Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Телекоммуникационные технологии

Отчет по лабораторной работе №3
Линейная фильтрация

Работу выполнил:
Чугунов А.А.
Группа: 33501/4
Преподаватель:
Богач Н.В.

Санкт-Петербург
2017

1. Цель работы

Изучить воздействие ФНЧ на тестовый сигнал с шумом.

2. Теоретическая информация

Линейный фильтр — динамическая система, применяющая некий линейный оператор ко входному сигналу для выделения или подавления определённых частот сигнала. Суть фильтрации заключается в пропускании сигнала через линейную цепочку с какими-то заданными параметрами для получения нужного нам конечного сигнала. Пускай имеем два сигнала - начальный и конечный, которые имеею следующий вид:

$$x(t) = A_x e^{j(2\pi ft+\phi_x)} \tag{1}$$

$$y(t) = A_y e^{j(2\pi ft+\phi_y)} \tag{2}$$

Поделив конечный сигнал(2) на начальный(1), получаем частотную характеристику:

$$G(f) = \frac{A_y}{A_x} e^{j(\phi_y-\phi_x)} \tag{3}$$

Спектр выходного сигнала можно посчитать по следующей формуле:

$$Y(f) = X(f)G(f) \tag{4}$$

Таким образом, варьируя частотную характеристку линейной цепи можем получать на выходе различные сигналы. Если же работать во временной области пренебрегая преобразованиями Фурье, можем пропустить через линейную цепль функцию Дирака. Спектр его равне единице. Тогда получаем следующее выражение:

$$Y(f) = G(f) \tag{5}$$

Реакция цепи g(t) на функцию Дирака носит назавние импульсной характеристики. Импульсная характеристика связана с частотной следующими соотношениями:

$$g(t) = \int_{-\infty}^{\infty} G(f)e^{j(2\pi ft)}df \tag{6}$$

$$G(t) = \int_{0}^{\infty} g(t)e^{-j(2\pi ft)}dt \tag{7}$$

Таким образом, во временной области получаем следующую формулу:

$$y(t) = x(t) * g(t) \tag{8}$$

В данной работе будем пользоваться тремя фильтрами: медианным фильтром, фильтром скользящего среднего и фильтром Баттерворта. Медианный фильтр и фильтр скользящего среднего являются простой интерполяцией. Работая с обоими фильтами неоходимо задаться некоторым окном значений. В случае фильтра скользяцего среднего, необходимо на каждом шаге из окна выбирать среднее арифметическое значений окна. В случае медианного фильтра, необходимо сортировать значения окна и выбирать его медиану, то есть значение, стоящее посередине.

Работая с фильтром Баттерворта, необходимо знать его частотную характеристку. Её можно получить из следующей формулы:

$$G^2(\omega) = \frac{G_0^2}{1 + (\frac{\omega}{\omega_c})^{2n}} \tag{9}$$

Здесь, n - порядок фильтра, Wc - частота среда, Go - коэффициент усиления по постоянной составляющей.

3. Ход выполнения работы

Для начала сгенерируем гармонический сигнал. В данной работе будет использовать синус.(Рисунок. 3.1)

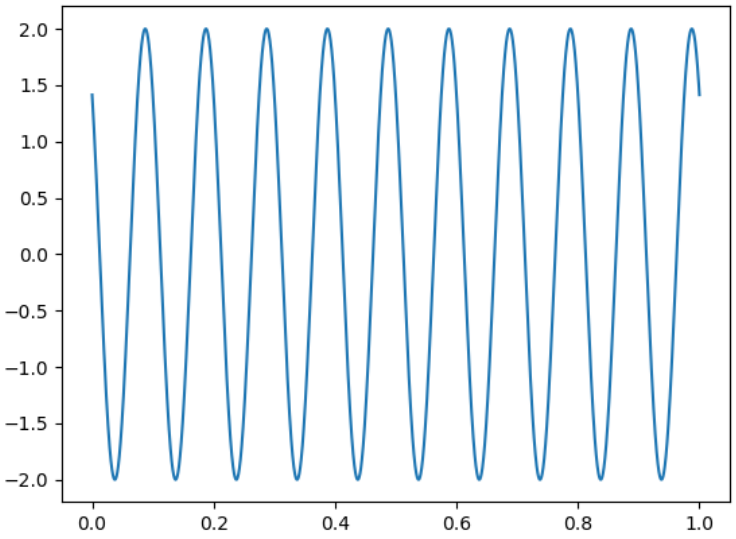

Рисунок 3.1. Гармонический сигнал

Теперь добавим шума в сигнал. Шумом будет служить синус, но более высокой частоты и более низкой амплитуды.(Рисунок. 3.2)

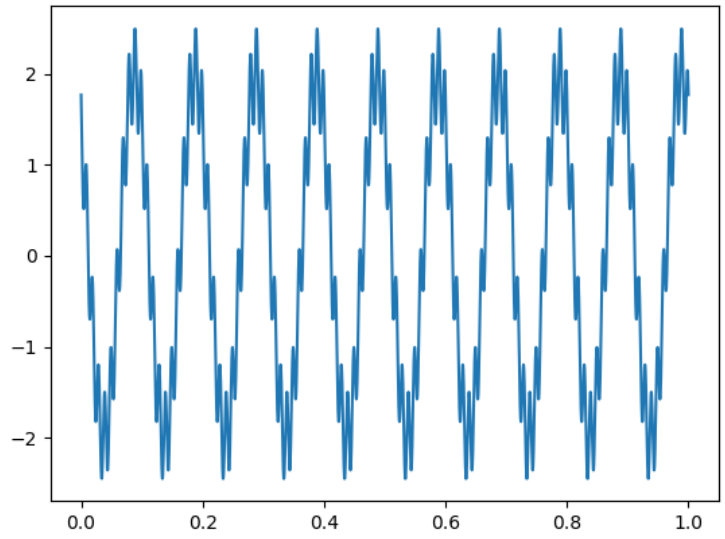

Рисунок 3.2. Зашумленный гармонический сигнал

Рассмотрим спектры обоих сигналов.(Рисунок. 3.3 и Рисунок. 3.4)

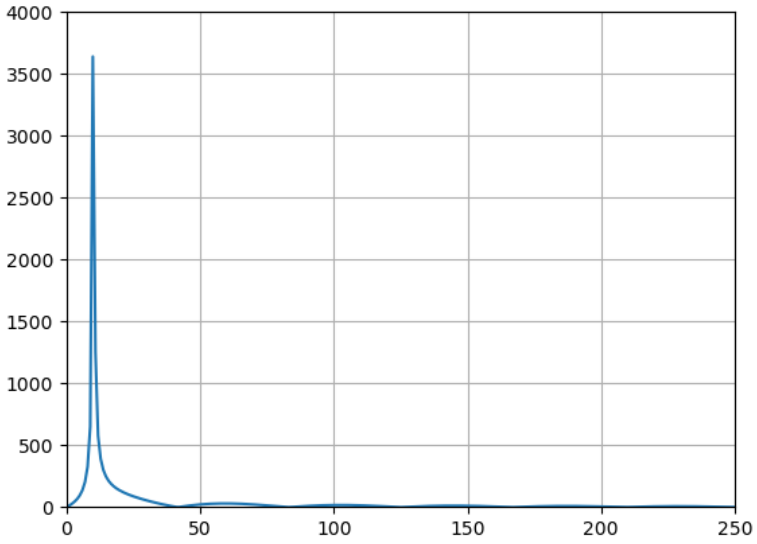

Рисунок 3.3. Спектр гармонического сигнала

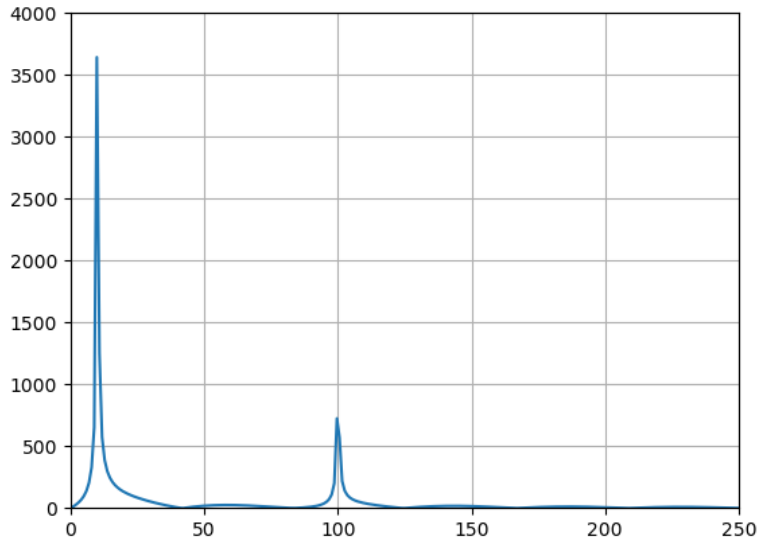

Рисунок 3.4. Спектр зашумленного гармонического сигнала

На спектре зашумелнного гармонического сигнала наблюдаем паразитую частоту, которая является частотой добавленных шумов. Теперь воспользуемся фильтром скользящего среднего. Здесь, окно - 50 дискретных значений.(Рисунок. 3.5)

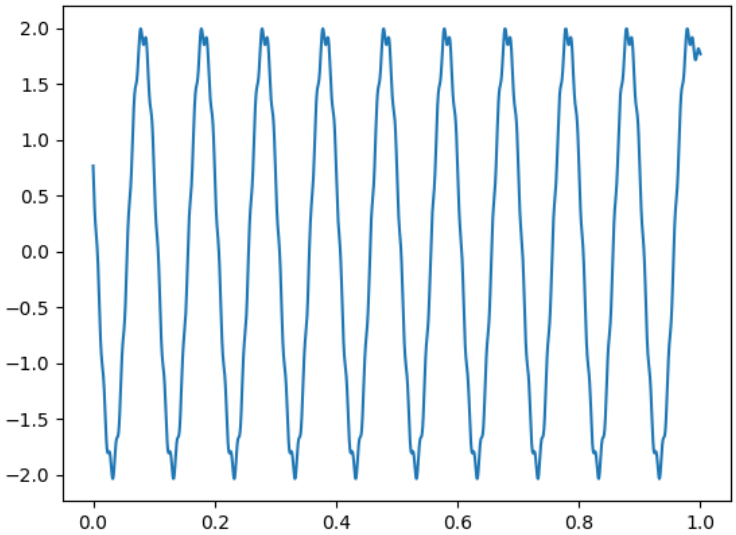

Рисунок 3.5. Сигнал после фильтрации фильтром скользящего среднего

Получили неплохие результаты, однако на практике окно в 50 значений недопустимо, что говорит нам о том, что для нашей задачи данный фильтр не подходит. Рассмотрим спектр сигнала после фильтрации.(Рисунок. 3.6)

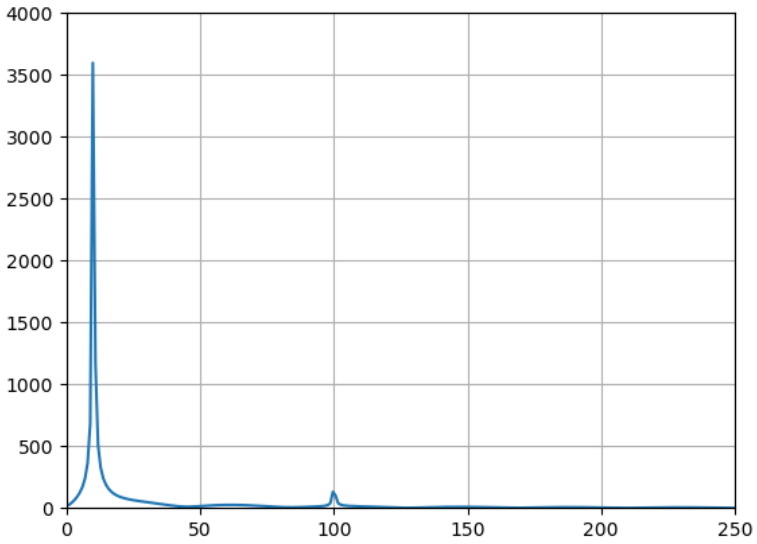

Рисунок 3.6. Спектр после фильтрации фильтром скользящего среднего

Однако в частотной области паразитная частота практически исчезла. Теперь воспользуемся медианным фильтром. Здесь окно - 85 значений, что тоже недопустимо на практике. Результат фильтрации(Рисунок. 3.7) и спектр(Рисунок. 3.8).

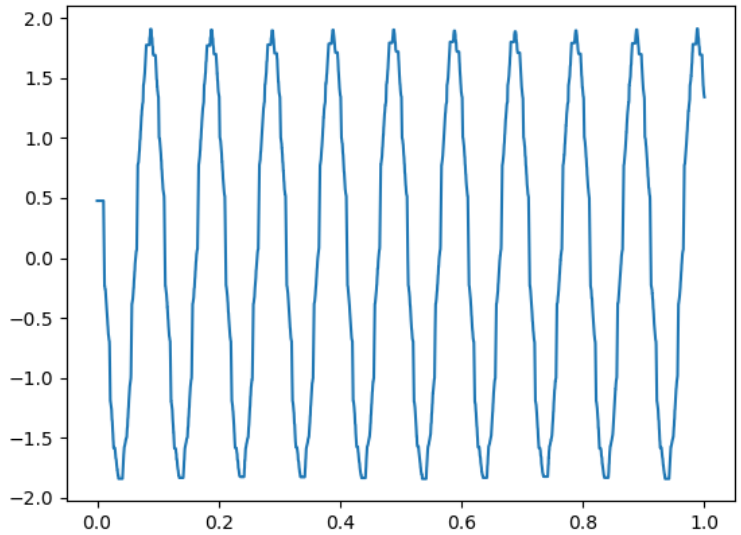

Рисунок 3.7. Сигнал после фильтрации медианным фильтром

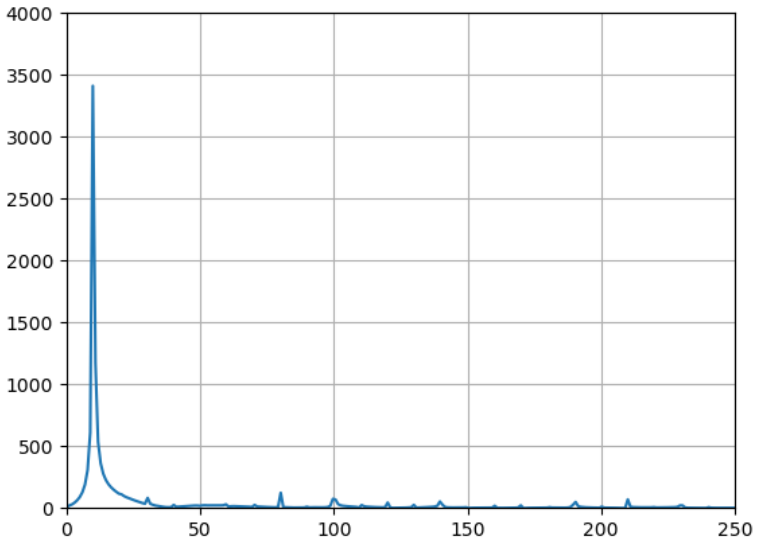

Рисунок 3.8. Спектр после фильтрации медианным фильтром

Опять получили не самые плохие результаты, более того, в частотной области паразитная частота была удалена, однако на практите медианный фильтр применять не будем для решения такого рода задач.

Переходим к последнему фильтру - фильтру Баттерворта. Задавшись частотой и порядком фильтра получили следующий сигнал(Рисунок. 3.9) и спектр(Рисунок. 3.10).

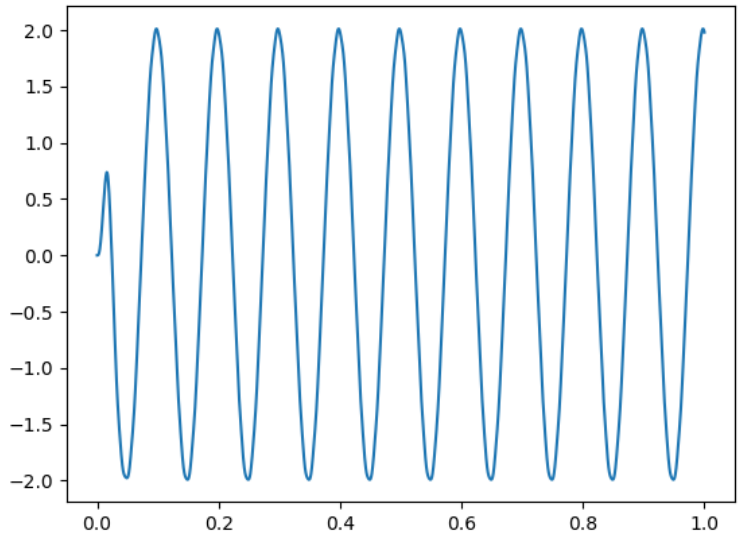

Рисунок 3.9. Сигнал после фильтрации фильтром Баттерворта

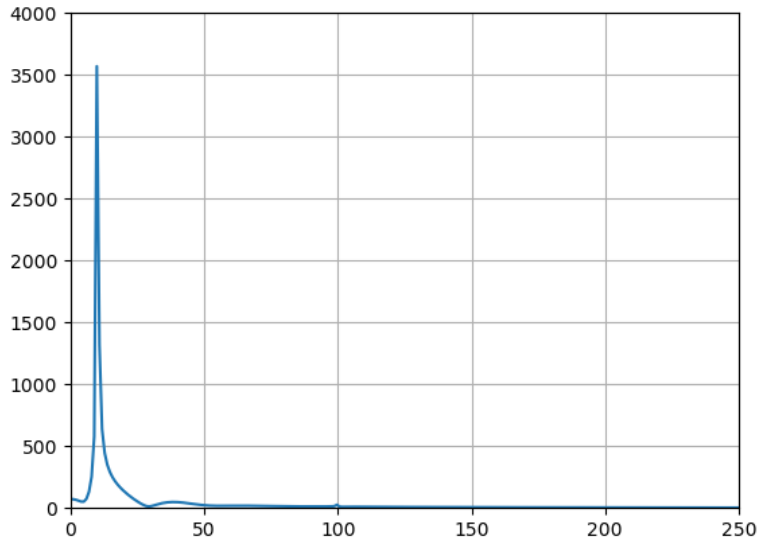

Рисунок 3.10. Спектр после фильтрации фильтром Баттерворта

Получили отличный результаты и в частотной и во временной области. Фильтр справился со своей задачей и отфильтровал шумы из нашего сигнала. Код программы представлен ниже. Фильтр скользящего среднего пришлось реализовывать самостоятельно в виду его отсутствия в библиотеках языка Python.

Листинг 1: LinearFiltration.py

```
1  from scipy.fftpack import fft
2  import numpy as np
3  import time as time
4  import random
5  import matplotlib.pyplot as plt
6  from scipy import signal
7
8
9  Fdiscrete = 4e3
10 t = np.linspace(0, 1, int(Fdiscrete))
11 A = 2
12 f0 = 10
13 phi = np.pi / 4
14 sig = A * np.cos(2 * np.pi * f0 * t + phi)
15 noiseSig = A / 4 * np.cos(20 * np.pi * f0 * t + phi) + sig
16 sig = np.asarray(sig)
17 Nfft = int(2 ** np.ceil(np.log2(len(sig))))
18 sp = fft(sig, Nfft)
19 sp_dB = 20 * np.log10(np.abs(sp))
20 f = np.arange(0, Nfft - 1) / Nfft * Fdiscrete
21 plt.figure()
22 plt.grid()
23 plt.plot(f[:int(Nfft / 2)], np.abs(sp[:int(Nfft / 2)]))
24 plt.axis([0, 250, 0, 4000])
25
26 Nfft = int(2 ** np.ceil(np.log2(len(noiseSig))))
27 sp = fft(noiseSig, Nfft)
28 sp_dB = 20 * np.log10(np.abs(sp))
29 f = np.arange(0, Nfft - 1) / Nfft * Fdiscrete
```

```
30 plt.figure()
31 plt.grid()
32 plt.plot(f[:int(Nfft / 2)], np.abs(sp[:int(Nfft / 2)]))
33 plt.axis([0, 250, 0, 4000])
34
35 plt.figure()
36 plt.plot(t, sig)
37 plt.figure()
38 plt.plot(t, noiseSig)
39
40 filteredSig = np.asarray(noiseSig)
41 window = np.arange(50)
42 for si in (np.arange(len(noiseSig) - len(window))):
43     sum = 0
44     for i in window:
45         sum = sum + noiseSig[si + i]
46     filteredSig[si] = sum / len(window)
47 lastPart = np.arange(len(noiseSig) - len(window), len(noiseSig))
48 for si in lastPart:
49     sum = 0
50     for i in np.arange((len(noiseSig) - si)):
51         sum = sum + noiseSig[si + i]
52     for iti in np.arange(len(window) - (len(noiseSig) - si)):
53         sum = sum + noiseSig[len(noiseSig) - 1]
54     filteredSig[si] = sum / len(window)
55 plt.figure()
56 plt.plot(t, filteredSig)
57
58 Nfft = int(2 ** np.ceil(np.log2(len(filteredSig))))
59 sp = fft(filteredSig, Nfft)
60 sp_dB = 20 * np.log10(np.abs(sp))
61 f = np.arange(0, Nfft - 1) / Nfft * Fdiscrete
62 plt.figure()
63 plt.grid()
64 plt.plot(f[:int(Nfft / 2)], np.abs(sp[:int(Nfft / 2)]))
65 plt.axis([0, 250, 0, 4000])
66
67 noiseSignal = A / 4 * np.cos(20 * np.pi * f0 * t + phi) + sig
68 pureSig = signal.medfilt(noiseSignal, 85)
69 plt.figure()
70 plt.plot(t, pureSig)
71
72 Nfft = int(2 ** np.ceil(np.log2(len(pureSig))))
73 sp = fft(pureSig, Nfft)
74 sp_dB = 20 * np.log10(np.abs(sp))
75 f = np.arange(0, Nfft - 1) / Nfft * Fdiscrete
76 plt.figure()
77 plt.grid()
78 plt.plot(f[:int(Nfft / 2)], np.abs(sp[:int(Nfft / 2)]))
79 plt.axis([0, 250, 0, 4000])
80
81 cutoff = 0.1
82 noiseSig =  A / 4 * np.cos(20 * np.pi * f0 * t + phi) + sig
83 b, a = signal.butter(4, cutoff/(f0/2), btype='low', analog=False, output='ba')
84 y = signal.lfilter(b, a, noiseSig)
85 # Get the filter coefficients so we can check its frequency response.
86 plt.figure()
87 plt.plot(t, y)
88
89 Nfft = int(2 ** np.ceil(np.log2(len(y))))
```

```
sp = fft(y, Nfft)
sp_dB = 20 * np.log10(np.abs(sp))
f = np.arange(0, Nfft - 1) / Nfft * Fdiscrete
plt.figure()
plt.grid()
plt.plot(f[:int(Nfft / 2)], np.abs(sp[:int(Nfft / 2)]))
plt.axis([0, 250, 0, 4000])

plt.show()
```

4. Выводы

Проделав лабораторную работу, рассмотрели понятие линейной фильтрации и научились пользоваться тремя фильтрами. Установили, что фильтр Баттерворта справляется с задачей фильтрации шумов лучше, чем это делают медианный фильтр и фильтр скользящего среднего. Выяснили, что отфильтровать сигнал можно при помощи грамотного подбора параметров для частотной характеристики линейной цепи. Подобрав верные параметры, получаем возможность полностью убрать зашумляющие сигналы, частоты которых выше заданной нами частосты среза. Неполное удаление шума фильтром возможно из-за пологости частотной характеристики на полосе подавления, однако, меняя порядок фильтра, этой проблемы можно избежать.