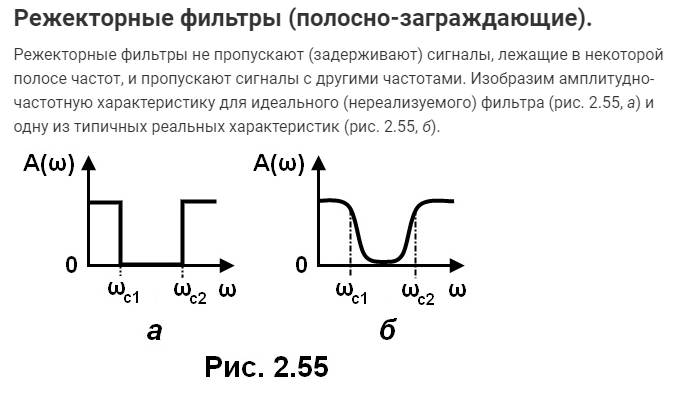
2. Фильтрация (выбрать один из 4 основных типов фильтров). Получить фильтрованный  
сигнал.

режекторный фильтры (полосно-заграждающие).



def ex\_3\_2():

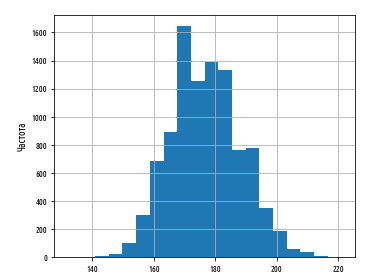
    df = load\_data()

    df['Рост, см'].hist(bins=20)

    plt.xlabel('Рост, см.')

    plt.ylabel('Частота')

    plt.show()



def ex\_3\_3():

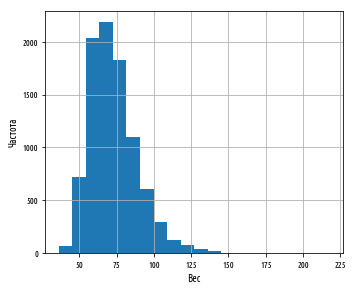
    df = load\_data()

    df['Вес'].hist(bins=20)

    plt.xlabel('Вес')

    plt.ylabel('Частота')

    plt.show()



def ex\_3\_5():

    '''Визуализация разброса значений веса спортсменов на

       полулогарифмической гистограмме с целью удаления

       асимметрии'''

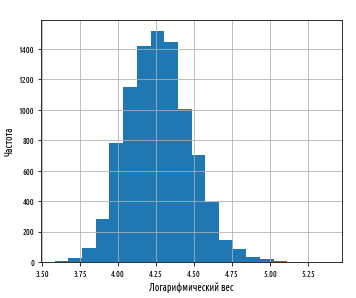
    df = load\_data()

    df['Вес'].apply(np.log).hist(bins=20)

    plt.xlabel('Логарифмический вес')

    plt.ylabel('Частота')

    plt.show()



3. Эффект ”Эхо”. Получить сигнал после работы аудио-эффекта.

% sampling frequency [Hz]

Fs = 1000;

% sampling period [s]

Ts = 1/Fs;

% time vector [s]

t = 0:Ts:3;

% Signal A frequency [Hz] and amplitude

amplitude\_A = 5;

f\_A = 3;

sig\_A = amplitude\_A.\*sin(2\*pi\*f\_A.\*t);

% Signal A frequency [Hz] and amplitude

amplitude\_B = 2;

f\_B = 25;

sig\_B = amplitude\_B.\*sin(2\*pi\*f\_B.\*t);

% sum of the two signals

X = sig\_A + sig\_B;

plot(t,X,'LineWidth',2);

hold on;

grid on;

% Low pass filter at ~15Hz

% normalized frequency wn

wn = 15/Fs;

b\_1 = fir1(45,wn);

b\_2 = fir1(70,wn);

% filtered signal

a = 1; % fir filter does not have poles (transfer function denominator = 1)

Y\_1 = filtfilt(b\_1,a,X);

plot(t,Y\_1,'-r','LineWidth',2);

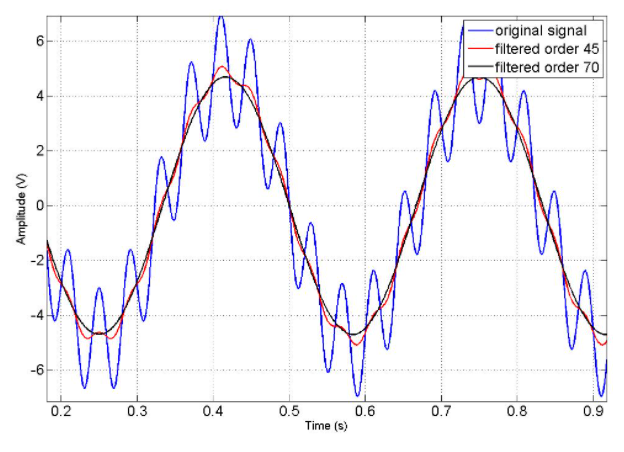
Y\_2 = filtfilt(b\_2,a,X);

plot(t,Y\_2,'-k','LineWidth',2);

xlabel('Time (s)');

ylabel('Amplitude (V)');

legend('original signal','filtered order 45',...'filtered order 70');



from \_\_future\_\_ import division

import numpy as np

import matplotlib.pyplot as plt

data = np.random.rand(301) - 0.5

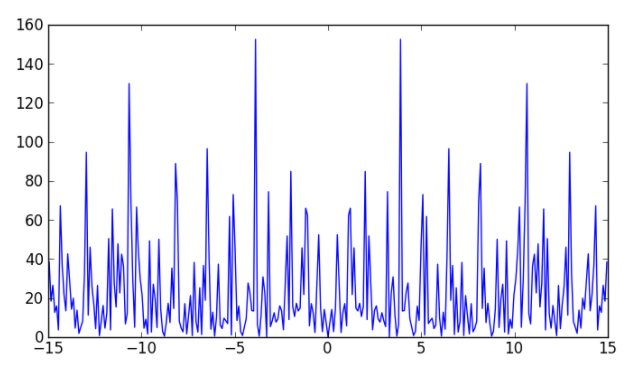
ps = np.abs(np.fft.fft(data))\*\*2

time\_step = 1 / 30

freqs = np.fft.fftfreq(data.size, time\_step)

idx = np.argsort(freqs)

plt.plot(freqs[idx], ps[idx])



import numpy as np

import pylab as pl

rate = 30.0

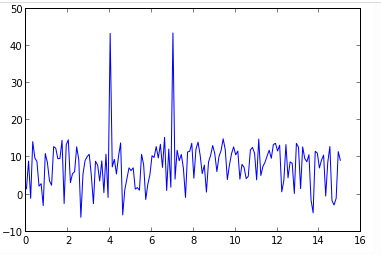
t = np.arange(0, 10, 1/rate)

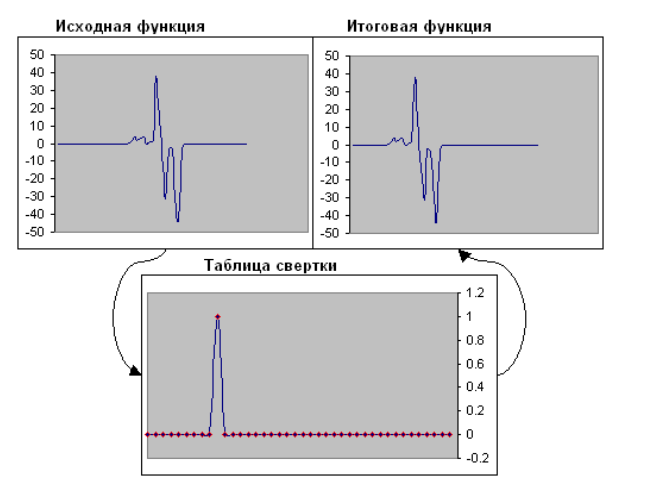
x = np.sin(2\*np.pi\*4\*t) + np.sin(2\*np.pi\*7\*t) + np.random.randn(len(t))\*0.2

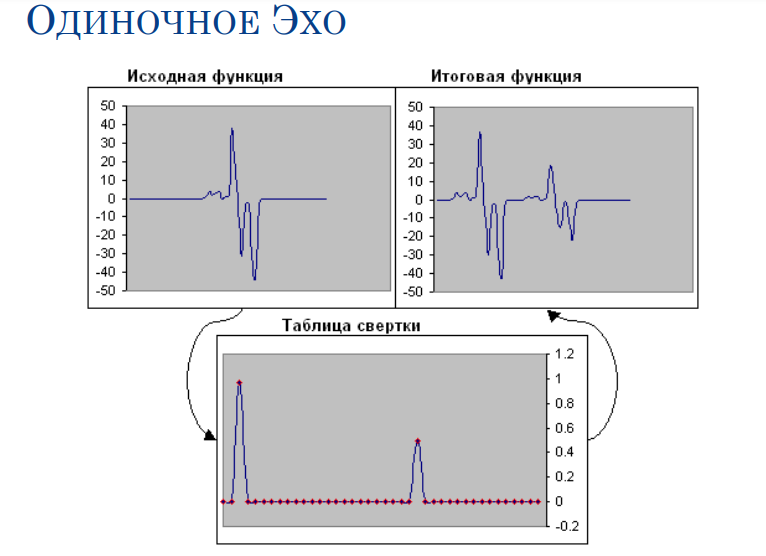
p = 20\*np.log10(np.abs(np.fft.rfft(x)))

f = np.linspace(0, rate/2, len(p))

plot(f, p)









4. Эффект ”Distortion”(классический вариант). Получить сигнал после работы  
аудио-эффекта.

**def** **signal\_chirp**(amp=1.0, freq=0.0, beta=0.25, period=100, \*\*kwargs):

"""

Create Chirp signal

Parameters

----------

amp : float

Signal magnitude

beta : float

Modulation bandwidth: beta < N for complex, beta < 0.5N for real

freq : float or int

Linear frequency of signal

period : integer

Number of points for signal (same as period)

kwargs : bool

Complex signal if is\_complex = True

Modulated by half-sine wave if is\_modsine = True

"""

is\_complex = kwargs.get('is\_complex', False)

is\_modsine = kwargs.get('is\_modsine', False)

t = np.linspace(0, 1, period)

tt = np.pi \* (freq \* t + beta \* t \*\* 2)

**if** is\_complex **is** True:

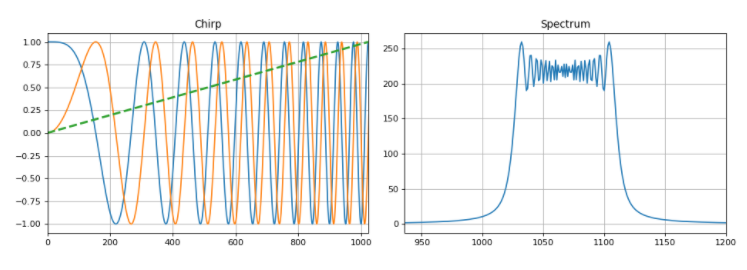
res = amp \* (np.cos(tt) + 1j \* np.sin(tt))

**else**:

res = amp \* np.cos(tt)

**if** is\_modsine **is** True:

**return** res \* np.sin(np.pi \* t)

**return** res 

from scipy.fft import irfft

new\_sig = irfft(yf)

plt.plot(new\_sig[:1000])

plt.show()

