

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное автономное образовательное учреждение высшего образования  
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №42

КУРСОВАЯ РАБОТА (ПРОЕКТ)  
ЗАЩИЩЕНА С ОЦЕНКОЙ  
РУКОВОДИТЕЛЬ

\_\_\_\_\_  
ассистент  
должность, уч. степень, звание

\_\_\_\_\_  
подпись, дата

\_\_\_\_\_  
Н.А.Янковский  
инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
К КУРСОВОЙ РАБОТЕ (ПРОЕКТУ)

по дисциплине:  
ЦИФРОВАЯ ОБРАБОТКА И ПЕРЕДАЧА СИГНАЛОВ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № \_\_\_\_\_ 4128

\_\_\_\_\_  
подпись, дата

\_\_\_\_\_  
В. А. Воробьев  
инициалы, фамилия

Санкт-Петербург  
2023

## СОДЕРЖАНИЕ

1 СПИСОК СОКРАЩЕНИЙ	3
2 ТЕХНИЧЕСКОЕ ЗАДАНИЕ	4
3 ВВЕДЕНИЕ	5
4 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ	6
4.1 Рэлеевские замирания	6
4.2 Аддитивный белый гауссовский шум	7
4.3 Равномерная дискретизация	8
4.4 Равномерное квантование	9
4.5 Фильтр Баттерворта	10
5 ВЫПОЛНЕНИЕ ЗАДАНИЯ	12
5.1 Выбор технологий	12
5.2 Структура программы	13
5.3 Результат работы	18
ЗАКЛЮЧЕНИЕ	20
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	21
ПРИЛОЖЕНИЕ	22

## **1 СПИСОК СОКРАЩЕНИЙ**

АЦП - Аналого-цифровое преобразование

ЦАП - Цифроаналоговое преобразование

ФНЧ - Фильтр нижних частота

АБГШ - Аддитивный белый гауссовский шум

ЛАЧХ - Логарифмическая амплитудно-частотная характеристика

ШИМ - Широтно-импульсная модуляция

## 2 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Реализовать АЦП в соответствии с рисунком 1.

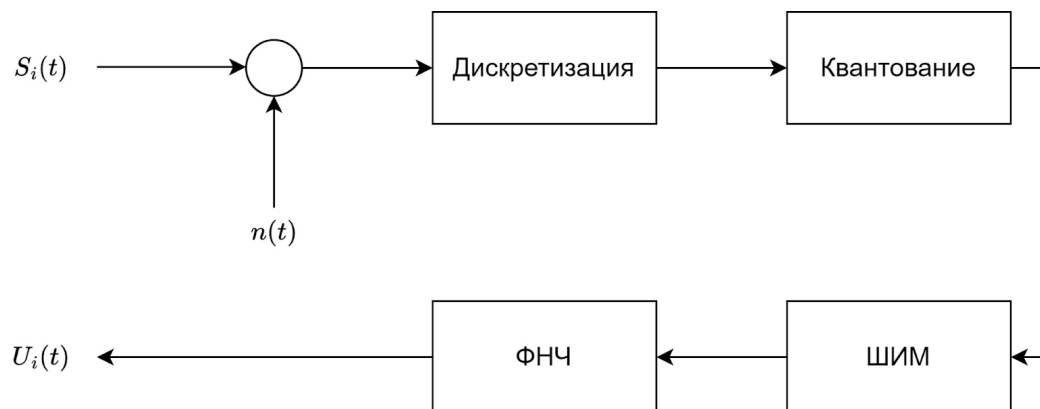


Рисунок 1 - модель АЦП

Для реализации данной модели требуется выбрать методы дискретизации, квантования, ФНЧ и генерации шума.

Номер варианта = номер в списке группы  $\text{mod } 20 = 5 \% 20 = 5$

В соответствии с вариантом к работе были выбраны следующие методы.

Таблица 1 - Вариант задания

Модель шума	Рэлеевские замирания + АБГШ
Дискретизация	Равномерная дискретизация
Квантование	Равномерное
Фильтр нижних частот	Баттерворта

### **3 ВВЕДЕНИЕ**

Цифровая обработка сигналов является важным направлением в современной технике и технологиях, находя применение в различных областях, начиная от телекоммуникаций и медицинских технологий и заканчивая звукозаписью и изображением. Это обширное поле исследований охватывает множество аспектов, включая методы аналого-цифрового и цифроаналогового преобразования сигналов, которые играют ключевую роль в обработке информации.

Наша работа направлена на раскрытие основных аспектов аналого-цифрового и цифроаналогового преобразования, а также на практическое применение разработанных методов для создания эффективных алгоритмов обработки сигналов. Полученные результаты не только позволят лучше понять процессы обработки сигналов, но и предоставят основу для оптимизации их применения в различных областях техники и информационных технологий.

## 4 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

### 4.1 Рэлеевские замирания

Рэлеевский шум представляет собой особый вид шума, который возникает в процессе передачи сигналов и воздействует на аналого-цифровое преобразование. Этот шум характеризуется случайными изменениями амплитуды сигнала, причем эти изменения происходят с некоторой случайной периодичностью. Рэлеевский шум может быть описан распределением Рэля [1]. Плотность распределения Рэля изображена на рисунке 2.

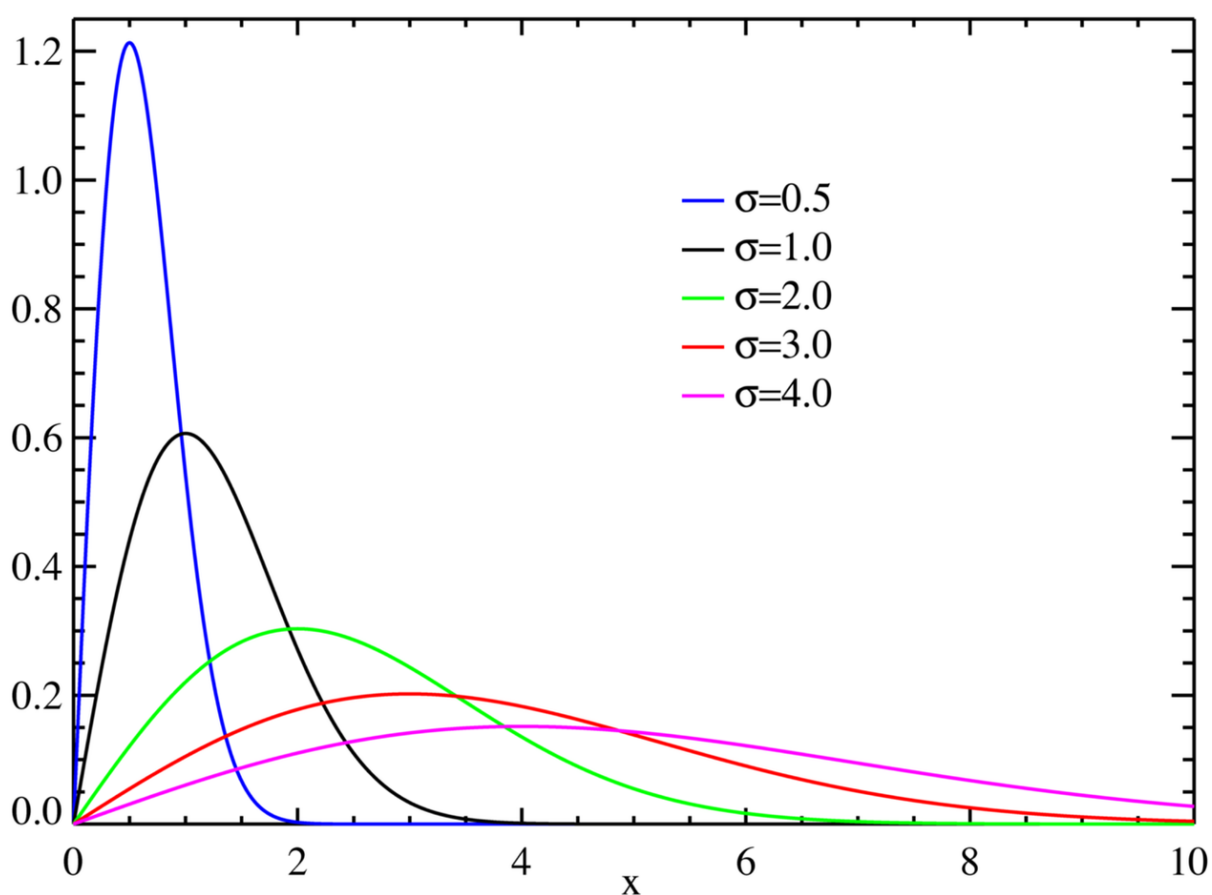


Рисунок 2 - Плотность распределения Рэля

Характеристика Рэлеевских замираний заключается в том, что

вероятность изменения амплитуды сигнала уменьшается с увеличением ее значения. Это означает, что небольшие изменения амплитуды происходят чаще, чем более значительные.

#### **4.2 Аддитивный белый гауссовский шум**

АБГШ является одним из наиболее распространенных типов шума в аналого-цифровых системах. Этот шум характеризуется случайными изменениями амплитуды сигнала, распределенными по нормальному закону. Он обусловлен множеством случайных факторов, таких как тепловое движение электронов в устройствах и прочие случайные воздействия на сигнал.

Характеристика АБГШ заключается в том, что его спектральная плотность постоянна на всех частотах, что означает, что он присутствует в равной мере во всех диапазонах частот. Это свойство делает его "белым" шумом [2].

Формула для генерации АБГШ проста и представляется выражением  $X(t) = A + N(t)$ , где  $A$  - амплитуда сигнала,  $N(t)$  - случайная величина с нормальным распределением, представляющая шум. Плотность нормального распределения изображена на рисунке 3.

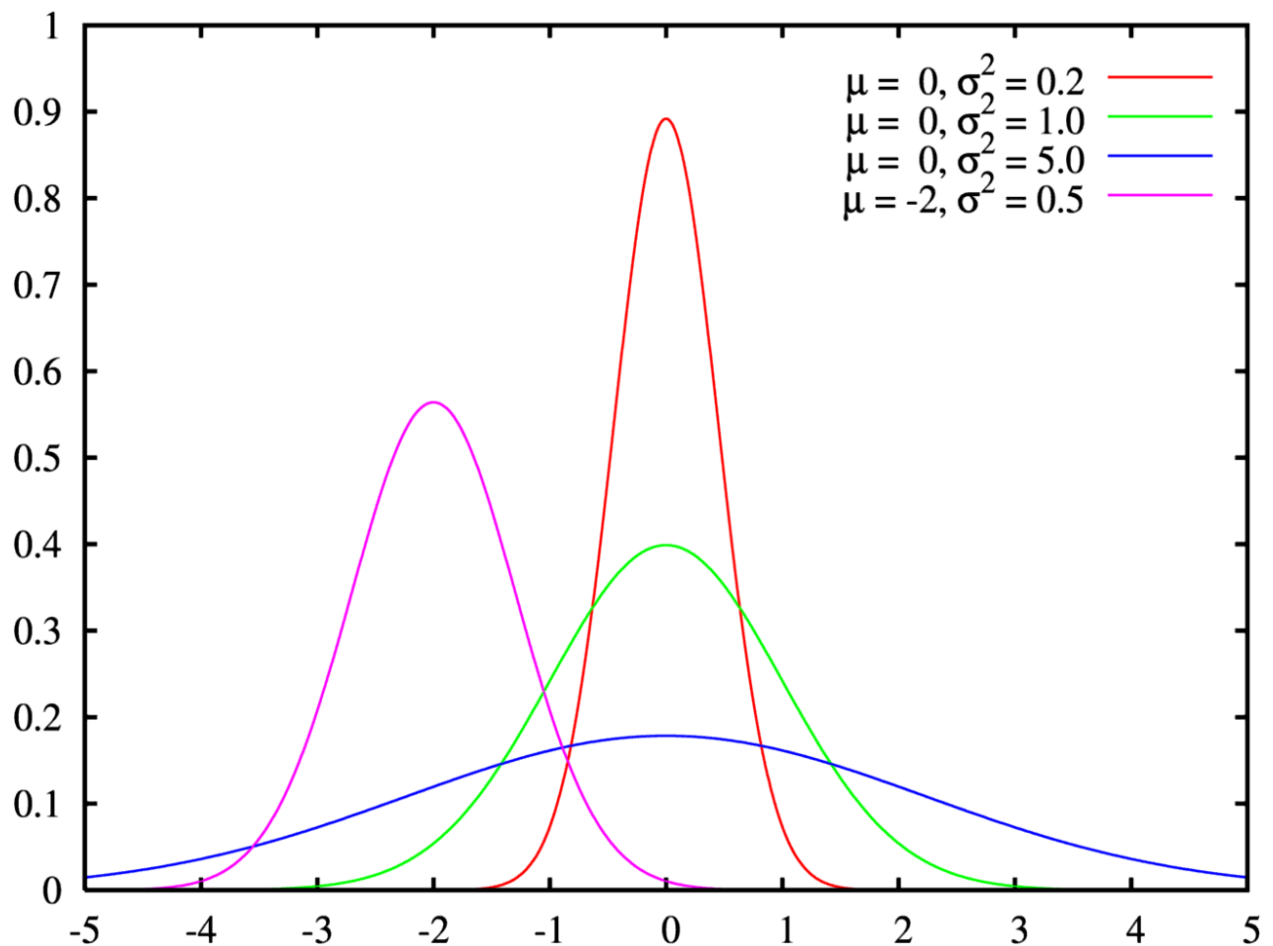


Рисунок 3 - Плотность нормального распределения

### 4.3 Равномерная дискретизация

Равномерная дискретизация представляет собой процесс преобразования непрерывного аналогового сигнала в последовательность дискретных отсчетов с постоянным интервалом времени [3]. Этот метод дискретизации широко используется в аналого-цифровых системах для представления аналоговых данных в цифровой форме. Пример равномерной дискретизации показан на рисунке 4.



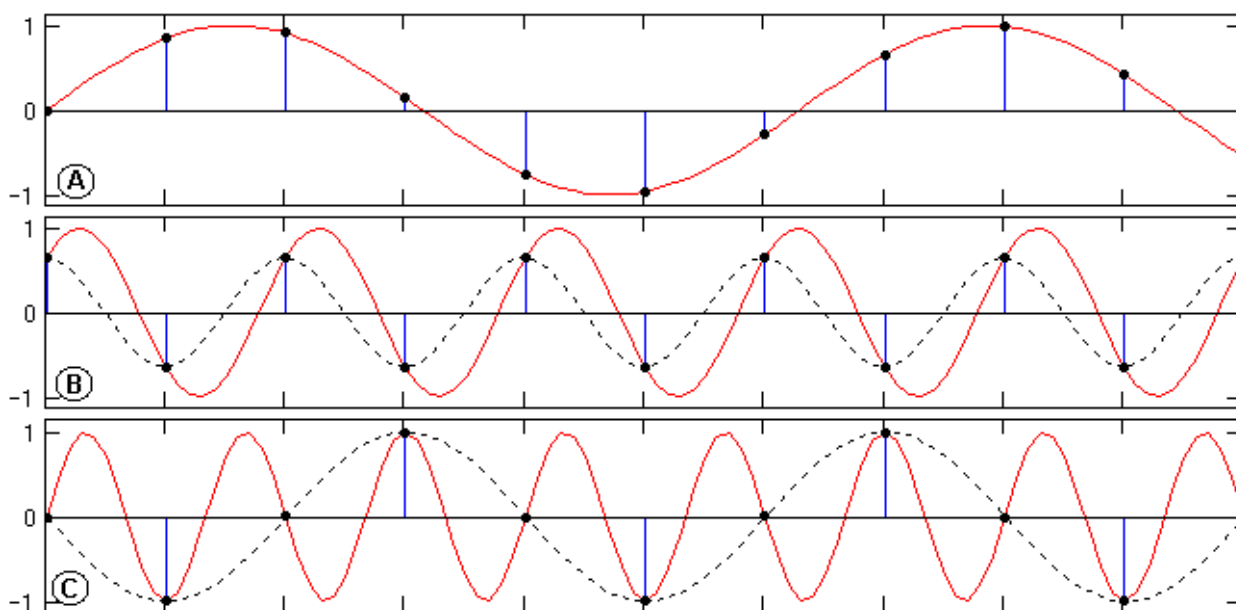


Рисунок 4 - Пример равномерной дискретизации

#### 4.4 Равномерное квантование

Равномерное квантование представляет собой процесс присвоения дискретных значений амплитуды аналогового сигнала, который был предварительно дискретизирован. Этот процесс играет ключевую роль в преобразовании аналогового сигнала в цифровую форму, где амплитуды предоставляются конечным числом уровней. Иллюстрация равномерного квантования изображена на рисунке 5.

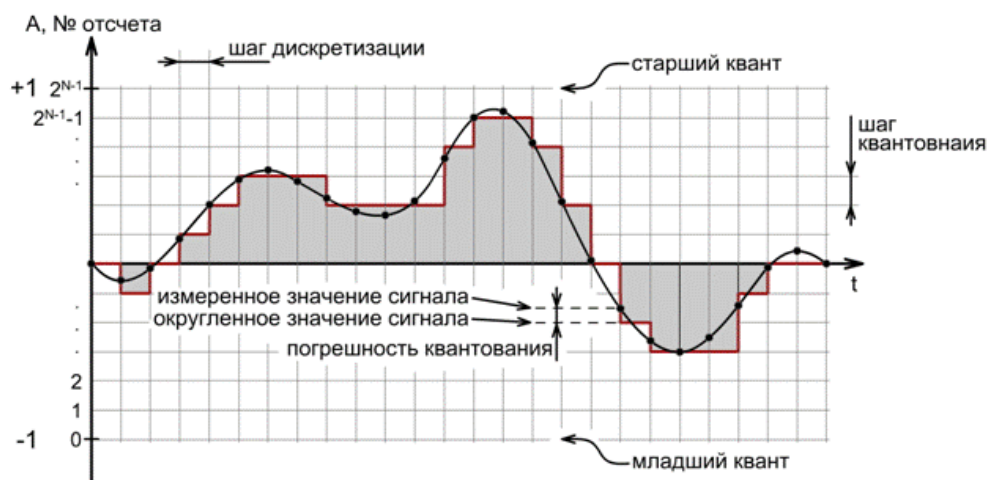


Рисунок 5 - Пример равномерного квантования

Характеристика равномерного квантования заключается в том, что диапазон возможных значений амплитуды разбивается на фиксированное число интервалов, называемых квантов. Каждый квант представляет определенный уровень амплитуды, и значение амплитуды внутри каждого кванта округляется до ближайшего уровня квантования.

#### **4.5 Фильтр Баттерворта**

Фильтр Баттерворта – это тип частотного фильтра, широко используемого в аналого-цифровом преобразовании для обработки сигналов. Этот фильтр принадлежит к классу бесконечной импульсной характеристики и обладает рядом характеристик, делающих его полезным инструментом в различных областях сигнальной обработки.

Характеристика фильтра Баттерворта включает в себя плоский частотный отклик в проходящей полосе и крутой спад в полосе подавления [4]. Это позволяет ему эффективно подавлять или усиливать сигналы в заданном частотном диапазоне, сохраняя при этом линейность фазовой характеристики. ЛАЧХ фильтра показана на рисунке 6.

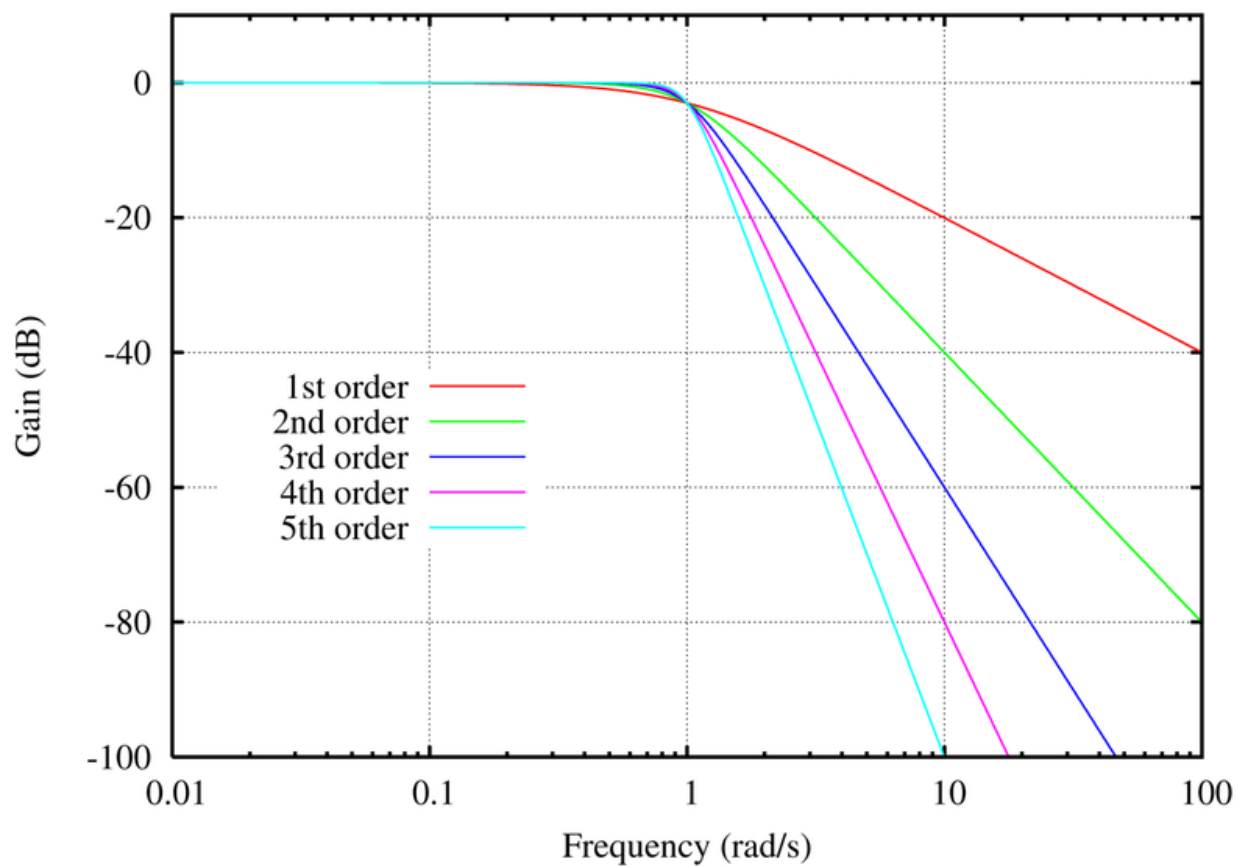


Рисунок 6 - ЛАЧХ для фильтров Баттерворта нижних частот

## **5 ВЫПОЛНЕНИЕ ЗАДАНИЯ**

### **5.1 Выбор технологий**

В выборе языка программирования для реализации курсовой работы принято решение в пользу Python. Этот язык программирования широко используется в области научных и инженерных исследований благодаря своей простоте, гибкости и обширному сообществу разработчиков. Python обеспечивает удобный синтаксис и множество библиотек, что делает его отличным выбором для решения задач цифровой обработки сигналов.

В принятии решения о выборе инструментария для курсовой работы, отказ от использования Matlab обоснован рядом факторов, несмотря на его широкое распространение в области обработки сигналов и инженерных расчетов. Python с его множеством библиотек обладает большей гибкостью и расширяемостью. Математическая база языка Python также является весьма разнообразной и обширной, что обеспечивает исследователей необходимыми средствами для реализации сложных алгоритмов.

В ходе работы было принято решение использовать стандартные библиотеки Python, такие как `math`, и библиотеки для научных вычислений, включая `NumPy` и `SciPy`. `NumPy` предоставляет эффективные средства для работы с многомерными массивами данных, а `SciPy` предлагает богатый набор функций для научных вычислений, включая методы обработки сигналов. Использование этих библиотек упрощает реализацию алгоритмов цифровой обработки сигналов и обеспечивает высокую производительность вычислений.


Важным компонентом выбора инструментария для курсовой работы стало также решение в пользу библиотеки `Matplotlib`. `Matplotlib` предоставляет мощные средства для визуализации данных, что является важным аспектом при анализе результатов цифровой обработки сигналов [5]. Визуализация сигналов, спектров и других характеристик обеспечивает наглядность и понимание процессов, происходящих в рамках курсовой

работы. Такой выбор библиотеки позволяет создавать высококачественные графики и диаграммы, что существенно облегчает восприятие результатов и выводы, сделанные в процессе исследования.

## 5.2 Структура программы

Сначала были написаны функции для генерации сигнала и шума. Исходный код изображен на рисунке 7. Поясним код:

- `enerate_rayleigh_noise(sigma: float = 0.3) -> float:`
  - Эта функция генерирует случайный шум с распределением Рэля (Rayleigh noise) с заданным параметром `sigma`.
  - Используется случайное число `u` из равномерного распределения и преобразуется в шум с помощью функции распределения Рэля.
- `generate_awgn_noise(mean: float = 0, std_dev: float = 1) -> float:`
  - Эта функция генерирует случайный шум с нормальным распределением.
  - Параметры `mean` и `std_dev` представляют среднее значение и стандартное отклонение нормального распределения соответственно.
- `generate_noise(coef: float) -> float:`
  - Эта функция комбинирует шумы с нормальным и распределением Рэля, умножая каждый из них на коэффициент `coef`.
- `st(t: float) -> float:`
  - Эта функция представляет собой пример синусоидального сигнала.



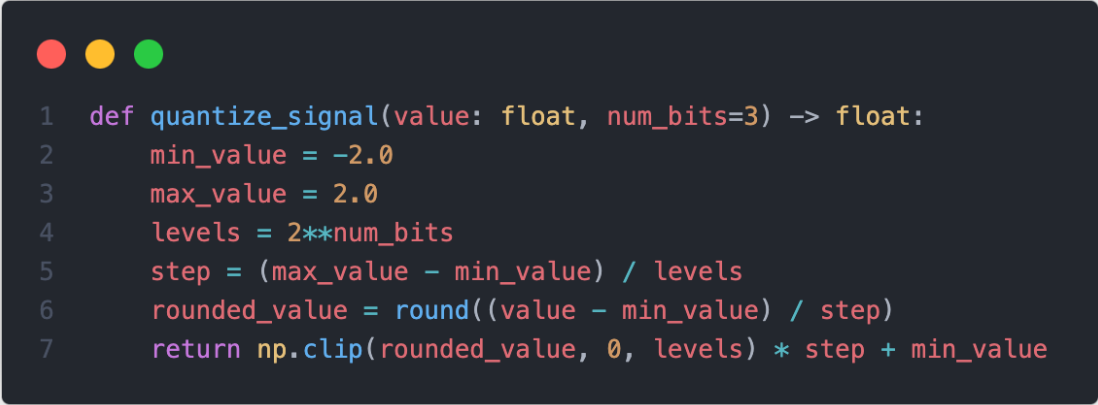
```

1  def generate_rayleigh_noise(sigma: float = 0.3) -> float:
2      u = random.random()
3      return sigma * math.sqrt(-2 * math.log(1 - u))
4
5
6  def generate_awgn_noise(mean: float = 0, std_dev: float = 1) -> float:
7      return random.gauss(mean, std_dev)
8
9
10 def generate_noise(coef: float) -> float:
11     return (generate_awgn_noise() + generate_rayleigh_noise()) * coef
12
13
14 def st(t: float) -> float:
15     return 2 * np.sin(t * 2 * math.pi)

```

Рисунок 7 - Код генерации шума

Далее была написана функция для квантования сигнала `quantize_signal(value: float, num_bits=3) -> float`. Эта функция принимает входное значение `value` и количество бит `num_bits` для квантования. Затем определяет диапазон входного сигнала, вычисляет количество уровней квантования, округляет входное значение до ближайшего уровня квантования, усекает результат для обеспечения нахождения в пределах допустимого диапазона, и затем возвращает квантованное значение. Исходный код функции представлен на рисунке 8.



```
1 def quantize_signal(value: float, num_bits=3) -> float:
2     min_value = -2.0
3     max_value = 2.0
4     levels = 2**num_bits
5     step = (max_value - min_value) / levels
6     rounded_value = round((value - min_value) / step)
7     return np.clip(rounded_value, 0, levels) * step + min_value
```

Рисунок 8 - Код квантования

После перешли к реализации низкочастотного фильтра Баттерворта. Данная функция `butter_lowpass_filter` реализует применение низкочастотного фильтра Баттерворта к входному сигналу. Исходный код представлен на рисунке 9. Процесс фильтрации осуществляется в несколько этапов:

- 1) Функция вычисляет частоту Найквиста (`nyquist`), которая представляет половину частоты дискретизации сигнала. Затем, производится нормализация частоты среза (`normal_cutoff`), представляющая собой отношение желаемой частоты среза к частоте Найквиста.
- 2) Проектирование низкочастотного фильтра Баттерворта при помощи функции `signal.butter`. Здесь используется фильтр четвертого порядка, а параметры передаются в функцию на основе рассчитанных ранее `normal_cutoff`.
- 3) Полученные коэффициенты фильтра (`b` и `a`) используются для фильтрации входного сигнала при помощи функции `signal.filtfilt`. Эта функция обеспечивает двустороннюю фильтрацию сигнала, что позволяет минимизировать фазовые искажения [6].

```

1 def butter_lowpass_filter(signal_values: np.ndarray, cutoff_frequency: float, sampling_rate: float) -> np.ndarray:
2     nyquist = 0.5 * sampling_rate
3     normal_cutoff = cutoff_frequency / nyquist
4     b, a = signal.butter(4, normal_cutoff, btype='low', analog=False)
5
6     return signal.filtfilt(b, a, signal_values)

```

Рисунок 9 - Код ФНЧ

После был реализован код для ШИМ. Он изображен на рисунке 10.

```

1 def pwm_signal(value: float) -> float:
2     return 1 if value > 0 else 0

```

Рисунок 10 - Код ШИМ

В конце был написан код для отображения графиков. Основная его часть сосредоточена в функции update. Исходный код этой функции изображен на рисунке 11.

Функция update в представленном коде выполняет обновление данных и визуализацию временных рядов в реальном времени. Процесс обновления включает в себя несколько этапов, каждый из которых обрабатывает различные аспекты данных.

Первоначально, функция увеличивает значение переменной времени t на определенный интервал time\_interval, что представляет собой шаг по



времени. Затем, получает текущие значения синусоиды и шума для заданного момента времени. Данные синусоиды обновляются путем удаления старого значения и добавления нового, сохраняя тем самым историю значений.

Далее, функция формирует сигнал, как сумму синусоиды и шума, и также обновляет соответствующий массив данных. После этого, сигнал проходит через процесс квантования, где его значения аппроксимируются до ограниченного набора уровней. Результат квантования также сохраняется для последующего отображения.

Дополнительно, функция генерирует последовательность значений ШИМ на основе данных после квантования. Эти значения ШИМ отображаются на соответствующем графике.

И, наконец, функция применяет низкочастотный фильтр Баттерворта к данным ШИМ для удаления высокочастотных компонентов и отображает результат на последнем графике.

```

1  def update(_):
2      global t
3      t = t + time_interval / 1000
4
5      original_value = st(t)
6      original_data.pop(0)
7      original_data.append(original_value)
8      axs[0, 0].clear()
9      axs[0, 0].set_ylim([-2.1, 2.1])
10     axs[0, 0].plot(t_values, original_data[start:end])
11     axs[0, 0].set_title('Синусоида')
12
13     noise_value = generate_noise(coef=0.5)
14     noised_data.pop(0)
15     noised_data.append(noise_value)
16     axs[0, 1].clear()
17     axs[0, 1].set_ylim([-2.1, 2.1])
18     axs[0, 1].plot(t_values, noised_data[start:end])
19     axs[0, 1].set_title('Шум')
20
21     signal_value = original_data[-1] + noised_data[-1]
22     signal_data.pop(0)
23     signal_data.append(signal_value)
24     axs[1, 0].clear()
25     axs[1, 0].set_ylim([-2.5, 2.5])
26     axs[1, 0].plot(t_values, signal_data[start:end])
27     axs[1, 0].set_title('Сигнал')
28
29     quantized_value = quantize_signal(signal_value)
30     quantized_data.pop(0)
31     quantized_data.append(quantized_value)
32     axs[1, 1].clear()
33     axs[1, 1].set_ylim([-2.1, 2.1])
34     axs[1, 1].plot(t_values, quantized_data[start:end])
35     axs[1, 1].set_title('Квантование')
36
37     pwm_data = [pwm_signal(x) for x in quantized_data]
38     axs[2, 0].clear()
39     axs[2, 0].set_ylim([-0.1, 1.1])
40     axs[2, 0].plot(t_values, pwm_data[start:end])
41     axs[2, 0].set_title('ШИМ')
42
43     filtered_data = butter_lowpass_filter(
44         np.array(pwm_data), cutoff_frequency, sampling_rate)
45     axs[2, 1].clear()
46     axs[2, 1].set_ylim([-0.2, 1.2])
47     axs[2, 1].plot(t_values, filtered_data[start:end])
48     axs[2, 1].set_title('Фильтрация')

```

Рисунок 11 - Код обновления графиков

### 5.3 Результат работы

Для демонстрации работы покажем графики в два момента времени. Полученные графики изображены на рисунках 12 и 13.

Исследование методов АЦП и ЦАП

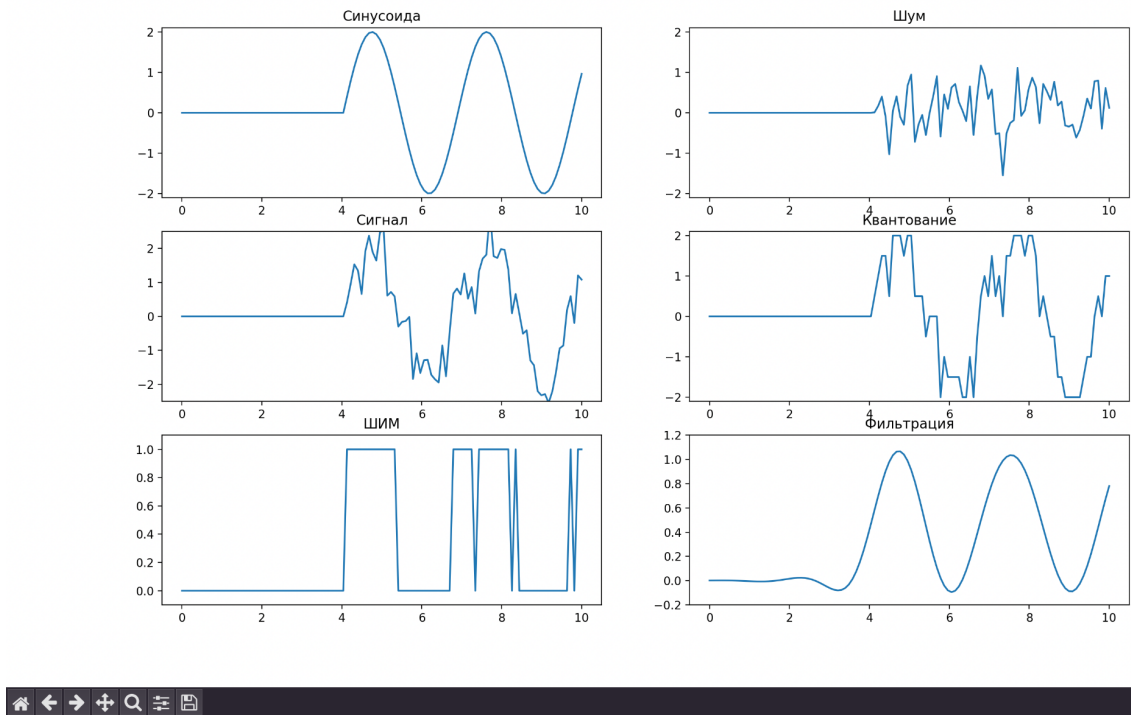


Рисунок 12 - Графики в момент запуска

Исследование методов АЦП и ЦАП

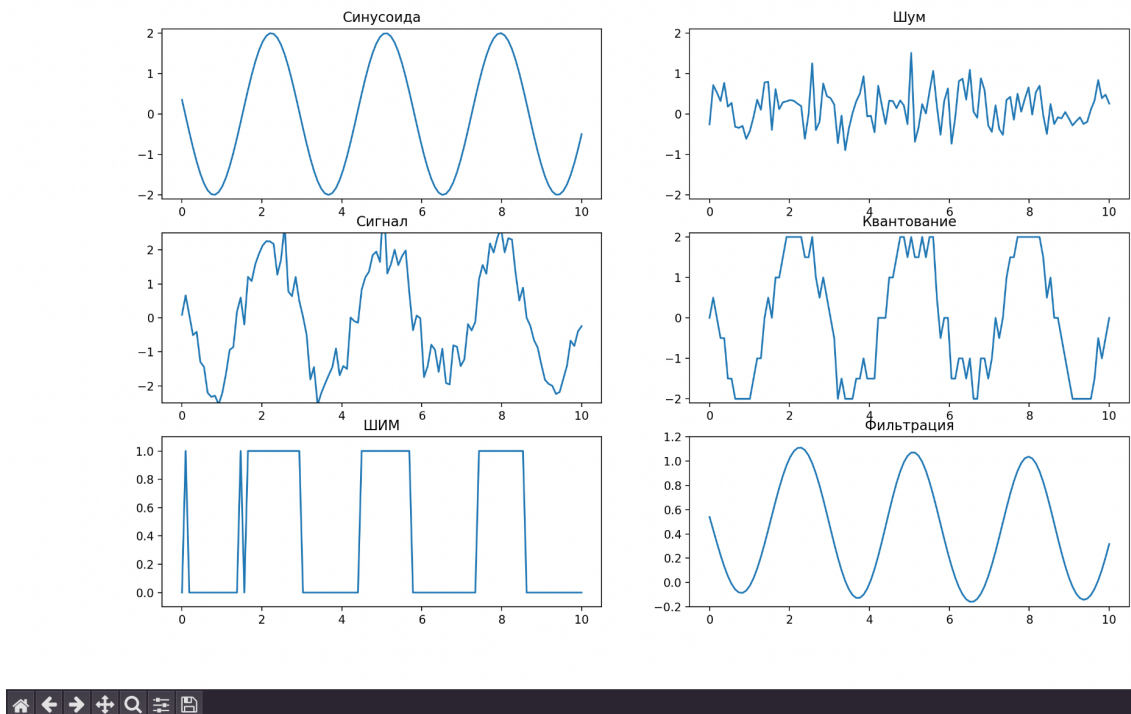


Рисунок 13 - Графики через промежуток времени

## ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы, мы успешно исследовали и реализовали ключевые аспекты цифровой обработки сигналов. Мы реализовали программу на языке Python с использованием библиотек NumPy, SciPy и Matplotlib. Код представлен в Приложении и на GitHub (URL: [https://github.com/vladcto/suai-labs/tree/355629db65b3539b04f89332b026364acb\\_d0016d/5\\_semester/%D0%A6%D0%9E%D0%9F%D0%A1/course\\_project](https://github.com/vladcto/suai-labs/tree/355629db65b3539b04f89332b026364acb_d0016d/5_semester/%D0%A6%D0%9E%D0%9F%D0%A1/course_project))

В ходе нашей работы мы достигли значительных результатов в области цифровой обработки сигналов. Изучив воздействие различных типов шума на сигналы, мы получили глубокое понимание их влияния, что позволит нам более эффективно управлять шумами в реальных условиях. Применение фильтра Баттерворта в обработке сигналов также позволило нам эффективно контролировать частотные характеристики системы.

Этот проект обогатил нас важными навыками в области цифровой обработки сигналов. Мы освоили методы моделирования шума, использования фильтров и применения алгоритмов дискретизации и квантования. Полученные навыки позволят нам успешно решать сложные задачи в области обработки сигналов и эффективно применять их в практических проектах.

Этот проект стал важным этапом в нашем профессиональном развитии, обогатив нас не только теоретическими знаниями, но и практическим опытом в области цифровой обработки сигналов. Полученные результаты и навыки предоставляют нам крепкую основу для дальнейших исследований и успешного применения в различных проектах, где цифровая обработка сигналов играет ключевую роль.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Распределения Рэля : сайт. – URL: [https://cyclowiki.org/wiki/Распределение\\_Рэля](https://cyclowiki.org/wiki/Распределение_Рэля) (дата обращения: 17.12.2023)
- 2 Аддитивный белый гауссовский шум : сайт. – URL: [https://ru.wikibrief.org/wiki/Additive\\_white\\_Gaussian\\_noise](https://ru.wikibrief.org/wiki/Additive_white_Gaussian_noise) (дата обращения: 17.12.2023)
- 3 Без паники! Цифровая обработка сигналов. / Юкио Сато :пер. с яп. Селиной .Т.Г М. : Додэка-XXI, 2010. —176 .с : ил. - Доп. тит. л яп. - ISBN 978-5-94120-251-5.
- 4 Фильтр Баттерворта : сайт. – URL: [https://ru.wikibrief.org/wiki/Butterworth\\_filter](https://ru.wikibrief.org/wiki/Butterworth_filter) (дата обращения: 17.12.2023)
- 5 Matplotlib : сайт. – URL: <https://matplotlib.org> (дата обращения: 17.12.2023)
- 6 Документация SciPy - Signal : сайт. – URL: <https://docs.scipy.org/doc/scipy/reference/signal.html#module-signal> (дата обращения: 17.12.2023)

## ПРИЛОЖЕНИЕ

### ИСХОДНЫЙ КОД

main.py

```
import copy
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from calculations import *

num_points = 150
cutoff_frequency = 0.5
sampling_rate = 10
time_interval = 32

fig, axs = plt.subplots(3, 2, figsize=(12, 6))
fig.suptitle("Исследование методов АЦП и ЦАП")

original_data = [0 for _ in range(num_points)]
noised_data = copy.copy(original_data)
signal_data = copy.copy(original_data)
quantized_data = copy.copy(original_data)
filtered_data = copy.copy(original_data)
pwm_data = copy.copy(original_data)

t = 0
end = num_points - 20
start = 20
```

```

length = end - start
t_values = np.linspace(0, 10, length)

def update(_):
    global t
    t = t + time_interval / 1000

    original_value = st(t)
    original_data.pop(0)
    original_data.append(original_value)
    axs[0, 0].clear()
    axs[0, 0].set_ylim([-2.1, 2.1])
    axs[0, 0].plot(t_values, original_data[start:end])
    axs[0, 0].set_title('Синусоида')

    noise_value = generate_noise(coef=0.5)
    noised_data.pop(0)
    noised_data.append(noise_value)
    axs[0, 1].clear()
    axs[0, 1].set_ylim([-2.1, 2.1])
    axs[0, 1].plot(t_values, noised_data[start:end])
    axs[0, 1].set_title('Шум')

    signal_value = original_data[-1] + noised_data[-1]
    signal_data.pop(0)
    signal_data.append(signal_value)
    axs[1, 0].clear()
    axs[1, 0].set_ylim([-2.5, 2.5])
    axs[1, 0].plot(t_values, signal_data[start:end])

```

```
axs[1, 0].set_title('Сигнал')
```

```
quantized_value = quantize_signal(signal_value)
quantized_data.pop(0)
quantized_data.append(quantized_value)
axs[1, 1].clear()
axs[1, 1].set_ylim([-2.1, 2.1])
axs[1, 1].plot(t_values, quantized_data[start:end])
axs[1, 1].set_title('Квантование')
```

```
pwm_data = [pwm_signal(x) for x in quantized_data]
axs[2, 0].clear()
axs[2, 0].set_ylim([-0.1, 1.1])
axs[2, 0].plot(t_values, pwm_data[start:end])
axs[2, 0].set_title('ШИМ')
```

```
filtered_data = butter_lowpass_filter(
    np.array(pwm_data), cutoff_frequency, sampling_rate)
axs[2, 1].clear()
axs[2, 1].set_ylim([-0.2, 1.2])
axs[2, 1].plot(t_values, filtered_data[start:end])
axs[2, 1].set_title('Фильтрация')
```

```
ani = FuncAnimation(fig, update, frames=num_points, interval=time_interval)
```

```
plt.show()
```

```
calculations.py
```



```

import math
import random

import numpy as np
from scipy import signal

def generate_rayleigh_noise(sigma: float = 0.3) -> float:
    u = random.random()
    return sigma * math.sqrt(-2 * math.log(1 - u))

def generate_awgn_noise(mean: float = 0, std_dev: float = 1) -> float:
    return random.gauss(mean, std_dev)

def generate_noise(coef: float) -> float:
    return (generate_awgn_noise() + generate_rayleigh_noise()) * coef

def st(t: float) -> float:
    return 2 * np.sin(t * 2 * math.pi)

def quantize_signal(value: float, num_bits=3) -> float:
    min_value = -2.0
    max_value = 2.0
    levels = 2**num_bits
    step = (max_value - min_value) / levels

```

```
rounded_value = round((value - min_value) / step)
return np.clip(rounded_value, 0, levels) * step + min_value
```

```
def butter_lowpass_filter(signal_values: np.ndarray, cutoff_frequency: float,
sampling_rate: float) -> np.ndarray:
    nyquist = 0.5 * sampling_rate
    normal_cutoff = cutoff_frequency / nyquist
    b, a = signal.butter(4, normal_cutoff, btype='low', analog=False)

    return signal.filtfilt(b, a, signal_values)
```

```
def pwm_signal(value: float) -> float:
    return 1 if value > 0 else 0
```