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

# Телекоммуникационные технологи

Отчет по лабораторной работе №1
Сигналы телекоммуникационных систем

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

Санкт-Петербург
2017

# Содержание

# 1. Цель работы

Познакомиться со средствами генерации и визуализации простых сигналов, с различными программными средствами цифоровой обработки сигнала, научиться ими пользоваться вследствие выполнения поставленных задач.

# 2. Постановка задачи

В командном окне MATLAB и в среде Simulink промоделировать сигналы из Главы 3, сс. 150–170 (см. Справочные материалы). В качестве альтернативного программного средства можно использовать язык python и различные сопутствующие инструменты(Jupyter Notebook). После выполнения моделирования так же требуется оформить отчет с помощью LaTeX

# 3. Теоретическая информация

В качестве языка выбираем python 3. Основные библиотеки, которые нам потребуются:

- scipy.signal - для обработки сигналов
- matplotlib - для построения графиков
- numpy - для удобной работы с матрицами и векторами

Для моделирования и выполнения кода будем использовать Jupyter Notebook. Приведем некоторые формулы, которые будут использоваться в данной работе:

- гармонический сигнал:
$$s_1(t) = \cos(2\pi f_0 t + \phi) \tag{1}$$
- затухающий гармонический сигнал:
$$s_2(t) = \cos(2\pi f_0 t + \phi)e^{-\alpha t} \tag{2}$$
- Гауссов импульс:
$$y(t) = \cos(2\pi f_c t)e^{-\alpha t^2} \tag{3}$$
- Функция Дирихле:
$$\frac{\sin(nx/2)}{nsin(nx/2)}, n \in \mathbb{Z} \tag{4}$$

# 4. Ход работы

Для реализации гармонического сигнала в соответствии с формулой (**??**) нам необходимо также реализовать функцию поэлементного умножения elem_multiple по аналогии с MATLAB.

## 4.1. Введение

Листинг 1: source01.py

```
import numpy as np
import math
import matplotlib.pyplot as plt

def elem_multiple(a, b):
  if a.shape != b.shape:
    raise IOError("Shapes␣of␣matrices␣are␣not␣the␣same")
    return
  else:
    res = np.zeros(a.shape)
    for i in range(a.shape[0]):
      for j in range(a.shape[1]):
        res[(i,j)] = a[(i,j)] * b[(i,j)]
    # print(res.shape)
    return res

Fs = 8000 # discrete frequency
t = np.matrix(np.arange(0.0, 1.0, (1 / Fs))).transpose()
A = 2 # amplitude
f0 = 1000
phi =  np.pi /4
s1 = A * np.cos(2 * np.pi *  f0 * t + phi) # harmonic signal
fig = plt.figure(figsize=(6,6))
plt.title('Гармонический␣сигнал')
plt.plot(t[1:100],s1[1:100])
fig.savefig('picturesNote/001harmonic.png', dpi=200)
plt.show()
```

Выполнив данный код, получаем график сигнала:

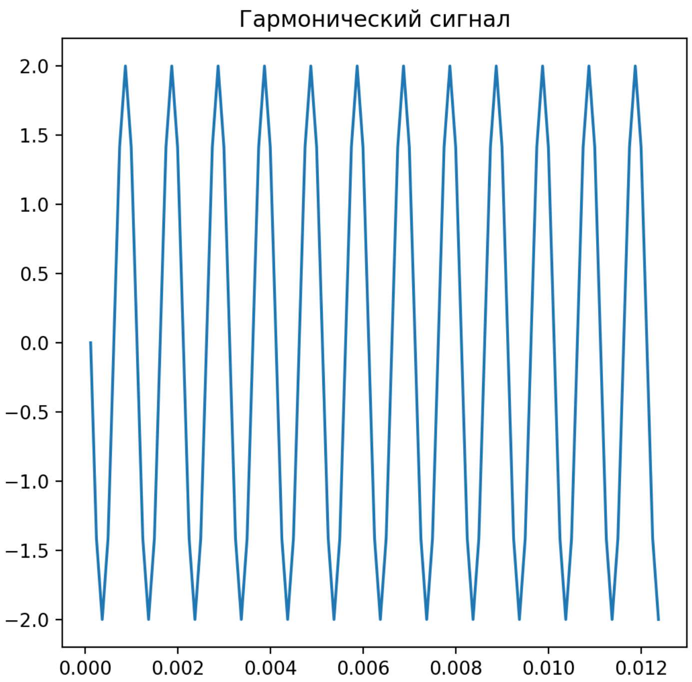


Рисунок 4.1. Гармонический сигнал

Далее реализуем уже затухающий гармонический сигнал (**??**):

Листинг 2: source02.py

```
import numpy as np
import matplotlib.pyplot as plt

def elem_multiple(a, b):
    if a.shape != b.shape:
        raise IOError("Shapes of matrices are not the same")
        return
    else:
        res = np.zeros(a.shape)
        for i in range(a.shape[0]):
            for j in range(a.shape[1]):
                res[(i,j)] = a[(i,j)] * b[(i,j)]
        # print(res.shape)
        return res

Fs = 8000 # discrete frequency
t = np.matrix(np.arange(0.0, 1.0, (1 / Fs))).transpose()
f0 = 1000
phi = np.pi /4
alpha = 1000
s2 = elem_multiple(s1, np.exp(-alpha * t))
fig = plt.figure(figsize=(6,6))
plt.title('Затухающий сигнал')
plt.plot(t[1:100],s2[1:100])
fig.savefig('picturesNote/002harmonic.png', dpi=100)
plt.show()
```

Выполнив данный код, получаем график сигнала:

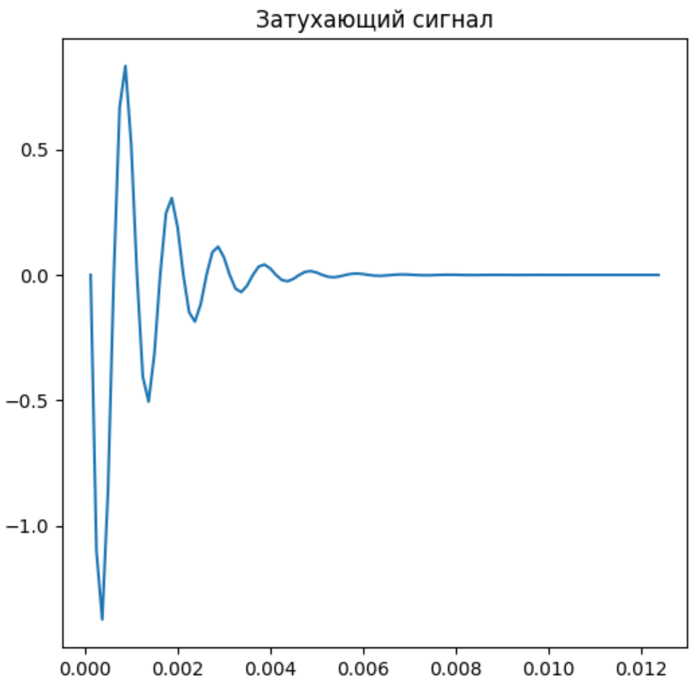


Рисунок 4.2. Затухающий гармонический сигнал

Далее промоделируем многоканальный сигнал (у каждого канала будет просто своя частота), взяв число канальности равное 5. Сигнал будет так же гармоническим: $s_3(t) = \cos(2\pi f_0 t)$, где $f_0 = (600, 800, 1000, 1200, 1400)$

Листинг 3: source03.py

```
import numpy as np
import matplotlib.pyplot as plt


Fs = 40000 # discrete frequency
t = np.matrix(np.arange(0.0, 1.0, (1 / Fs))).transpose()
f = np.matrix([600, 800, 1000, 1200, 1400])

s3 = np.matrix(np.cos(2*np.pi * t * f)) # 5 channels signal

fig = plt.figure(figsize=(8, 8))
plt.title('канальный5−␣сигнал')
plt.plot(t[1:50], s3[1:50])
fig.savefig('picturesNote/003_5chnls.png', dpi=200)
plt.show()
```

Выполнив данный код, получаем график сигнала:

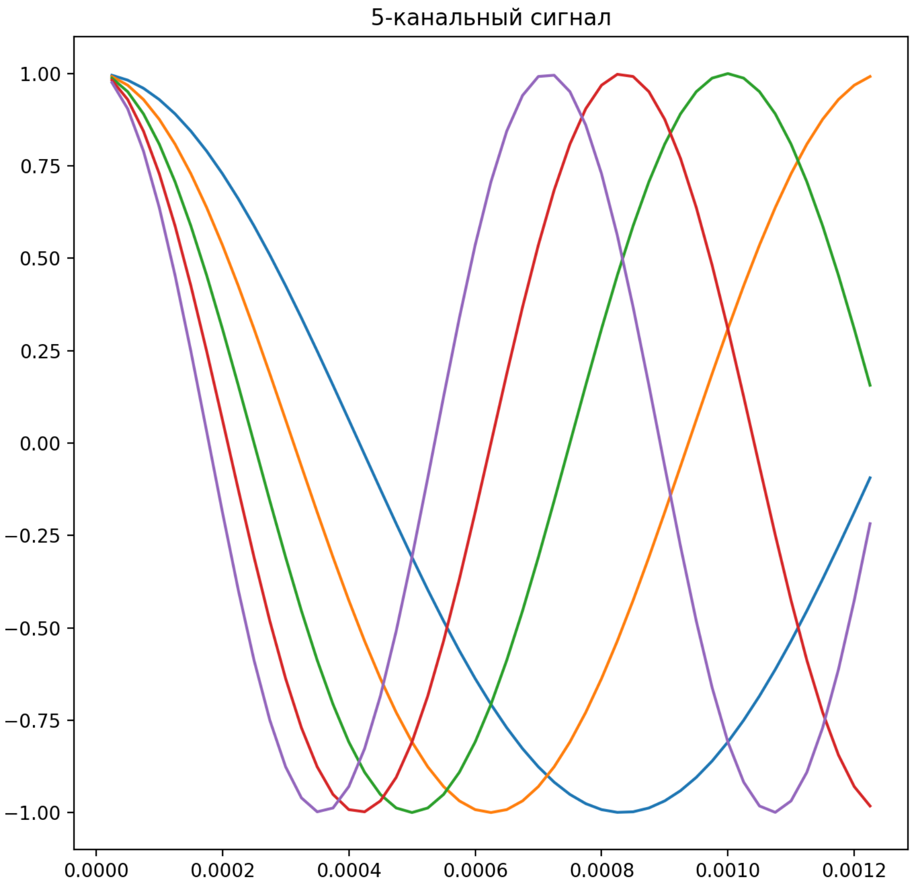


Рисунок 4.3. Многоканальный сигнал

В соотвествии с теорией получаем семейство косинусов с различными фазами - искомый многоканальный сигнал.

## 4.2. Одиночные импульсы

В библиотеке python scipy.signal в отличие от MATLAB нет функций генерации одиночных имплусов, поэтому для примера реализуем прямоуголный импульс самостоятельно.

Листинг 4: source04.py

```
import numpy as np
import matplotlib.pyplot as plt

# rectangular impulse
def rect_impls_1(t, width):
    sig = np.zeros(len(t))
    for i in sig:
        if -width/2 <= i < width / 2:
            i = 1
    for i in range(len(sig)):
        if -width/2 <= t[i] < width / 2:
```

```
            sig[i] = 1
    return sig

Fs = 1000
t = np.matrix(np.arange(-0.84, 0.84, (1 / Fs))).transpose()
width = 0.2
A = 5
s = -A *rect_impls_1(t + width / 2, width) + A * rect_impls_1(t - width / 2 , width)
plt.title('Прямоугольный импульс')
plt.plot(t,s)
plt.gcf().savefig('picturesNote/004rectImp.png', dpi=100)
plt.show()
```

Выполнив данный код, получаем график сигнала:

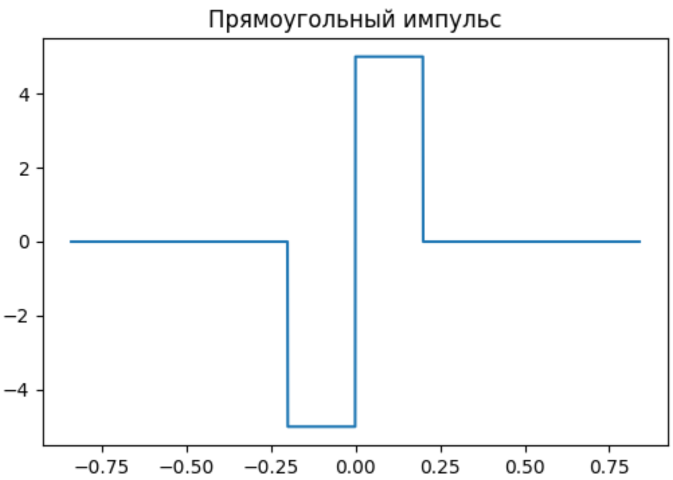


Рисунок 4.4. Прямоугольный импульс

Полученный график очевидно является прямоугольным импульсом.

Также построим импульс Гаусса, реализация которого представлена в scipy.signal. По определению Гауссов импульс представляется формулой (**??**).

Листинг 5: source05.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from scipy.fftpack import fft, fftfreq

Fs = 800
t = np.linspace(-1, 1, 2 * Fs)
i, e = signal.gausspulse(t, fc=5, retenv=True)
fig = plt.figure()
plt.plot(t, i, t, e, '--')
plt.title('Гауссов импульс')
fig.savefig('picturesNote/005_gaus.png', dpi=70)
plt.show()
N = len(t)
Fs = 800
T = 1.0 / Fs
y = i
```

```
yf = fft(y)
xf = fftfreq(N, 1.0 / Fs)
fig = plt.figure()
plt.title('Амплитудный спектр')
plt.plot(xf[1:800], np.abs(yf[1:800]))
fig.savefig('picturesNote/006_spctrgaus.png', dpi=70)
plt.show()
```

Выполнив данный код, получаем график сигнала:

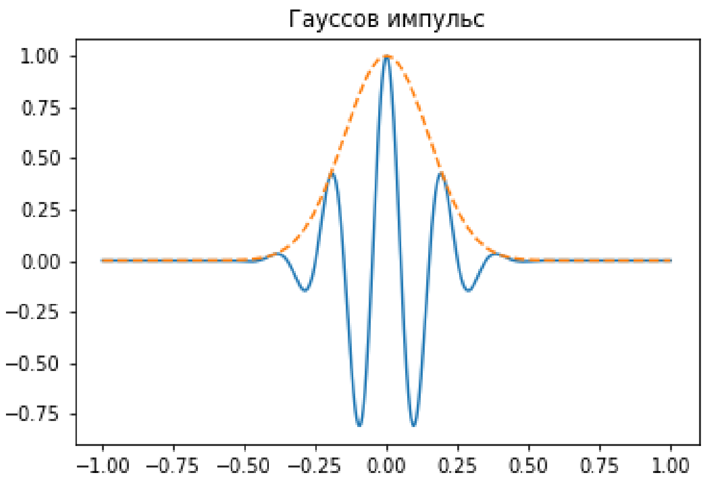


Рисунок 4.5. Импульс Гаусса

Кроме сигнала, для примера так же построили и спектр Гауссова импульса.

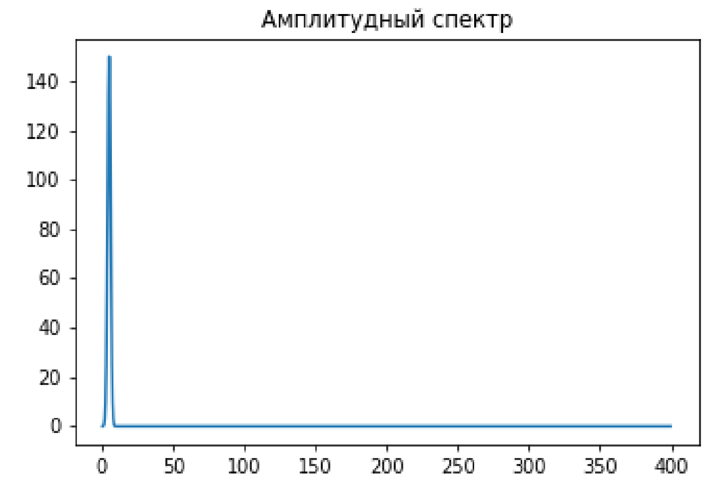


Рисунок 4.6. Спектр Импульс Гаусса

## 4.3. Генерация периодических сигналов

Периодические сигналы - это функции, которые позволяют формировать отсчеты периодических сигналов различной формы. Рассмотрим последовательность уже знакомых прямоугольных импульсов.

Листинг 6: source06.py

```
import numpy as np
import math
import matplotlib.pyplot as plt
from scipy import signal

Fs = 1000 # discrete frequency
t = np.matrix(np.arange(-0.4, 0.8, (1 / Fs))).transpose()
A = 5
fig = plt.figure()
plt.plot(t, A *signal.square( 2 * np.pi * 5 * t, 0.2))
fig.savefig('picturesNote/007rectImplses.png', dpi=100)
plt.show()
```

Выполнив данный код, получаем график сигнала:

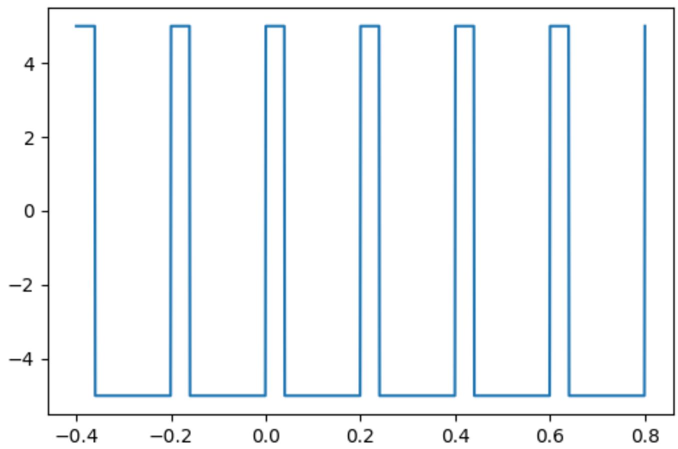


Рисунок 4.7. Прямоугольный периодический сигнал

С помощью функции sawtooth из scipy.signal реализуем последовательность треугольных импульсов.

Листинг 7: source07.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

Fs = 8000 # discrete frequency
t = np.linspace(0, 0.5, Fs)
fig = plt.figure()
plt.title('Последовательность треугольных импульсов')
plt.plot(t, signal.sawtooth(2 * np.pi * 5  * t))
fig.savefig('picturesNote/008_sawtooth.png', dpi=100)
plt.show()
```

Выполнив данный код, получаем график сигнала:

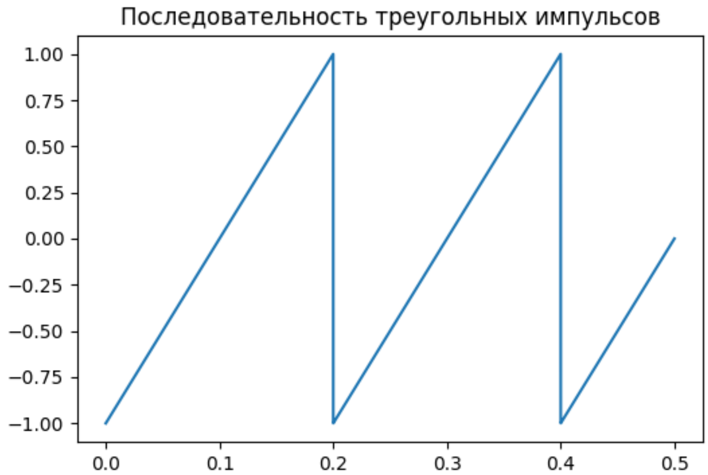


Рисунок 4.8. Последовательность треугольных импульсов

Реализуем также функцию Дирихле(**??**):

Листинг 8: source08.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from scipy import special


x = np.linspace(-8*np.pi, 8*np.pi, num=201)
fig = plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(x, special.diric(x, 7))
plt.title('Функция Дирихле, n={}'.format(7))
plt.subplot(2, 1, 2)
plt.plot(x, special.diric(x, 8))
plt.title('Функция Дирихле, n={}'.format(8))
fig.savefig('pictures/009_dirichle.png', dpi = 200)
plt.show()
```

Выполнив данный код, получаем график сигнала:

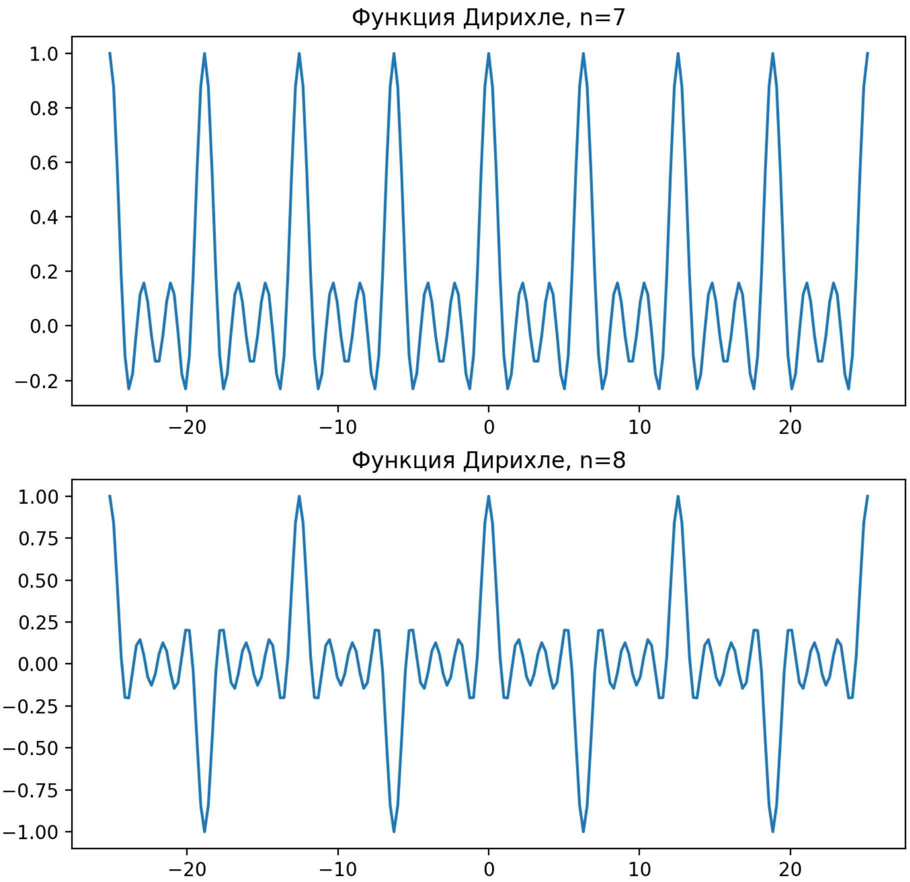


Рисунок 4.9. Функции Дирихле 7 и 8 степеней

# 5. Вывод

В результате выполнению данной работы нам удалось получить элементарные навыки использования python и JupyterNotebook для анализа и моделирования сигналов. Также мы узнали о различных видах сигналов:

- гармонические сигналы
- одиночные импульсы
- периодические сигналы
- сигналы с меняющейся частотой