

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

# Телекоммуникационные технологии

Отчет по лабораторной работе №1  
Сигналы телекоммуникационных систем

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

Санкт-Петербург  
2017

# Содержание

<b>1. Цель работы</b>	<b>2</b>
<b>2. Постановка задачи</b>	<b>2</b>
<b>3. Теоретическая информация</b>	<b>2</b>
<b>4. Ход работы</b>	<b>2</b>
4.1. Введение . . . . .	3
4.2. Одиночные импульсы . . . . .	6
4.3. Генерация периодических сигналов . . . . .	8
<b>5. Вывод</b>	<b>11</b>

## 1. Цель работы

Познакомиться со средствами генерации и визуализации простых сигналов, с различными программными средствами цифровой 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:

$$s_1(t) = \cos(2\pi f_0 t + \phi) \quad (1)$$

- затухающий гармонический сигнал:

$$s_2(t) = \cos(2\pi f_0 t + \phi)e^{-\alpha t} \quad (2)$$

- Гауссов импульс:

$$y(t) = \cos(2\pi f_c t)e^{-\alpha t^2} \quad (3)$$

- Функция Дирихле:

$$\frac{\sin(nx/2)}{n\sin(x/2)}, n \in \mathbb{Z} \quad (4)$$

## 4. Ход работы

Для реализации гармонического сигнала в соответствии с формулой (1) нам необходимо также реализовать функцию поэлементного умножения `elem_multiply` по аналогии с MATLAB.

## 4.1. Введение

Листинг 1: source01.py

```
1 import numpy as np
2 import math
3 import matplotlib.pyplot as plt
4
5 def elem_multiple(a, b):
6     if a.shape != b.shape:
7         raise IOError("Shapes_of_matrices_are_not_the_same")
8     return
9     else:
10        res = np.zeros(a.shape)
11        for i in range(a.shape[0]):
12            for j in range(a.shape[1]):
13                res[(i,j)] = a[(i,j)] * b[(i,j)]
14        # print(res.shape)
15        return res
16
17 Fs = 8000 # discrete frequency
18 t = np.matrix(np.arange(0.0, 1.0, (1 / Fs))).transpose()
19 A = 2 # amplitude
20 f0 = 1000
21 phi = np.pi /4
22 s1 = A * np.cos(2 * np.pi * f0 * t + phi) # harmonic signal
23 fig = plt.figure(figsize=(6,6))
24 plt.title('Гармонический_сигнал')
25 plt.plot(t[1:100], s1[1:100])
26 fig.savefig('picturesNote/001harmonic.png', dpi=200)
27 plt.show()
```

Выполнив данный код, получаем график сигнала:

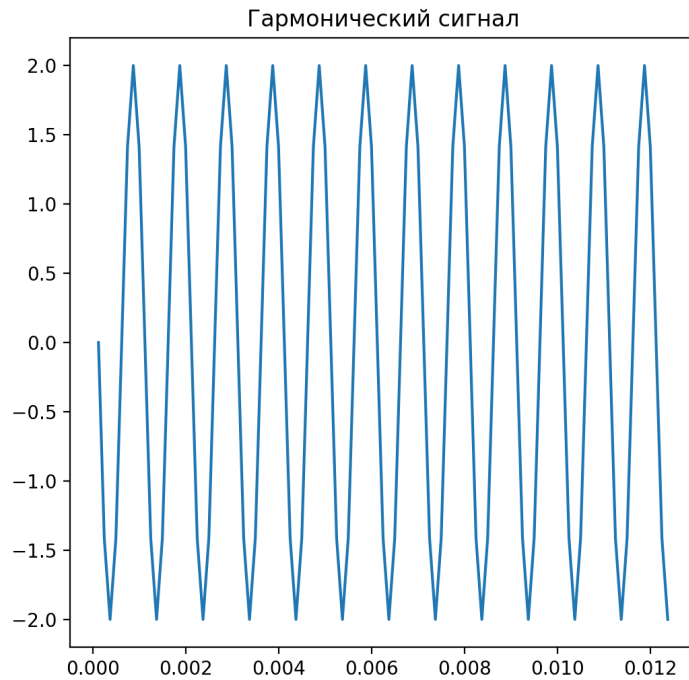


Рисунок 4.1. Гармонический сигнал

Далее реализуем уже затухающий гармонический сигнал (2):

Листинг 2: source02.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def elem_multiple(a, b):
5     if a.shape != b.shape:
6         raise IOError("Shapes_of_matrices_are_not_the_same")
7     return
8     else:
9         res = np.zeros(a.shape)
10        for i in range(a.shape[0]):
11            for j in range(a.shape[1]):
12                res[(i,j)] = a[(i,j)] * b[(i,j)]
13        # print(res.shape)
14        return res
15
16 Fs = 8000 # discrete frequency
17 t = np.matrix(np.arange(0.0, 1.0, (1 / Fs))).transpose()
18 f0 = 1000
19 phi = np.pi / 4
20 alpha = 1000
21 s2 = elem_multiple(s1, np.exp(-alpha * t))
22 fig = plt.figure(figsize=(6,6))
23 plt.title('Затухающий_сигнал')
24 plt.plot(t[1:100], s2[1:100])
25 fig.savefig('picturesNote/002harmonic.png', dpi=100)
26 plt.show()

```

Выполнив данный код, получаем график сигнала:

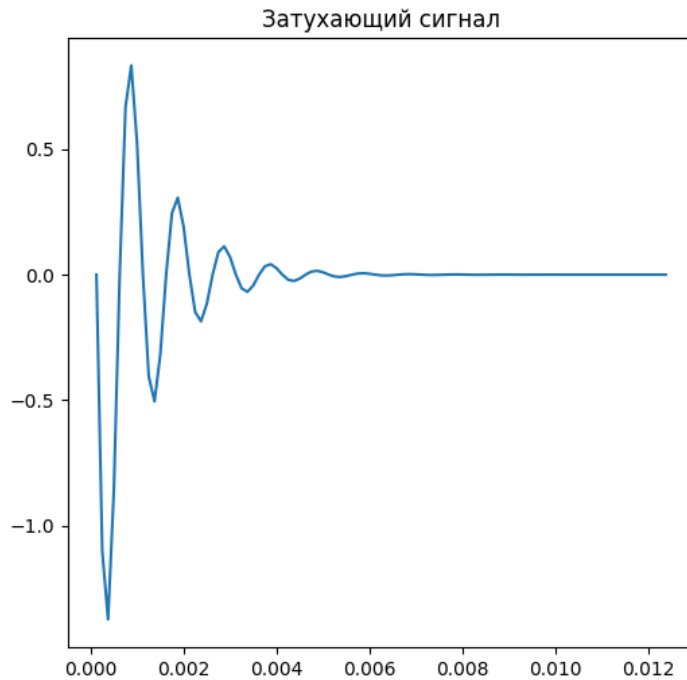


Рисунок 4.2. Затухающий гармонический сигнал

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

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 Fs = 40000 # discrete frequency
6 t = np.matrix(np.arange(0.0, 1.0, (1 / Fs))).transpose()
7 f = np.matrix([600, 800, 1000, 1200, 1400])
8
9 s3 = np.matrix(np.cos(2*np.pi * t * f)) # 5 channels signal
10
11 fig = plt.figure(figsize=(8, 8))
12 plt.title('канальный5-сигнал')
13 plt.plot(t[1:50], s3[1:50])
14 fig.savefig('picturesNote/003_5chnls.png', dpi=200)
15 plt.show()

```

Выполнив данный код, получаем график сигнала:

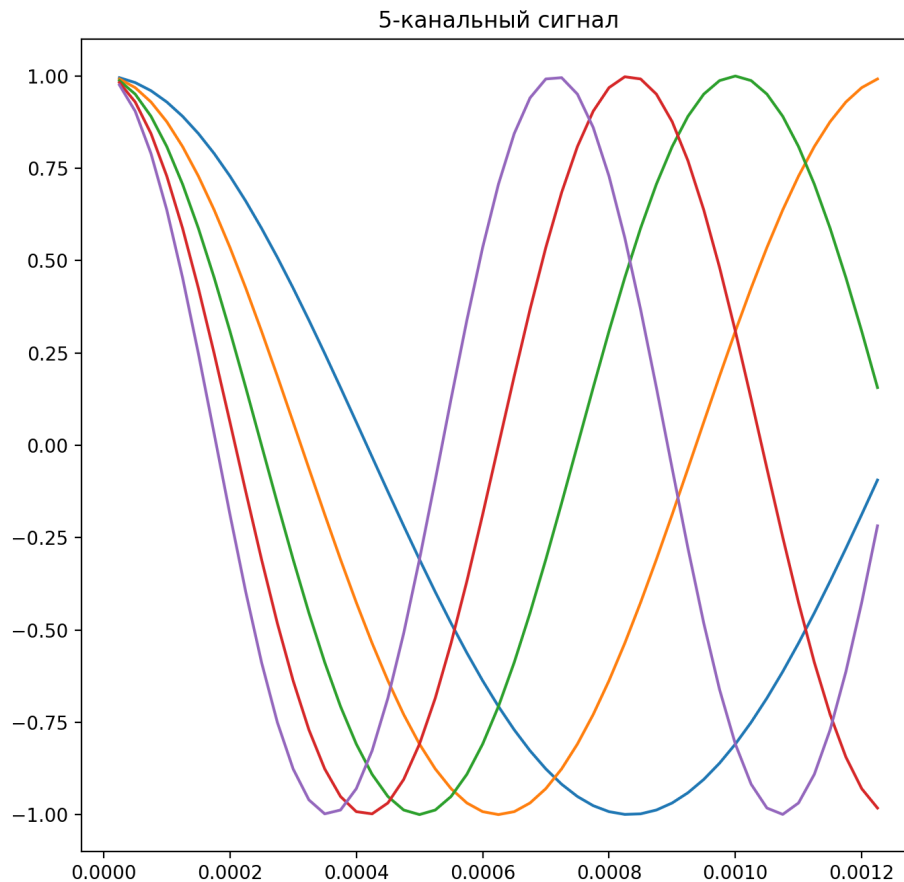


Рисунок 4.3. Многоканальный сигнал

В соответствии 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 импульсов, поэтому для примера реализуем прямоугольный импульс самостоятельно.

Листинг 4: `source04.py`

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # rectangular impulse
5 def rect_impls_1(t, width):
6     sig = np.zeros(len(t))
7     for i in sig:
8         if -width/2 <= i < width / 2:
9             i = 1
10    for i in range(len(sig)):
11        if -width/2 <= t[i] < width / 2:
```

```

12     sig[i] = 1
13     return sig
14
15 Fs = 1000
16 t = np.matrix(np.arange(-0.84, 0.84, (1 / Fs))).transpose()
17 width = 0.2
18 A = 5
19 s = -A * rect_impls_1(t + width / 2, width) + A * rect_impls_1(t - width / 2 ,
    ↪ width)
20 plt.title('Прямоугольный импульс')
21 plt.plot(t,s)
22 plt.gcf().savefig('picturesNote/004rectImp.png', dpi=100)
23 plt.show()

```

Выполнив данный код, получаем график сигнала:

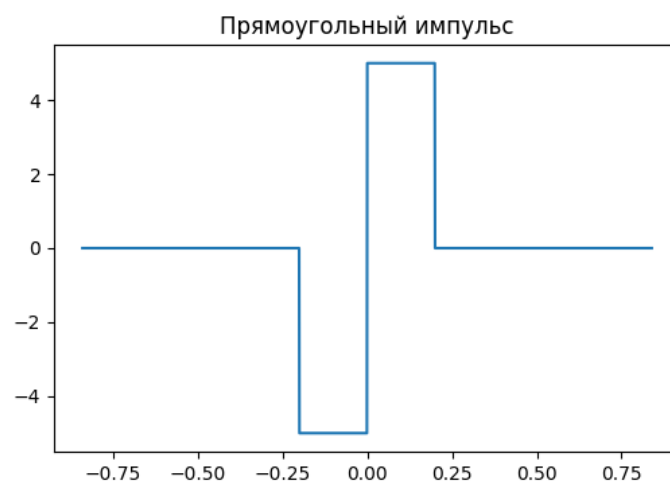


Рисунок 4.4. Прямоугольный импульс

Полученный график очевидно является прямоугольным импульсом.

Также построим импульс Гаусса, реализация которого представлена в `scipy.signal`. По определению Гауссов импульс представляется формулой (3).

Листинг 5: source05.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy import signal
4 from scipy.fftpack import fft, fftfreq
5
6 Fs = 800
7 t = np.linspace(-1, 1, 2 * Fs)
8 i, e = signal.gausspulse(t, fc=5, retenv=True)
9 fig = plt.figure()
10 plt.plot(t, i, t, e, '—')
11 plt.title('Гауссов импульс')
12 fig.savefig('picturesNote/005_gaus.png', dpi=70)
13 plt.show()
14 N = len(t)
15 Fs = 800
16 T = 1.0 / Fs
17 y = i

```



```

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

```

Выполнив данный код, получаем график сигнала:

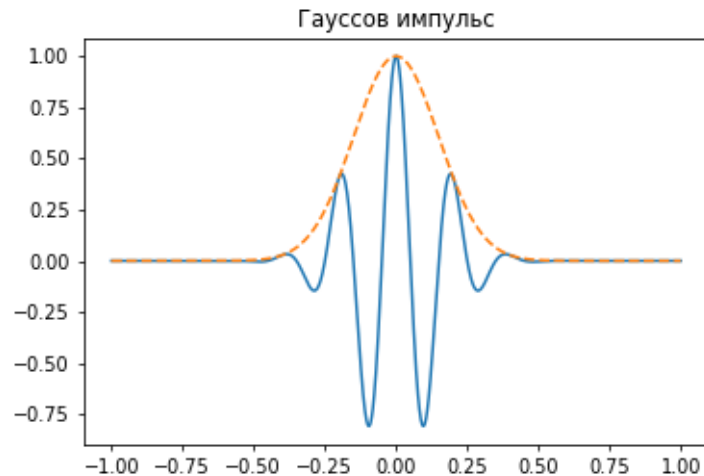


Рисунок 4.5. Импульс Гаусса

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

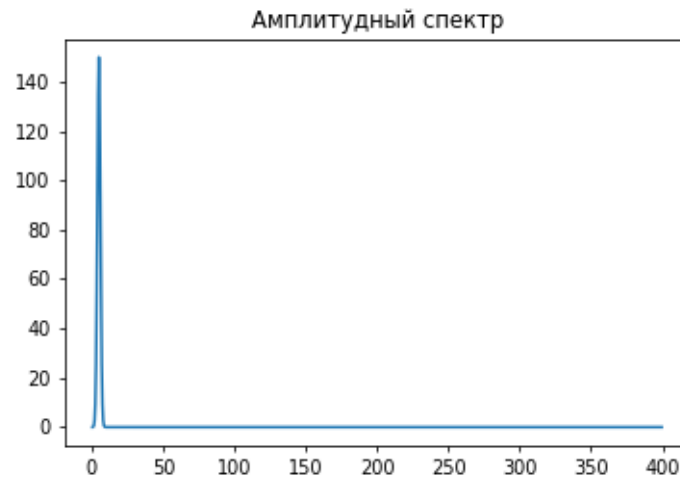


Рисунок 4.6. Спектр Импульс Гаусса

### 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

```

1 import numpy as np
2 import math
3 import matplotlib.pyplot as plt
4 from scipy import signal
5
6 Fs = 1000 # discrete frequency
7 t = np.matrix(np.arange(-0.4, 0.8, (1 / Fs))).transpose()
8 A = 5
9 fig = plt.figure()
10 plt.plot(t, A * signal.square(2 * np.pi * 5 * t, 0.2))
11 fig.savefig('picturesNote/007rectImplses.png', dpi=100)
12 plt.show()

```

Выполнив данный код, получаем график сигнала:

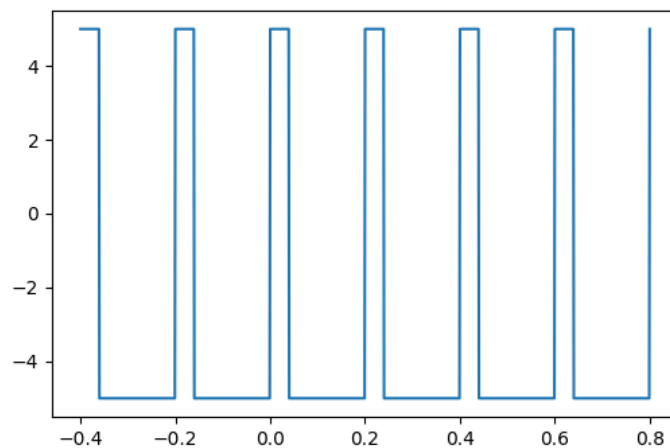


Рисунок 4.7. Прямоугольный периодический сигнал

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

Листинг 7: source07.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy import signal
4
5 Fs = 8000 # discrete frequency
6 t = np.linspace(0, 0.5, Fs)
7 fig = plt.figure()
8 plt.title('Последовательность_треугольных_импульсов')
9 plt.plot(t, signal.sawtooth(2 * np.pi * 5 * t))
10 fig.savefig('picturesNote/008_sawtooth.png', dpi=100)
11 plt.show()

```

Выполнив данный код, получаем график сигнала:

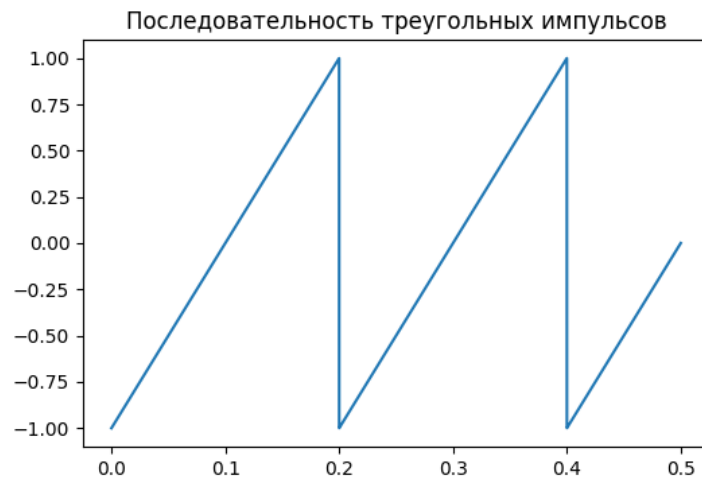


Рисунок 4.8. Последовательность треугольных импульсов

Реализуем также функцию Дирихле(4):

Листинг 8: source08.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy import signal
4 from scipy import special
5
6
7 x = np.linspace(-8*np.pi, 8*np.pi, num=201)
8 fig = plt.figure(figsize=(8, 8))
9 plt.subplot(2, 1, 1)
10 plt.plot(x, special.diric(x, 7))
11 plt.title('Функция Дирихле, n={}'.format(7))
12 plt.subplot(2, 1, 2)
13 plt.plot(x, special.diric(x, 8))
14 plt.title('Функция Дирихле, n={}'.format(8))
15 fig.savefig('pictures/009_dirichle.png', dpi = 200)
16 plt.show()

```

Выполнив данный код, получаем график сигнала:

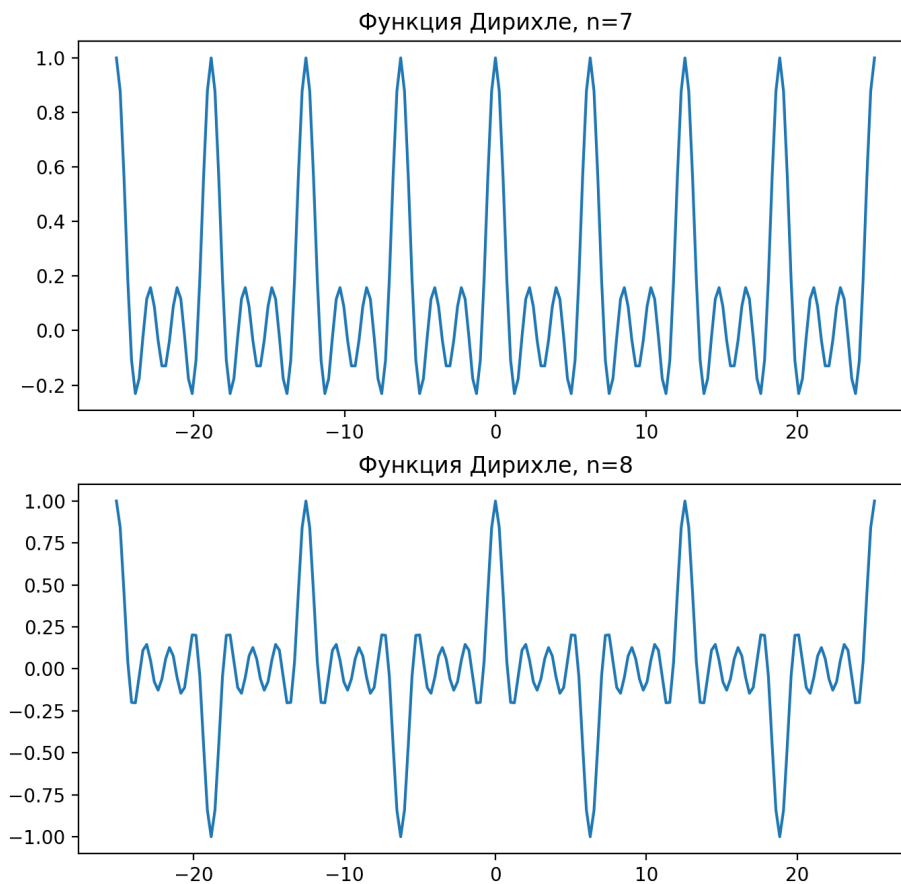


Рисунок 4.9. Функции Дирихле 7 и 8 степеней

## 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