.plot(f, Xmag)
plt.title('Spectrum of 5-Second Segment - xsalta01.wav')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.show()
```



c) [0.5b] Ve spektru vidíte rušení na 50Hz nebo 60Hz a jejich harmonických frekvencích. Vysvětlete, čím je způsobeno.

Interference at 50 Hz or 60 Hz and their harmonics usually comes from the mains voltage used for electrical equipment and lighting in the area. This interference is known as electrical noise or mains noise and can affect electrophysiological recordings such as ECGs. Here are some reasons why this can happen:

1. Mains voltage: Most countries use a mains voltage with a frequency of 50 Hz or 60 Hz. Electrical equipment and wiring can cause unwanted electromagnetic interference that can reach the electrodes when measuring bioelectrical signals.

2. Electromagnetic fields: Electrical equipment generates electromagnetic fields that can be picked up by the electrodes used for ECG. These fields can cause noise at specific frequencies and their harmonics.

3. Improper grounding: Poor grounding of electrical equipment or instruments can cause mains noise to penetrate the measured signals.

4.2. [3b] Převzorkujte nahraný signál

a) [2b] Převzorkujte signál na vzorkovací frekvenci 100 Hz, nezapomeňte na filtr pro antialiasing. Můžete například odstranit část spektra od $\frac{Fs}{2}$ nebo použít filtr dolní propusti.

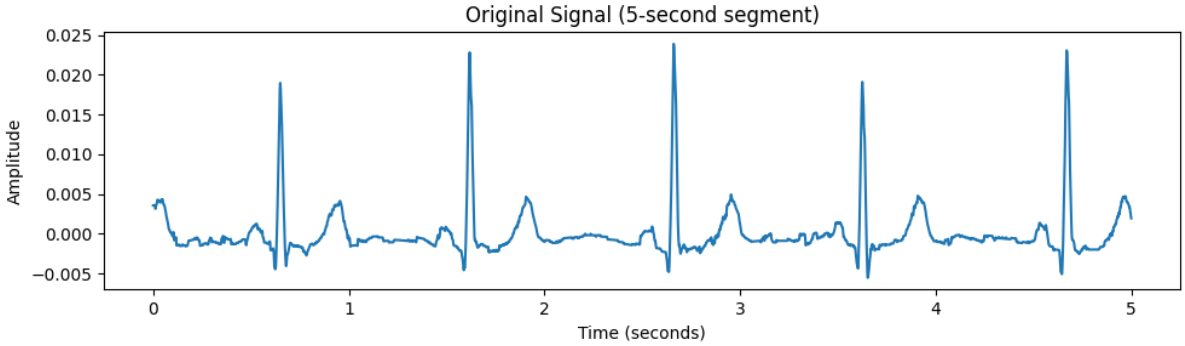
In [ ]:
```
from scipy import signal

# Target sampling frequency
fs_target = 100
```

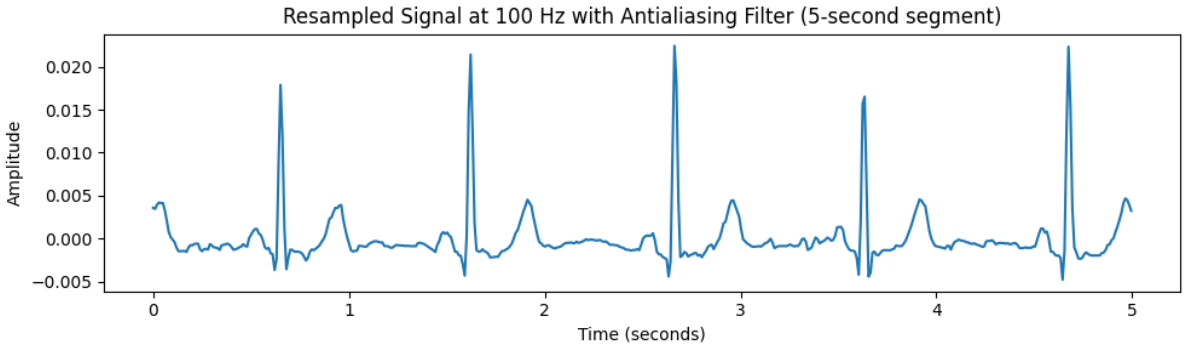
```
# Calculate the downsampling factor
downsampling_factor = fs // fs_target

# Antialiasing lowpass filter with a lower cutoff frequency
nyquist = 0.5 * fs
cutoff = 0.2 * nyquist
b, a = signal.butter(8, cutoff/nyquist, btype='lowpass')

# Apply the antialiasing filter
resampled_signal = signal.filtfilt(b, a, x)

# Downsample
resampled_signal = resampled_signal[::downsampling_factor]
```

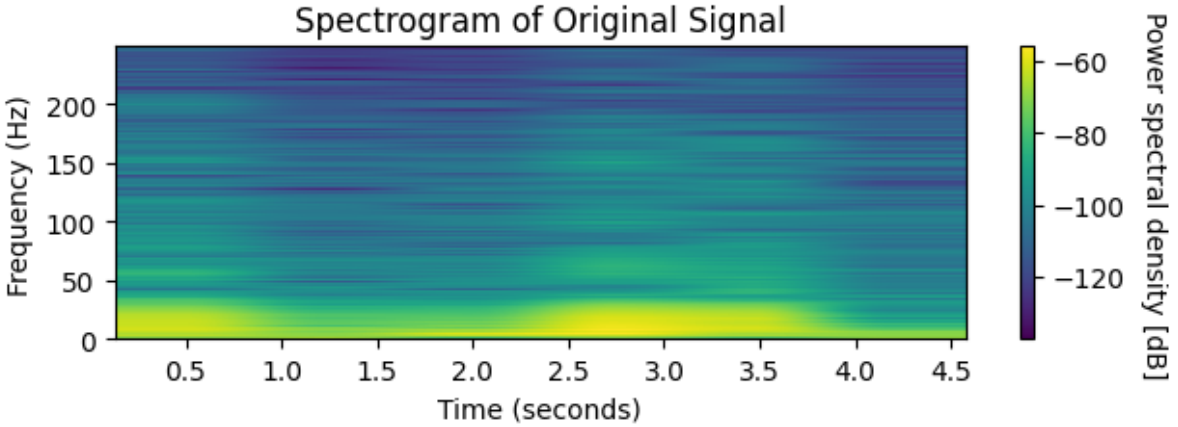
b) [1b] Zobrazte 5 sekundový úsek původního a převzorkovaného signálu v časové doméně a zobrazte i jejich spektra.

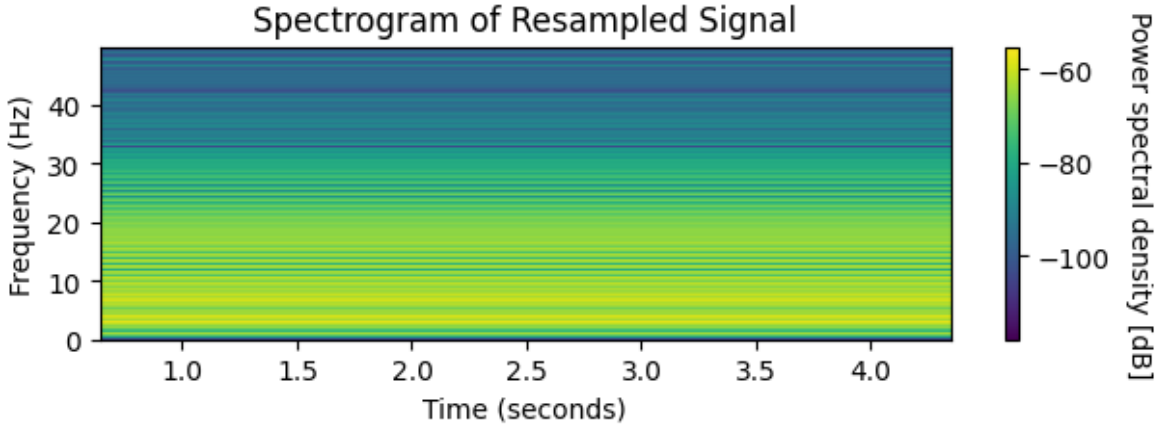
In [ ]:
```
# Define time arrays for original and resampled signals
time_original = np.linspace(0, len(x) / fs, len(x))
time_resampled = np.linspace(0, len(resampled_signal) / fs_target, len(resampled_si

# Choose a 5-second segment
start_time = 0
duration = 5
end_time = start_time + duration

start_idx = int(start_time * fs)
end_idx = int(end_time * fs)

# Select the segment for the original signal
segment_signal = x[start_idx:end_idx]

# Select the segment for the resampled signal
segment_signal_resampled = resampled_signal[:int(duration * fs_target)]

# Plot the original and resampled signals in the time domain
plt.figure(figsize=(10, 6))

plt.subplot(2, 1, 1)
plt.plot(time_original[start_idx:end_idx], segment_signal)
plt.title('Original Signal (5-second segment)')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')

plt.subplot(2, 1, 2)
plt.plot(time_resampled[:int(duration * fs_target)], segment_signal_resampled)
plt.title('Resampled Signal at 100 Hz with Antialiasing Filter (5-second segment)')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')

plt.tight_layout()
plt.show()

# Plot the spectrogram for the original signal
plt.subplot(2, 1, 1)
plt.specgram(segment_signal, Fs=fs, cmap='viridis', NFFT = 499)
plt.title('Spectrogram of Original Signal')
plt.xlabel('Time (seconds)')
plt.ylabel('Frequency (Hz)')

cbar = plt.colorbar()
cbar.set_label('Power spectral density [dB]', rotation=270, labelpad=15)
```

```
# Plot the spectrogram for the resampled signal
plt.subplot(2, 1, 2)
plt.specgram(segment_signal_resampled, Fs=fs_target, cmap='viridis', NFFT = 499)  #
plt.title('Spectrogram of Resampled Signal')
plt.xlabel('Time (seconds)')
plt.ylabel('Frequency (Hz)')

cbar = plt.colorbar()
cbar.set_label('Power spectral density [dB]', rotation=270, labelpad=15)

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





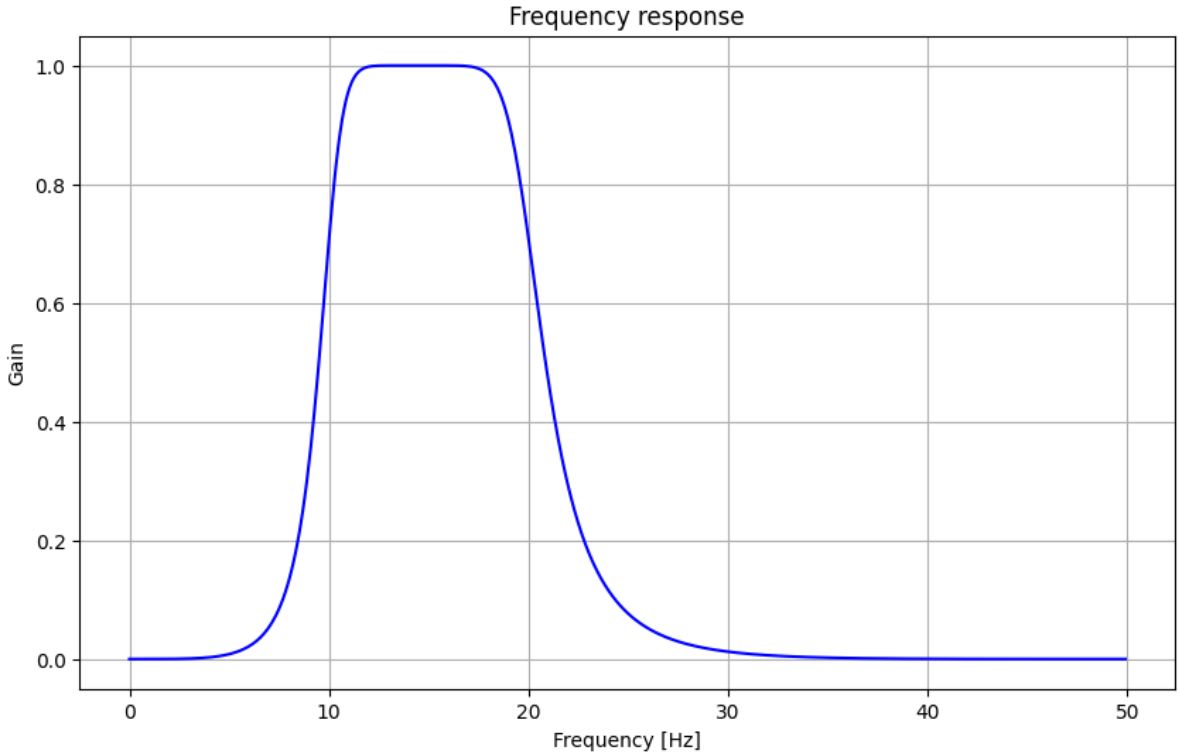


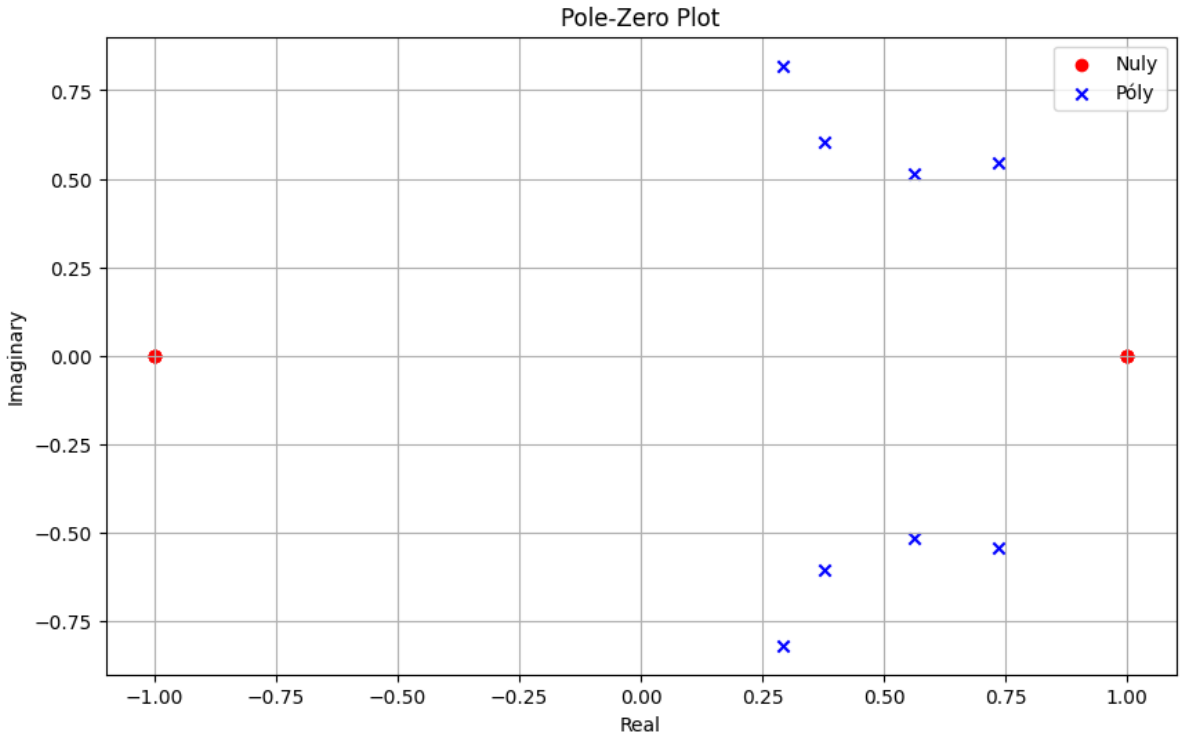


:### 4.3. [4b] Vyfiltrujte nahraný signál pásmovou propustí 10Hz-20Hz

a) [2b] Vytvořte filtr pásmové propusti, možnosti jsou dvě: buďto filtrovat pomocí klasického návrhu filtrů, kde získáte koeficienty `a` a `b` (pomocí např. scipy.butter) a zobrazíte charakteristiku filtru + nuly a póly. Nebo se můžete vydat cestou filtrování ve frekvenční doméně, frekvenční charakteristiku vykreslete pomocí spektrální masky.

In [ ]:
```
# Creating a band pass filter
lowcut = 10
highcut = 20
nyquist = 0.5 * fs_target
low = lowcut / nyquist
high = highcut / nyquist

# Butterworth filter design
order = 4
b, a = signal.butter(order, [low, high], btype='band')

# Display filter characteristics
w, h = signal.freqz(b, a, fs=fs_target)
plt.figure(figsize=(10, 6))
plt.plot(w, np.abs(h), 'b')
plt.title('Frequency response')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Gain')
plt.grid()
plt.show()

# Show zeros and poles
z, p, _ = signal.tf2zpk(b, a)
plt.figure(figsize=(10, 6))
plt.scatter(np.real(z), np.imag(z), marker='o', color='red', label='Nuly')
plt.scatter(np.real(p), np.imag(p), marker='x', color='blue', label='Póly')
plt.title('Pole-Zero Plot')
plt.xlabel('Real')
plt.ylabel('Imaginary')
plt.legend()
plt.grid()
plt.show()
```

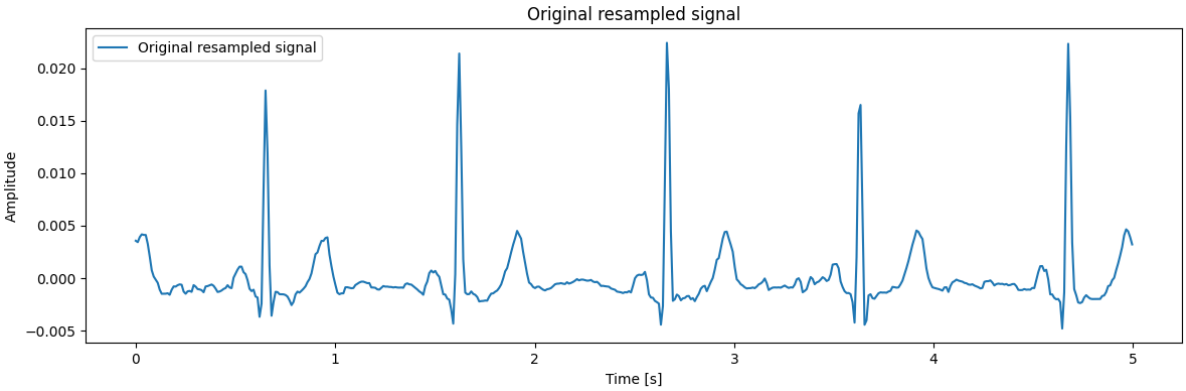


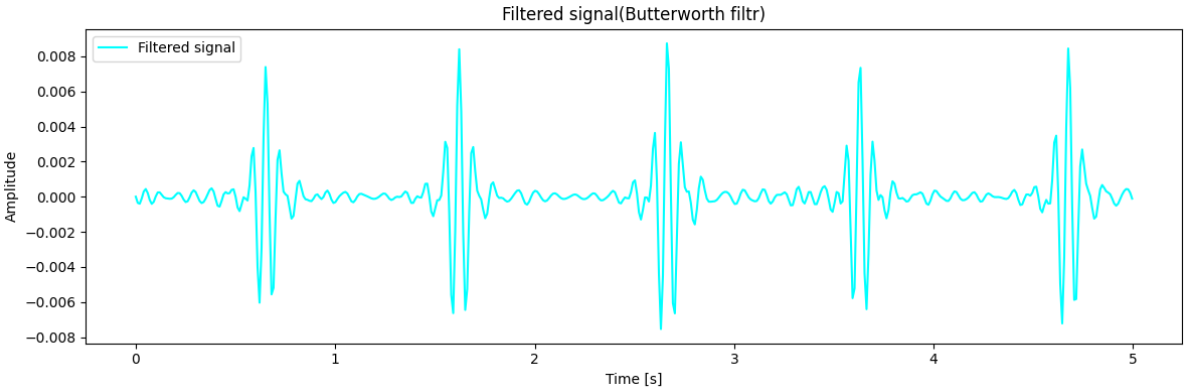


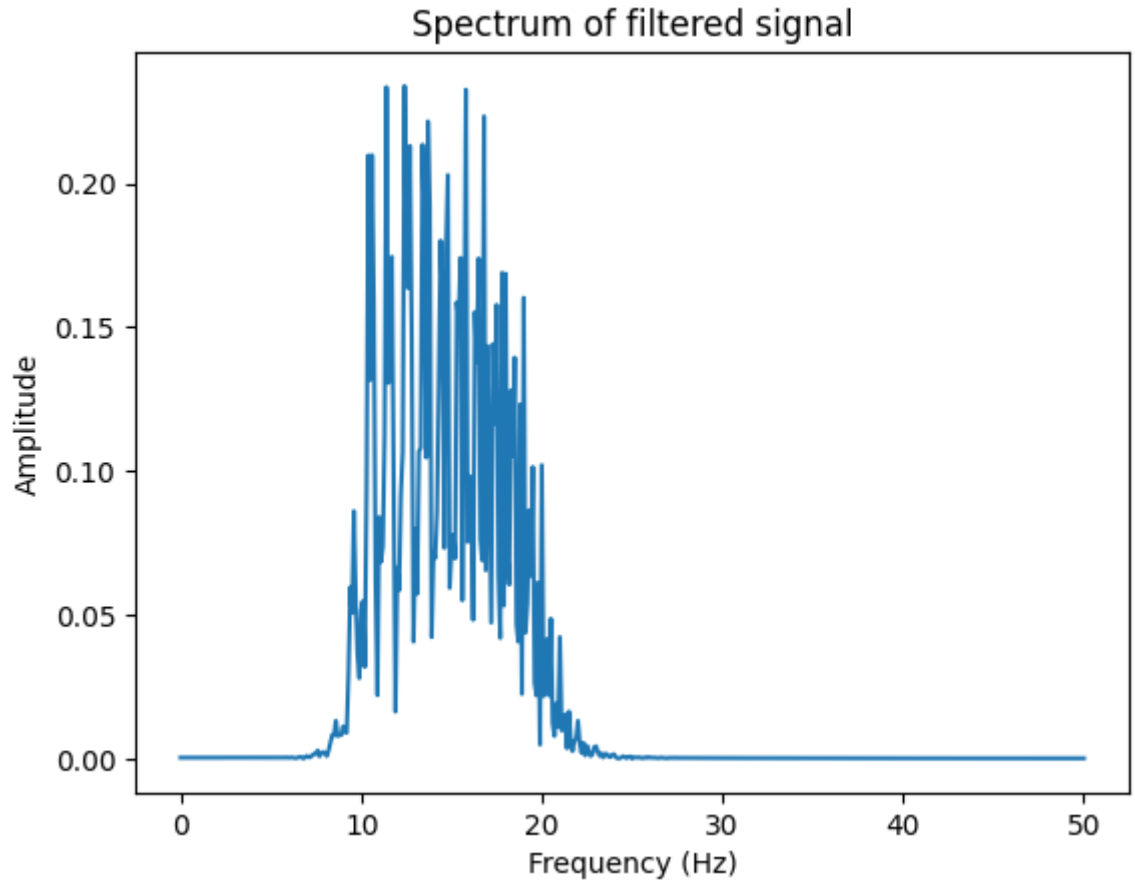
b) [1b] Použijte navržený filtr na nahraný signál. Pokud máte navržený klasický filtr, proveďte filtrování z obou stran, abyste se vyhnuli fázovému posunu, to za vás zajistí například funkce scipy.signal.filtfilt. Vykreslete původní a vyfiltrovaný signál v časové doméně a spočítejte a zobrazte jejich spektra.

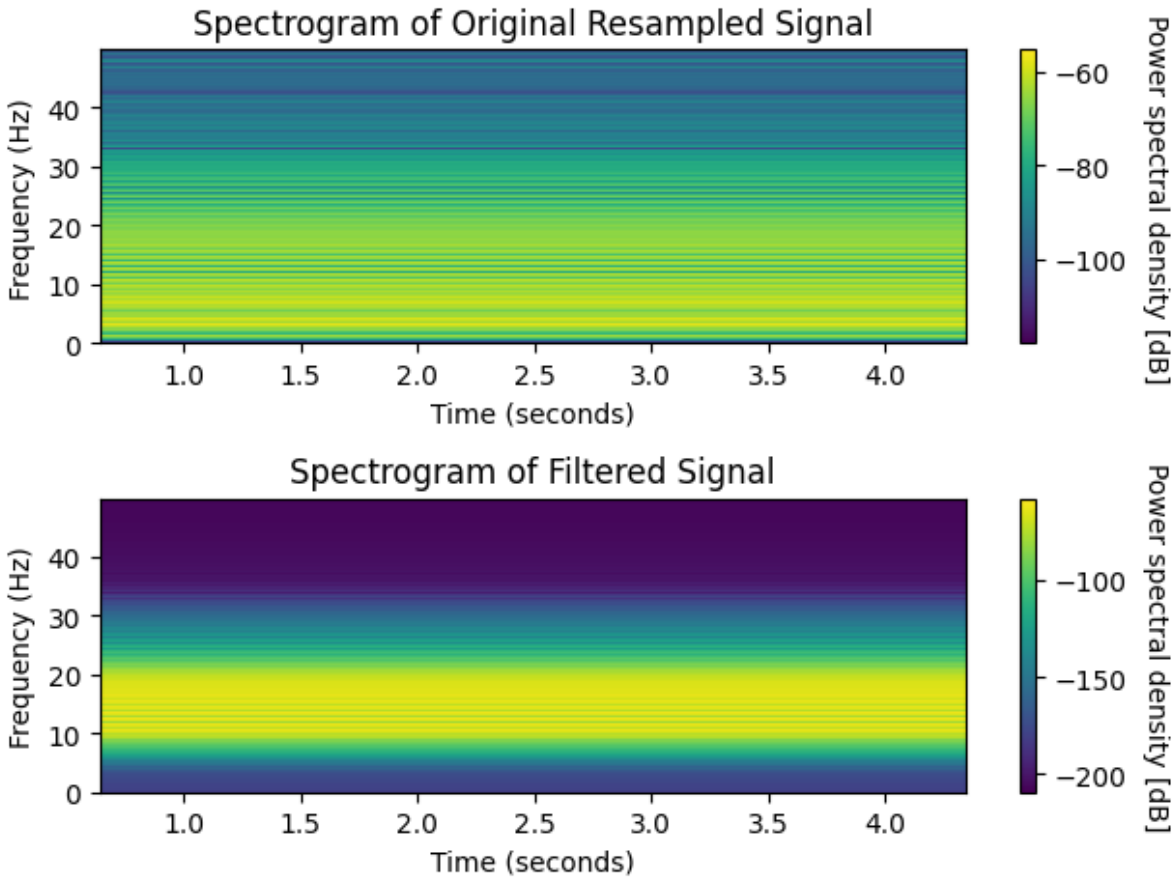
In [26]:
```
from matplotlib import cm
from scipy.signal import spectrogram, lfilter, freqz, tf2zpk, ellipord, ellip

# Filter the signal
filtered_signal = signal.filtfilt(b, a, resampled_signal)

# Rendering of original and filtered signal
plt.figure(figsize=(12, 8))
```

```
time_resampled_subset = time_resampled[:int(duration * fs_target)]

plt.subplot(2, 1, 1)
plt.plot(time_resampled_subset, segment_signal_resampled, label = 'Original resampl
plt.title('Original resampled signal')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(time_resampled_subset, filtered_signal[:len(time_resampled_subset)], label
plt.title('Filtered signal(Butterworth filtr)')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()

plt.tight_layout()
plt.show()

##### only half of spectrum

# Calculate the FFT (Fast Fourier Transform) to obtain the spectrum
X = np.fft.fft(filtered_signal)
N = len(filtered_signal)
kall = np.arange(0, int(N/2) + 1) # here we run till N/2+1 !
Xmag = np.abs(X[kall])
Xphase = np.angle(X[kall])

f = kall / N * fs_target

# Plot the spectrum
plt.plot(f, Xmag)
plt.title('Spectrum of filtered signal')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude')
plt.show()
#####


# Plot the spectrogram for the original resampled signal
plt.subplot(2, 1, 1)
plt.specgram(segment_signal_resampled, Fs=fs_target, cmap='viridis', NFFT = 499)
plt.title('Spectrogram of Original Resampled Signal')
plt.xlabel('Time (seconds)')
plt.ylabel('Frequency (Hz)')

cbar = plt.colorbar()
cbar.set_label('Power spectral density [dB]', rotation=270, labelpad=15)

# Plot the spectrogram for the filtered signal
plt.subplot(2, 1, 2)
plt.specgram(filtered_signal[:len(time_resampled_subset)], Fs=fs_target, cmap='viri
plt.title('Spectrogram of Filtered Signal')
plt.xlabel('Time (seconds)')
plt.ylabel('Frequency (Hz)')

cbar = plt.colorbar()
cbar.set_label('Power spectral density [dB]', rotation=270, labelpad=15)

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


Original resampled signal
Original resampled signal
Amplitude
Time [s]


Filtered signal(Butterworth filtr)
Filtered signal
Amplitude
Time [s]


Spectrum of filtered signal
Amplitude
Frequency (Hz)



c) [1b] Okomentujte rozdíl mezi filtrovaným a nefiltrovaným signálem a jejich spektry. Pokud bychom použili filtrování pouze z jedné strany (obyčejnou konvoluci), jaké je teoreticky největší posunutí ve vzorcích, které se může objevit a proč?

Difference between filtered and unfiltered signal:

Time domain:

Unfiltered signal: Various frequency components may be present in the unprocessed, unfiltered signal, including those we wish to remove.

Filtered signal: After applying a band-pass filter, frequency components outside the target region (in this case 10-20 Hz) are suppressed, resulting in the removal of unwanted frequencies.

Spectral Domain:

Filtered signal: In the spectrum of the unfiltered signal we can see the presence of a wide range of frequency components, including those outside the target domain.

Filtered signal: In the spectrum of the filtered signal, there should be significant suppression of frequencies outside the 10-20 Hz band.

Filtering from one side only:

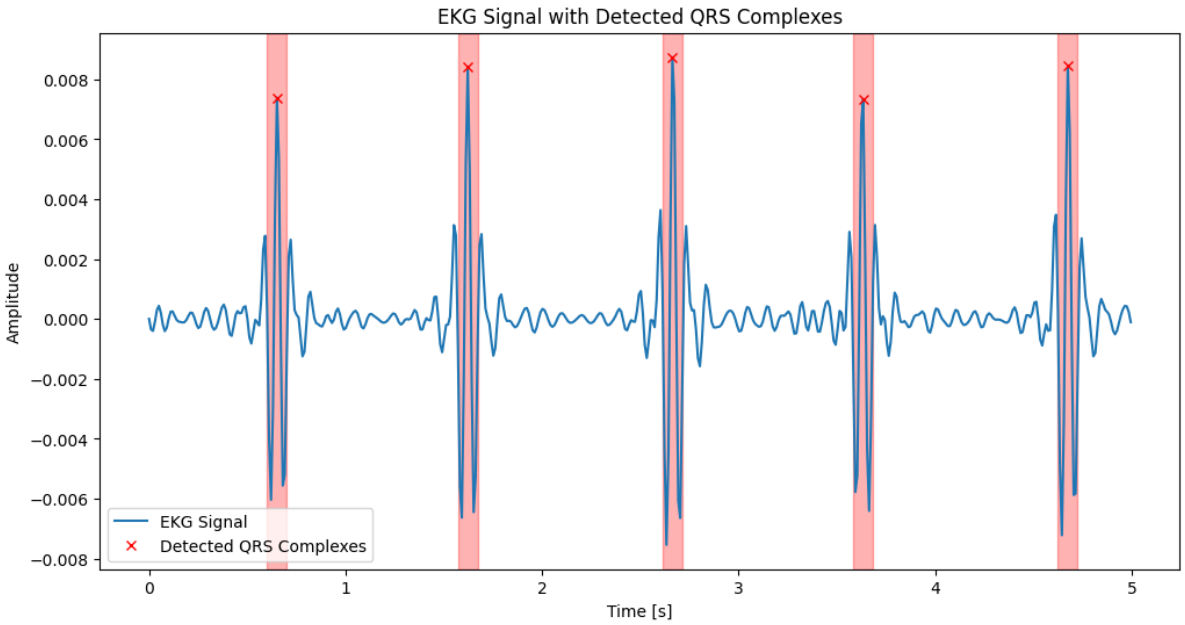
The use of one-sided filtering (for example, using convolution without signal reversal) will cause a phase shift in the filtered signal. The largest possible shift in the patterns that can occur corresponds to the position of the farthest frequency from the center of the filter band. If we filtered the signal with a bandpass of 10 Hz - 20 Hz and used only one-sided

convolution, the largest shift in the signal would be caused by the frequency furthest from 10 Hz or 20 Hz. This shift would be reflected by phase delays in the individual frequency components of the signal.

4.4. [3b] Vytvořte detektor QRS v časové doméně. Detekované QRS komplexy uložte do vhodné struktury a zároveň zobrazte graf v časové ose se zvýrazněnými QRS detekcemi.

a) [1b] Detekujte QRS v převzorkovaném vyfiltrovaném signálu pomocí tresholdu (prahu). Pro tuto detekci musíte nejdříve získat vzorek jednoho QRS ze signálu, spočítat si maximální amplitudu a jako treshold vzít vámi určené procento této hodnoty. **Dávejte pozor na možnost otočeného QRS v signálu.** Do vykresleného signálu s detekcemi vykreslete i čáru udávající použitý treshold.

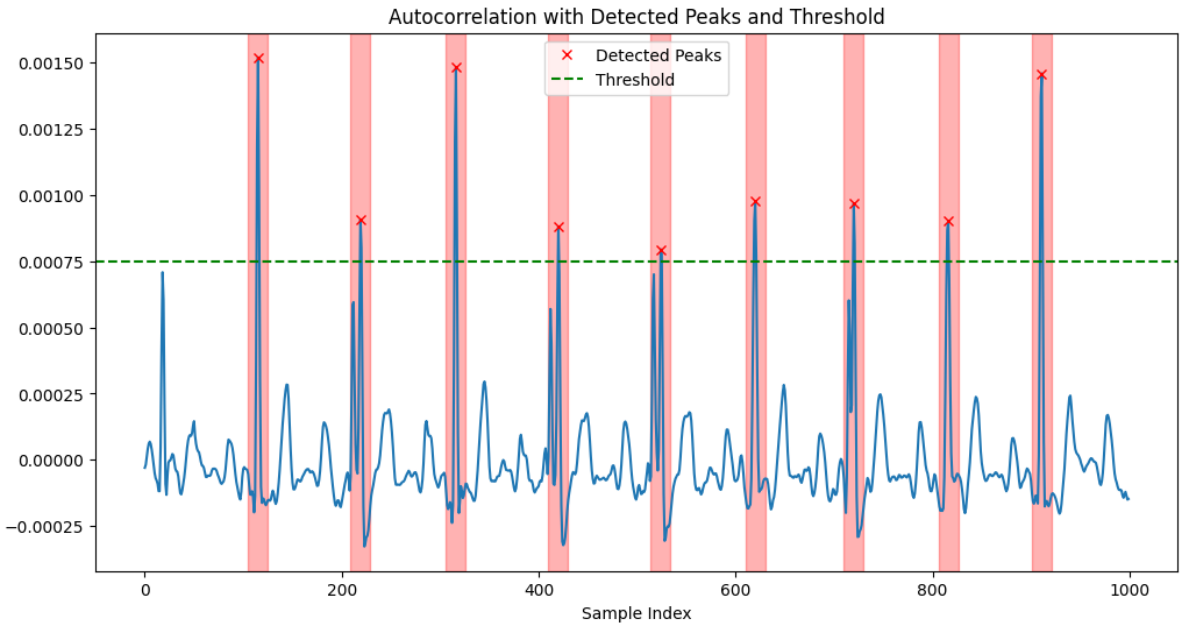
In [ ]:
```
from scipy.signal import find_peaks

# Recorded ECG signal (example, replace with your signal)
fs = fs_target
t = time_resampled_subset
ekg_signal = filtered_signal[:len(t)]  # Adjust the length of ekg_signal

# QRS complex detection using thresholding
max_amplitude = np.max(ekg_signal)
threshold_percentage = 2
threshold = max_amplitude * (threshold_percentage / 100)
qrs_peaks, _ = find_peaks(ekg_signal, height=threshold, distance=70)

# Creating a structure for storing detected QRS complexes
qrs_complexes = [{'index': peak, 'amplitude': ekg_signal[peak]} for peak in qrs_pea

# Graph display with detected QRS complexes
plt.figure(figsize=(12, 6))
plt.plot(t, ekg_signal, label='EKG Signal')
plt.plot(t[qrs_peaks], ekg_signal[qrs_peaks], 'rx', label='Detected QRS Complexes')

# Highlight the entire width of the graph for each detected QRS complex
for peak in qrs_peaks:
    plt.axvspan(t[peak - 5], t[peak + 5], alpha=0.3, color='red')

plt.title('EKG Signal with Detected QRS Complexes')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()
plt.show()
```



b) [2b] Detekujte QRS v signálu pomocí autokorelace v převzorkovaném nefiltrovaném signálu. Pro tuto detekci musíte nejdříve získat vzorek jednoho QRS ze signálu. Dále budete autokorelovat signál právě s tímto výstřižkem. QRS se budou nacházet na místech, kde vám budou vycházet vysoké hodnoty korelace. Do vykresleného signálu s detekcemi zaznačte i vámi zvolený výstřižek.

In [ ]:

```
# Select a sample of one QRS complex
QRS_sample = resampled_signal[qrs_peaks[0]:qrs_peaks[0] + int(fs_target)]  # První

# Autocorrelation of the signal with the selected QRS pattern
autocorr = np.correlate(resampled_signal, QRS_sample, mode='same')

# Detection of QRS complexes by thresholding on the resulting autocorrelation
threshold_autocorr = 0.00075
qrs_peaks_autocorr, _ = find_peaks(autocorr, height=threshold_autocorr, distance=25

# Creating a structure for storing detected QRS complexes using autocorrelation
qrs_complexes_autocorr = [{'index': peak, 'amplitude': autocorr[peak]} for peak in

# Plot the autocorrelation with detected peaks and threshold
plt.figure(figsize=(12, 6))
plt.plot(autocorr)
plt.plot(qrs_peaks_autocorr, autocorr[qrs_peaks_autocorr], 'rx', label='Detected Pe
plt.axhline(y=threshold_autocorr, color='g', linestyle='--', label='Threshold')

for peak in qrs_peaks_autocorr:
    plt.axvspan(peak - 10, peak + 10, alpha=0.3, color='red')  # Adjust the width (

plt.title('Autocorrelation with Detected Peaks and Threshold')
plt.xlabel('Sample Index')
plt.legend()
plt.show()
```

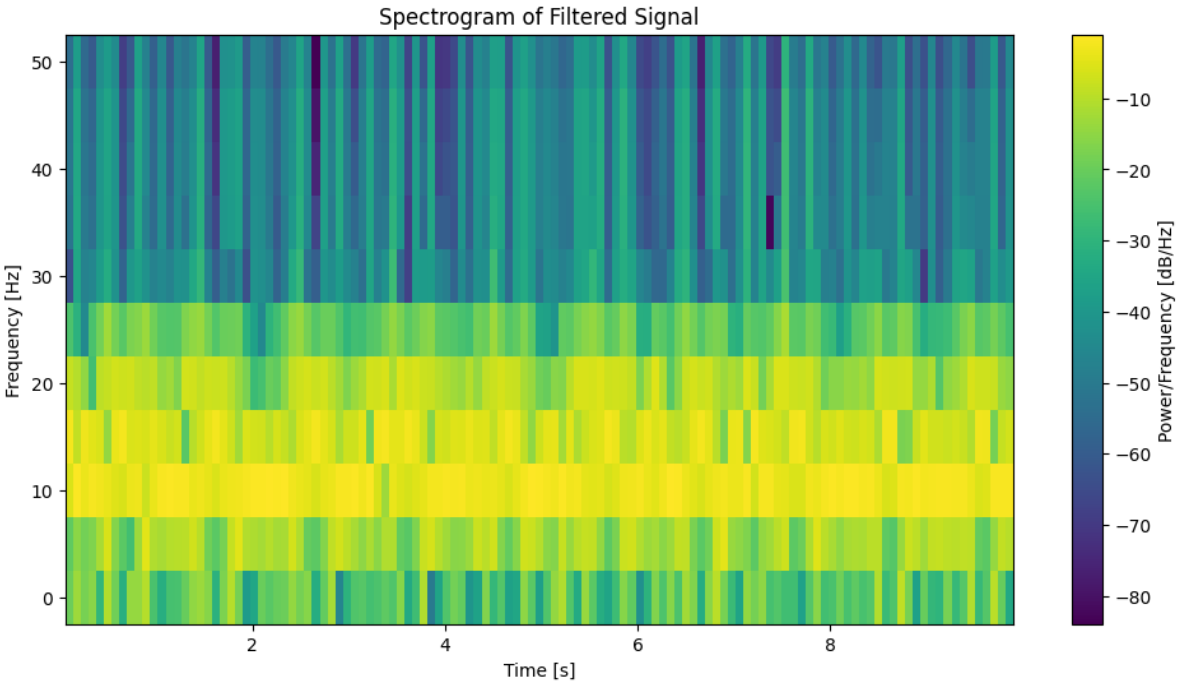


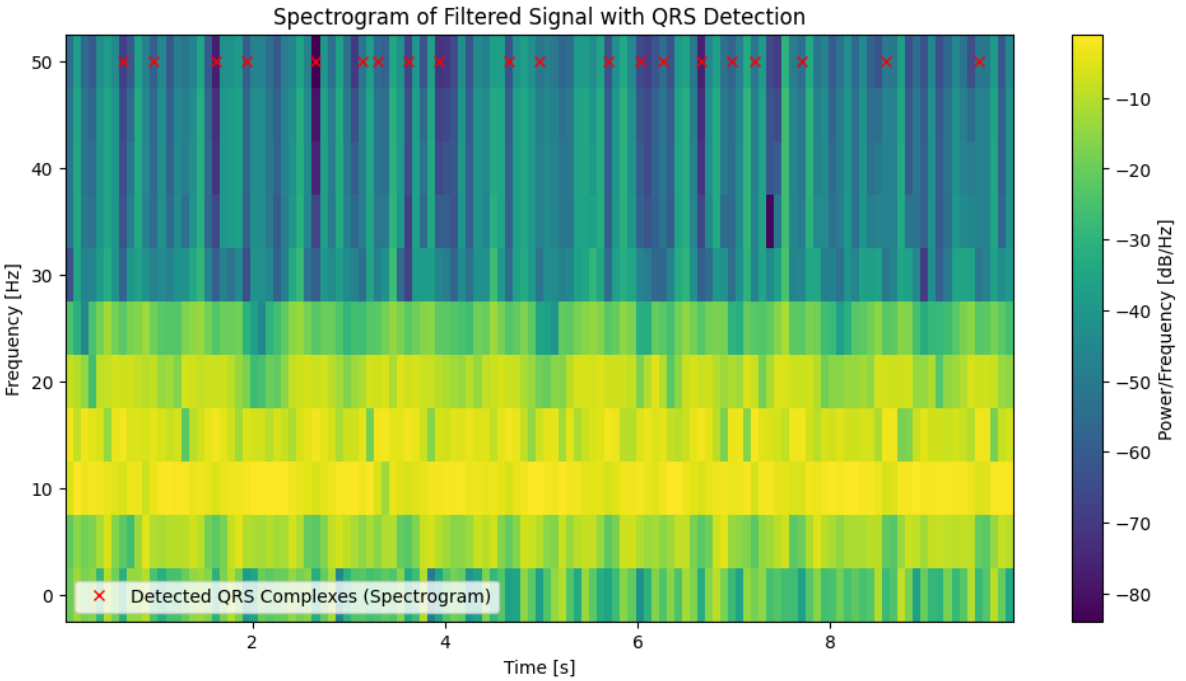
4.5. [3.5b] Vytvořte detektor QRS v frekvenční doméně a detekované QRS zakreslete jako v předchozí úloze 4.4

a) [2b] Detekujte QRS pomocí použití spektrogramu. Spočítejte a zobrazte spektrogram nahraného převzorkovaného filtrovaného signálu. Použijte parametry, `hop_size=120ms` a `window_len=200ms`, popřípadě si zkuste s těmito parametry pohrát. Spektrogram dále normalizujte v čase. Spočítejte sumy energie spektra pro jednotlivé časové biny. Dále vytvořte práh podle hodnoty energie spektra u prvního vámi zvoleného QRS komplexu. Tento práh použijte pro detekci zbylých QRS komplexů v signálu.

In [ ]:
```
# Parameters for spectrogram
hop_size = int(0.12 * fs_target)  # 120 ms
window_len = int(0.2 * fs_target)  # 200 ms

# Creating a spectrogram
frequencies, times, Sxx = spectrogram(filtered_signal, fs=fs_target, window='hammir

# Normalization of spectrogram in time
normalized_spectrogram = np.abs(Sxx) / np.sum(np.abs(Sxx), axis=0, keepdims=True)

# Spectrogram display
plt.figure(figsize=(12, 6))
plt.pcolormesh(times, frequencies, 10 * np.log10(normalized_spectrogram), shading='
plt.title('Spectrogram of Filtered Signal')
plt.xlabel('Time [s]')
plt.ylabel('Frequency [Hz]')
plt.colorbar(label='Power/Frequency [dB/Hz]')
plt.show()

# Calculating the sum of the spectrum energy for each time bin
energy_sum = np.sum(np.abs(Sxx), axis=0)

# Threshold detection by spectrum energy at the first QRS complex
threshold_energy = energy_sum[qrs_peaks[0]]

# QRS complex detection based on threshold
qrs_peaks_spectrogram, _ = find_peaks(energy_sum, height=threshold_energy)

# Creating a structure to store detected QRS complexes using spectrogram
qrs_complexes_spectrogram = [{'index': peak, 'energy': energy_sum[peak]} for peak i
```

```
# Graph display with QRS complex detection using spectrogram
plt.figure(figsize=(12, 6))
plt.pcolormesh(times, frequencies, 10 * np.log10(normalized_spectrogram), shading='
plt.plot(times[qrs_peaks_spectrogram], np.ones_like(qrs_peaks_spectrogram) * freque
plt.title('Spectrogram of Filtered Signal with QRS Detection')
plt.xlabel('Time [s]')
plt.ylabel('Frequency [Hz]')
plt.colorbar(label='Power/Frequency [dB/Hz]')
plt.legend()
plt.show()
```





b) [1b] Detekujte QRS pomocí použití obálek a Hilbertovy transformace.

Hilbertova transformace je spočítaná podle následujícího vzorce

$x_a = F^{-1}(F(x)2U) = x + iy,$

kde F je Fourierova transformace a F^{-1} je její zpětná varianta. U je Heavisideova funkce neboli funkce jednotkového skoku, která je definována: $U(x)$:

$$\begin{cases} 0.5 & x = 0 \\ 1 & 0 < x < \frac{N}{2} \text{ pro } N \text{ liché} \\ 0.5 & x = \frac{N}{2} \text{ pro } N \text{ liché} \\ 1 & 0 < x \leq \frac{N}{2} \text{ pro } N \text{ sudé} \\ 0 & jinak \end{cases}$$

kde N je počet koeficientů Fourierovy transformace - pokud není určeno jinak, je to počet vzorků signálu.

Jinými slovy obálku spočítate tak, že:

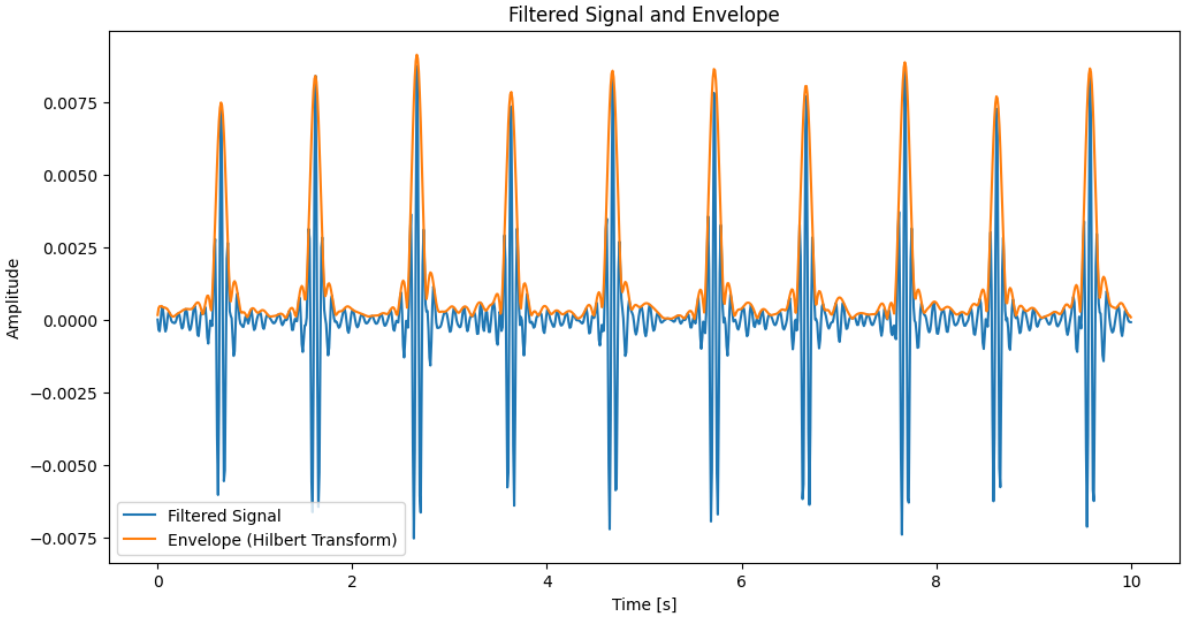
- Spočítáte FFT F na filtrovaném a převzorkovaném signálu
- Vynulujete pravou symetrickou část spektra
- Levou část spektra vynasobíte 2 kromě prvního a prostředního binu (při sudém počtu frekvenčních binů).
- Provedete zpětnou FFT F^{-1}

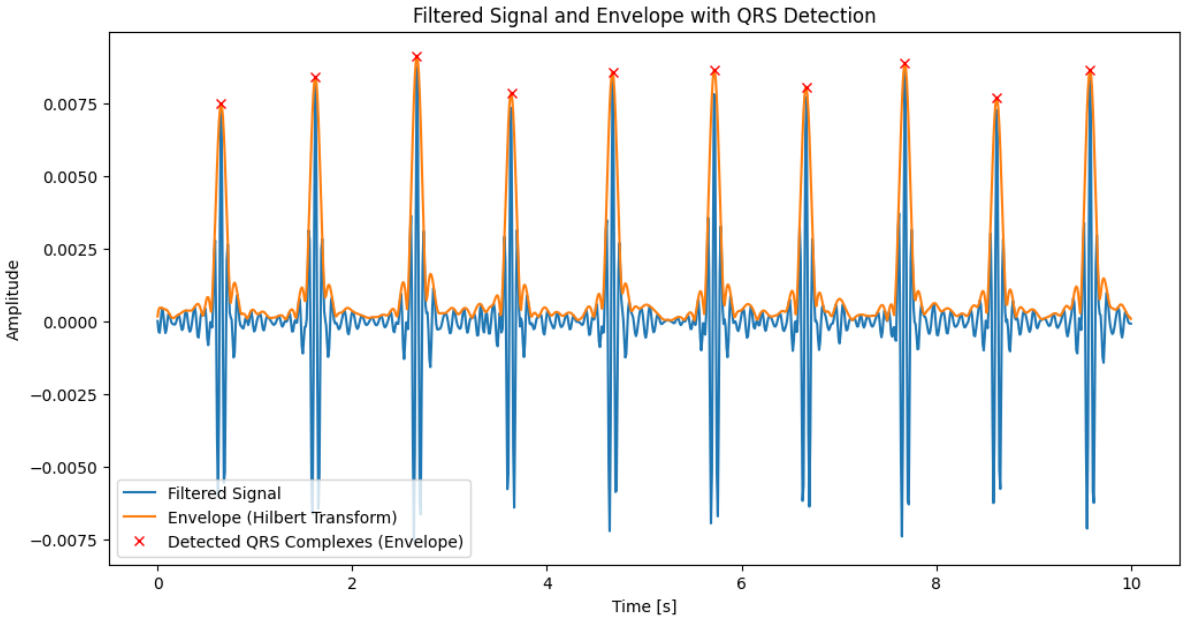
Abyste získali obálku signálu, je třeba vzít absolutní hodnotu signálu získaného Hilbertovou transformací.

Obálku a signál vykreslete do jednoho grafu přes sebe, obálka by měla obalovat daný signál.

In [ ]:
```
# FFT calculation on filtered and oversampled signal
fft_result = np.fft.fft(filtered_signal)
N = len(fft_result)

# Resetting the right symmetric part of the spectrum
fft_result[N//2+1:] = 0

# Multiplication of the left part of the spectrum
fft_result[1:N//2] *= 2

# Backward FFT
hilbert_result = np.fft.ifft(fft_result)

# Obtaining the signal envelope (absolute values of the Hilbert transform)
envelope = np.abs(hilbert_result)

# Signal and envelope display
plt.figure(figsize=(12, 6))
plt.plot(time_resampled, filtered_signal, label='Filtered Signal')
plt.plot(time_resampled, envelope, label='Envelope (Hilbert Transform)')
plt.title('Filtered Signal and Envelope')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()
plt.show()

# Detection of QRS complexes using envelope thresholding
threshold_envelope = 0.006
qrs_peaks_envelope, _ = find_peaks(envelope, height=threshold_envelope, distance=50

# Creating a structure to store detected QRS complexes using an envelope
qrs_complexes_envelope = [{'index': peak, 'amplitude': envelope[peak]} for peak in
```

```
# Chart display with QRS complex detection using envelope
plt.figure(figsize=(12, 6))
plt.plot(time_resampled, filtered_signal, label='Filtered Signal')
plt.plot(time_resampled, envelope, label='Envelope (Hilbert Transform)')
plt.plot(time_resampled[qrs_peaks_envelope], envelope[qrs_peaks_envelope], 'rx', la
plt.title('Filtered Signal and Envelope with QRS Detection')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()
plt.show()
```





c) [0.5b] Při kterých metodách detekcí QRS nám vadí otočený (flipnutý) signál, při kterých ne a proč?

The flipped signal may affect some QRS complex detection methods, especially those that are sensitive to signal polarity. Here are some methods that could be affected:

1. **Thresholding:** Thresholding-based methods may be sensitive to signal polarity. If the QRS complex is positive in the original signal, its amplitude may be negative in the flipped signal and vice versa. This may cause the thresholding values to not be set correctly for detection.

2. **Autocorrelation:** Similar to thresholding, autocorrelation can be sensitive to signal polarity. Autocorrelation results may differ between the original and flipped signal.

3. **Waveform-based methods:** Some methods for detecting QRS complexes may be designed with the assumption of a specific QRS waveform. If the signal is flipped, the waveform may change and affect detection.

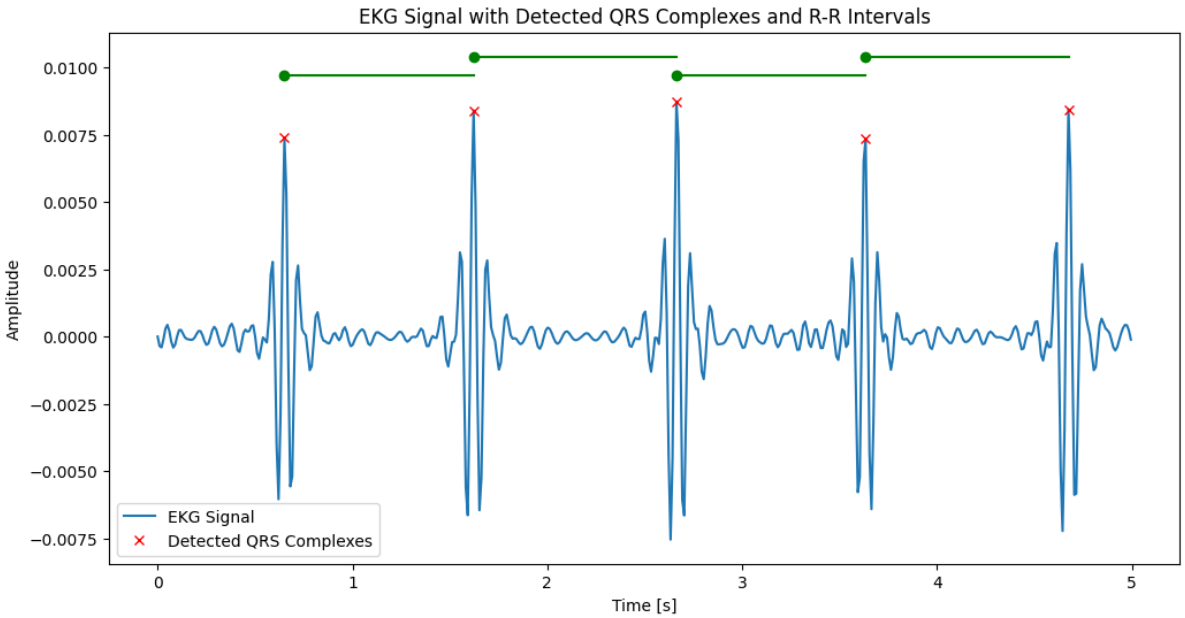
- Methods that are independent of signal polarity and that may be less sensitive to signal flipping include:*

1. **Envelope methods:** Methods based on signal envelopes, such as the Hilbert transform, may be less sensitive to polarity because they work with the amplitude of the signal independently of its polarity.

2. **Methods based on frequency analysis:** Some QRS detection methods may be based on characteristics of the frequency content of the signal that are invariant to polarity change.

4.6 [2b] Detekce R-R intervalu

In [ ]:
```
# Obtaining the time positions of detected QRS
qrs_positions = t[qrs_peaks]

# R-R interval calculation
rr_intervals = np.diff(qrs_positions)

rr_times = qrs_positions[:-1]

# Rendering of ECG signal with detected QRS
plt.figure(figsize=(12, 6))
plt.plot(t, ekg_signal, label='EKG Signal')

# Plotting of detected QRS
plt.plot(qrs_positions, ekg_signal[qrs_peaks], 'rx', label='Detected QRS Complexes'

plt.scatter(rr_times, rr_intervals*0.01, color='g')

# Plotting R-R intervals over the ECG signal
for i, rr_interval in enumerate(rr_intervals):
    plt.plot([qrs_positions[i], qrs_positions[i + 1]], [rr_interval*0.01, rr_interv


plt.title('EKG Signal with Detected QRS Complexes and R-R Intervals')
plt.xlabel('Time [s]')
plt.ylabel('Amplitude')
plt.legend()
plt.show()
```


EKG Signal with Detected QRS Complexes and R-R Intervals
Amplitude
Time [s]
EKG Signal
Detected QRS Complexes
0.0100
0.0075
0.0050
0.0025
0.0000
−0.0025
−0.0050
−0.0075
0
1
2
3
4
5