Medical Informatics-Signal Correletion
Assignment 2

Vasileios Nastos
MSc in department of Informatics
and Telecommunications
University of Ioannina
Arta, 47100, Greece
Id:137

May 17, 2023

1 ECG

ECG (Electrocardiogram) data refers to recordings of the electrical activity of the heart over time. ECG recordings are obtained by placing electrodes on the skin of a person's chest, arms, and legs, which measure the electrical activity of the heart and transmit it to a recording device. The resulting ECG data typically consists of a time series of voltage measurements that reflect the electrical activity of the heart during each heartbeat. The ECG waveform is characterized by a series of distinct peaks and valleys, each of which corresponds to a different phase of the cardiac cycle. ECG data is commonly used in medical applications to diagnose and monitor heart conditions, such as arrhythmias, heart attacks, and other cardiovascular diseases. ECG data can also be used in research to study the electrical activity of the heart and to develop new methods for analyzing and interpreting ECG recordings. The analysis of ECG data involves various techniques such as signal processing, feature extraction, and machine learning. These techniques are used to extract meaningful information from the ECG waveform, such as heart rate variability, QRS complex duration, and ST segment changes, which can be used to diagnose and monitor various heart conditions.

For this assignment we use a subset of the data that are single column(Specififc frequency). Descriptive analytics for the data are presented in table 1

Statistic Meter	**Value**
Samples	900.000
Mean	0.197
Median	0.020
Std	0.554
Iqr	0.090
Skewness	2.277
Kurtosis	6.642

Table 1: Statistic meters about ecg

2 Code

Listing 1: Πλήρης Κώδικας Άσκησης 2

```
import numpy as np,os
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
import statistics
from scipy.stats import skew,kurtosis
```

```
from tabulate import tabulate


class ECG:
    def __init__(self) -> None:
        self.signal=np.loadtxt('ecg.txt')
        self.sliced_signal=self.signal[100:1000]
        self.frequency=1000
        self.cutoff_freq=None
        self.fig=plt.figure(figsize=(13,10))
        self.gs=GridSpec(4,2,figure=self.fig)
        self.rowc=0
        self.colc=0
        self.filename=None

    def set_filepath(self,fn):
        self.filename=fn

    def update_column_counter(self):
        self.colc+=1
        if self.colc!=0 and self.colc%2==0:
            self.rowc+=1
            self.colc=0

    def statistics(self):
        q1,q3,_=statistics.quantiles(data=self.sliced_signal,n=4)
        rows=[
            ['Samples',self.sliced_signal.shape[0]],
            ['Mean',statistics.mean(self.sliced_signal)],
            ['Median',statistics.median(self.sliced_signal)],
            ['Std',statistics.stdev(self.sliced_signal)],
            ['Iqr',q3-q1],
            ['Skewness',skew(self.sliced_signal)],
            ['Kurtosis',kurtosis(self.sliced_signal)]
        ]
        print(tabulate(tabular_data=rows,headers=['Statistic Meter','Value'],tablefmt='fancy_grid',
             floatfmt='.3f'))
        with open(os.path.join('stats.tex'),'w') as writer:
            writer.write(tabulate(tabular_data=rows,headers=['Statistic Meter','Value'],tablefmt='latex',
                 floatfmt='.3f'))

    def plot_signal(self,in_axis=False):
        if not in_axis:
            plt.figure(figsize=(15,6))
            plt.plot(np.arange(self.signal.shape[0]),self.signal)
            plt.ylim(self.signal.min(),self.signal.max())
            plt.xticks(np.arange(0,self.signal.shape[0],5000))
            plt.xlabel("n(samples)")
            plt.ylabel('ecg')
            ax=plt.gca()
            ax.spines['top'].set_visible(False)
            ax.spines['right'].set_visible(False)
            plt.title('Full signal plot')
            plt.savefig(os.path.join('','figures','sliced_signal_full.png'),dpi=300)
            plt.show()

        else:
            ax=self.fig.add_subplot(self.gs[self.rowc,:2])
            ax.plot(np.arange(self.signal.shape[0]),self.signal)
            ax.set_ylim(self.signal.min(),self.signal.max())
            ax.set_xticks(np.arange(0,self.signal.shape[0],5000))
            ax.set_xlabel("n(samples)")
            ax.set_ylabel('ecg')
            ax.spines['right'].set_visible(False)
            ax.spines['top'].set_visible(False)
            ax.set_title('Full signal plot')
            self.rowc+=1
            self.colc=0

    def plot_sliced_signal(self,in_axis=False):
        if not in_axis:
```

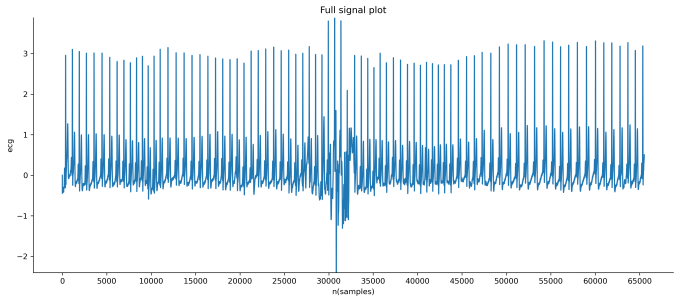
```
        plt.figure(figsize=(10,3))
        plt.plot(np.arange(self.sliced_signal.shape[0]),self.sliced_signal)
        plt.ylim(self.sliced_signal.min()-0.5,self.sliced_signal.max()+0.5)
        plt.xticks(np.arange(0,self.sliced_signal.shape[0],100))
        plt.xlabel("n(samples)")
        plt.ylabel('ecg')
        ax=plt.gca()
        ax.spines['top'].set_visible(False)
        ax.spines['right'].set_visible(False)
        plt.title('Sliced signal(100:1000) plot')
        plt.savefig(os.path.join('','figures','ecg_sliced_signal_100_1000.png'),dpi=300)
        plt.show()
    else:
        ax=self.fig.add_subplot(self.gs[self.rowc,self.colc])
        ax.plot(np.arange(self.sliced_signal.shape[0]),self.sliced_signal)
        ax.set_ylim(self.sliced_signal.min()-0.5,self.sliced_signal.max()+0.5)
        ax.set_xticks(np.arange(0,self.sliced_signal.shape[0],100))
        ax.set_xlabel("n(samples)")
        ax.set_ylabel('ecg')
        ax.spines['top'].set_visible(False)
        ax.spines['right'].set_visible(False)
        ax.set_title('Sliced signal(100:1000) plot')
        self.update_column_counter()


def amplitude_range(self,in_axis=False):
    # Fasma megethous
    windowed_sliced_signal=np.hanning(len(self.sliced_signal))*self.sliced_signal
    fft_signal=np.fft.fft(windowed_sliced_signal)
    self.magnitude_spectrum=2.0/len(self.sliced_signal)*np.abs(fft_signal)

    sampling_rate=1000 #Hz
    self.frequency_axis=np.linspace(0,sampling_rate,len(self.magnitude_spectrum))
    if not in_axis:
        plt.figure(figsize=(10,6))
        plt.plot(self.frequency_axis,self.magnitude_spectrum)
        plt.xlabel('Frequency (Hz)')
        plt.ylabel('Ampitude')

        ax=plt.gca()
        ax.spines['right'].set_visible(False)
        ax.spines['top'].set_visible(False)
        plt.title('Magnitude spectrum signal plot')
        plt.savefig(os.path.join('','figures','magnitude_spectrum_fft_signal.png'),dpi=300)
        plt.show()
    else:
        ax=self.fig.add_subplot(self.gs[self.rowc,self.colc])
        ax.plot(self.frequency_axis,self.magnitude_spectrum)
        ax.set_xlabel('Frequency (Hz)')
        ax.set_ylabel('Ampitude')

        ax.spines['right'].set_visible(False)
        ax.spines['top'].set_visible(False)
        ax.set_title('Magnitude spectrum signal plot')
        self.update_column_counter()


def calculate_snr(self,signal,noise,cutoff_freq):
    signal_spectrum=2.0/len(signal)*np.abs(np.fft.fft(signal))
    noise_spectrum=2.0/len(noise)*np.abs(np.fft.fft(noise))

    freqs=np.fft.fftfreq(len(signal),d=1/self.frequency)
    freq_mask=freqs<=cutoff_freq
    signal_power=np.sum(signal_spectrum[freq_mask]**2)
    noise_power=np.sum(noise_spectrum[freq_mask]**2)

    return signal_power/noise_power

def cutoff_frequency(self,in_axis=False,fixed_frequency=None):
    noise=np.random.normal(0,0.5,self.sliced_signal.shape[0])
```

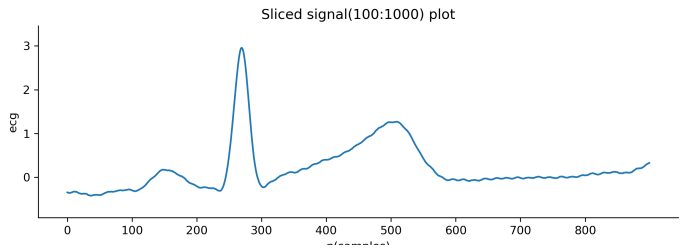
```
    if fixed_frequency:
        self.cutoff_freq=fixed_frequency
    else:
        cutoff_freqs=np.linspace(0,self.frequency,num=100)
        snrs=[self.calculate_snr(signal=self.sliced_signal,noise=noise,cutoff_freq=cutoff_freq) for
            cutoff_freq in cutoff_freqs]

        best_cutoff_idx=np.argmax(snrs)
        self.cutoff_freq=cutoff_freqs[best_cutoff_idx]

        max_indeces=np.array(snrs).argsort()[-2:]


    if not in_axis:
        plt.figure(figsize=(10,5))
        plt.plot(self.frequency_axis,self.magnitude_spectrum)
        plt.axvline(x=self.cutoff_freq, color='red', linestyle='--', label=f'Cutoff Frequency = {self
            .cutoff_freq:.2f} Hz')
        plt.ylim(self.magnitude_spectrum.min(),self.magnitude_spectrum.max()+0.5)
        plt.xlim(0,self.frequency_axis.max()+2)
        plt.xticks(np.arange(0,self.frequency_axis.max(),100))
        ax=plt.gca()
        ax.spines['top'].set_visible(False)
        ax.spines['right'].set_visible(False)
        plt.legend()
        plt.title('Cutoff frequency plot')
        plt.savefig(os.path.join('','figures','cutoff_frequency(frequency_spectrum)_signal.png'),dpi
            =300)
        plt.show()
    else:
        ax=self.fig.add_subplot(self.gs[self.rowc,self.colc])
        ax.plot(self.frequency_axis,self.magnitude_spectrum)
        ax.axvline(x=self.cutoff_freq, color='red', linestyle='--', label=f'Cutoff Frequency = {self.
            cutoff_freq:.2f} Hz')
        ax.set_ylim(self.magnitude_spectrum.min(),self.magnitude_spectrum.max()+0.5)
        ax.set_xlim(0,self.frequency_axis.max()+2)
        ax.set_xlabel('Frequency (Hz)')
        ax.set_ylabel('Amplitude')
        ax.set_xticks(np.arange(0,self.frequency_axis.max(),100))
        ax.spines['top'].set_visible(False)
        ax.spines['right'].set_visible(False)
        ax.legend()
        ax.set_title('Cutoff frequency plot')
        self.update_column_counter()

def zero_high_freq_components(self,in_axis=False):
    fft_signal=np.fft.fft(self.sliced_signal)
    indices = np.where(np.abs(np.fft.fftfreq(self.sliced_signal.size, d=1/len(self.sliced_signal))) >
        self.cutoff_freq)[0]

    fft_signal[indices]=0
    fft_signal[-indices]=0

    filtered_signal=np.fft.ifft(fft_signal).real

    if not in_axis:
        plt.figure(figsize=(10,5))
        plt.plot(2.0/len(filtered_signal)*np.abs(np.fft.fft(filtered_signal)),color='r')
        plt.xlabel('Frequency (Hz)')
        plt.ylabel('Amplitude')
        plt.title('Filtered Spectrum')
        ax=plt.gca()
        ax.spines['top'].set_visible(False)
        ax.spines['right'].set_visible(False)
        plt.savefig(os.path.join('','figures','zero_high_freq_components.png'),dpi=300)
        plt.show()
    else:
        ax=self.fig.add_subplot(self.gs[self.rowc,self.colc])
        ax.plot(2.0/len(filtered_signal)*np.abs(np.fft.fft(filtered_signal)),color='r')
        ax.set_xlabel('Frequency (Hz)')
        ax.set_ylabel('Amplitude')
```

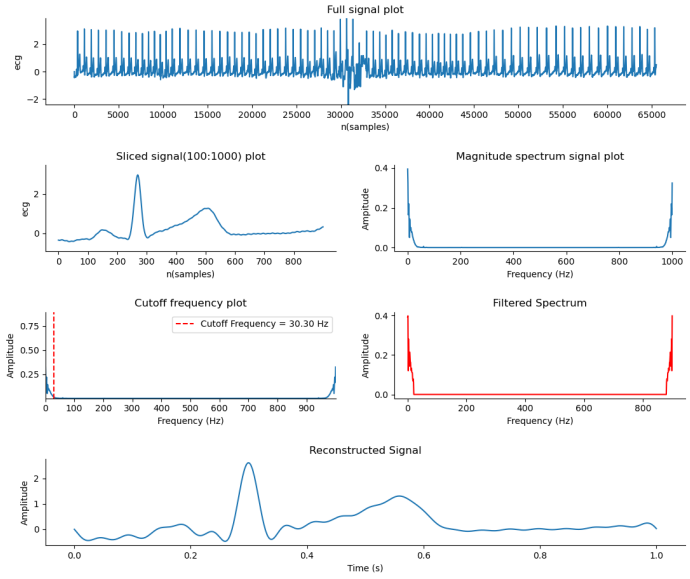
```
            ax.set_title('Filtered Spectrum')
            ax.spines['top'].set_visible(False)
            ax.spines['right'].set_visible(False)
            self.update_column_counter()


    def signal_reconstruction(self,in_axis=False):
        ecg_fft=np.fft.fft(self.sliced_signal)
        indeces=np.where(np.abs(np.fft.fftfreq(self.sliced_signal.size, d=1/len(self.sliced_signal))) >
             self.cutoff_freq)[0]

        ecg_fft[indeces]=0
        ecg_fft[-indeces]=0
        filtered_signal=np.fft.ifft(ecg_fft).real

        if not in_axis:
            plt.figure(figsize=(10,5))
            plt.plot(np.linspace(0,1,len(self.sliced_signal)),filtered_signal)
            plt.xlabel('Time (s)')
            plt.ylabel('Amplitude')
            ax=plt.gca()
            ax.spines['top'].set_visible(False)
            ax.spines['right'].set_visible(False)
            plt.title('Reconstructed Signal')
            plt.savefig(os.path.join('','figures','signal_reconstruction.png'),dpi=300)
            plt.show()
        else:
            ax=self.fig.add_subplot(self.gs[self.rowc,:2])
            ax.plot(np.linspace(0,1,len(self.sliced_signal)),filtered_signal)
            ax.set_xlabel('Time (s)')
            ax.set_ylabel('Amplitude')
            ax.spines['top'].set_visible(False)
            ax.spines['right'].set_visible(False)
            ax.set_title('Reconstructed Signal')
            self.rowc+=1

    def plot(self):
        self.set_filepath(os.path.join('','figures','Ecg_full_analysis.png'))
        self.fig.subplots_adjust(bottom=0.05,hspace=0.7)
        self.plot_signal(in_axis=True)
        self.plot_sliced_signal(in_axis=True)
        self.amplitude_range(in_axis=True)
        self.cutoff_frequency(in_axis=True)
        self.zero_high_freq_components(in_axis=True)
        self.signal_reconstruction(in_axis=True)
        self.fig.savefig(self.filename)
        plt.show()

    def distinct_plots(self):
        self.plot_signal(in_axis=False)
        self.plot_sliced_signal(in_axis=False)
        self.amplitude_range(in_axis=False)
        self.cutoff_frequency(in_axis=False)
        self.zero_high_freq_components(in_axis=False)
        self.signal_reconstruction(in_axis=False)

    def fixed_cutoff_frequencies(self,freq=None):
        self.set_filepath(os.path.join('','figures',f'Ecg_full_analysis_with_fix_freq_{freq}.png'))
        self.fig.subplots_adjust(bottom=0.05,hspace=0.7)
        self.plot_signal(in_axis=True)
        self.plot_sliced_signal(in_axis=True)
        self.amplitude_range(in_axis=True)
        self.cutoff_frequency(in_axis=True,fixed_frequency=freq)
        self.zero_high_freq_components(in_axis=True)
        self.signal_reconstruction(in_axis=True)
        self.fig.savefig(self.filename)
        plt.show()

import argparse
if __name__=='__main__':
    parser=argparse.ArgumentParser()
```

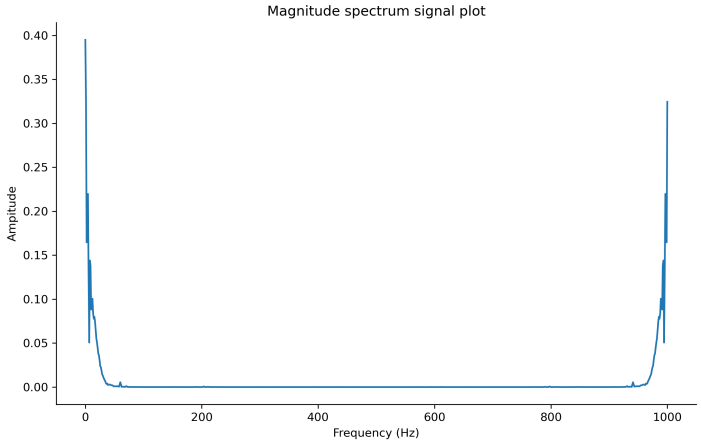
```
parser.add_argument('--s1',action='store_true',help='Select scenario 1 for execution')
parser.add_argument('--s2',action='store_true',help='Select scenario 2 for execution(Each plot will
    created distinctively)')
parser.add_argument('--s3',action='store_true',help='Select scenario 3 for using fixed frequency
    execution(Each plot will created distinctively)')
parser.add_argument('--s',action='store_true',help='Print dataset statistics')
parser.add_argument('--f',help='Fixed cutoff frequency')
args=parser.parse_args()

ecg=ECG()
if args.s:
    ecg.statistics()


if args.s1:
    ecg.plot()
elif args.s2:
    ecg.distinct_plots()
elif args.s3:
    if args.f:
        ecg.fixed_cutoff_frequencies(int(args.f))
else:
    raise ValueError("Scenario does not been implemented yet select s1 or s2")
```

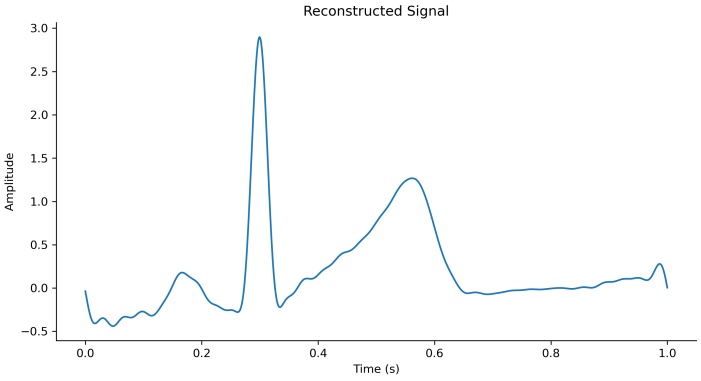
3 Results

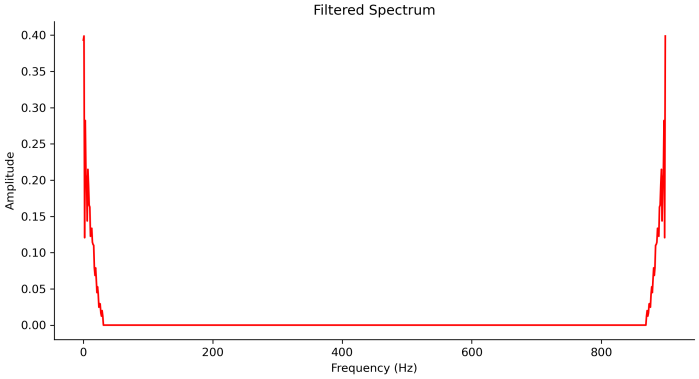


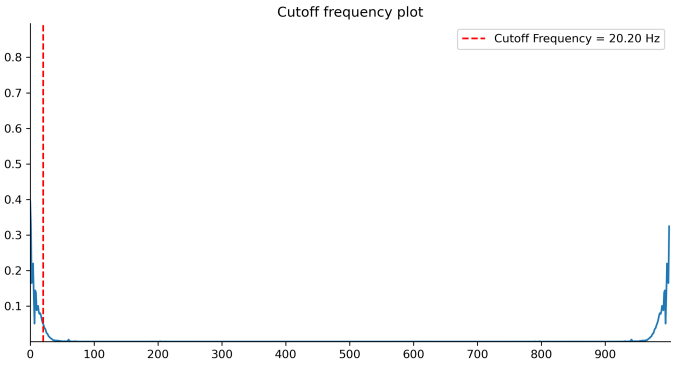


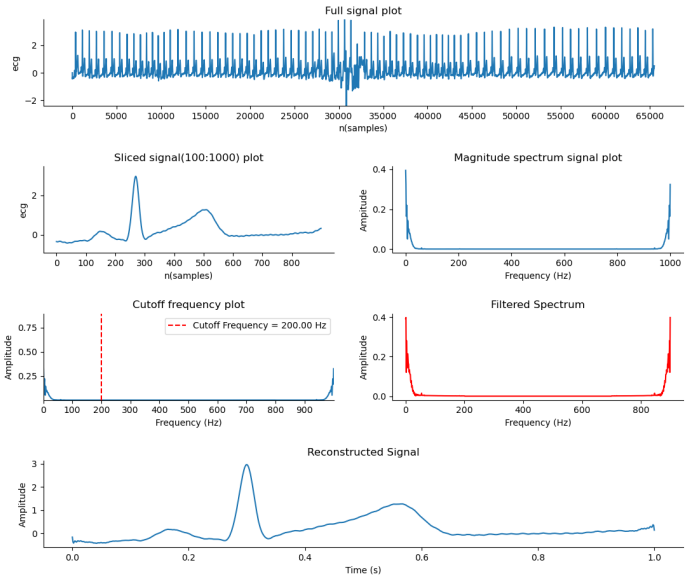

Full signal plot
ecg
n(samples)
Sliced signal(100:1000) plot
ecg
n(samples)
Magnitude spectrum signal plot
Amplitude
Frequency (Hz)
Cutoff frequency plot
Cutoff Frequency = 30.30 Hz
Amplitude
Frequency (Hz)
Filtered Spectrum
Amplitude
Frequency (Hz)
Reconstructed Signal
Amplitude
Time (s)

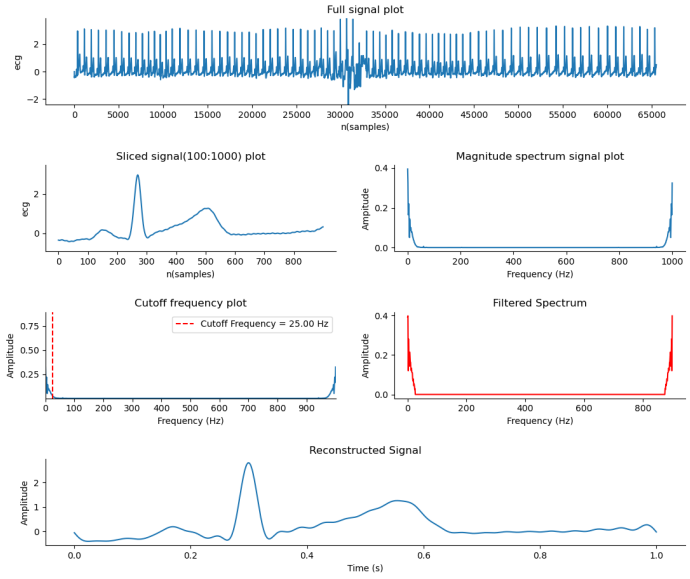

Magnitude spectrum signal plot
Amplitude
Frequency (Hz)


Reconstructed Signal
Amplitude
Time (s)


Filtered Spectrum
Amplitude
Frequency (Hz)


Cutoff frequency plot
Cutoff Frequency = 20.20 Hz


Full signal plot
ecg
n(samples)
Sliced signal(100:1000) plot
Magnitude spectrum signal plot
Amplitude
Frequency (Hz)
Cutoff frequency plot
Cutoff Frequency = 200.00 Hz
Filtered Spectrum
Reconstructed Signal
Time (s)


Full signal plot
ecg
n(samples)
Sliced signal(100:1000) plot
ecg
n(samples)
Magnitude spectrum signal plot
Amplitude
Frequency (Hz)
Cutoff frequency plot
Cutoff Frequency = 25.00 Hz
Amplitude
Frequency (Hz)
Filtered Spectrum
Amplitude
Frequency (Hz)
Reconstructed Signal
Amplitude
Time (s)