



**«Московский государственный технический университет
имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ

Информатики и систем управления

КАФЕДРА

Проектирования и технологии производства ЭА

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к домашнему заданию

по курсу «Киберфизические системы»

Вариант 5

Студент

(Подпись, дата) **К.С. Кутаев**
(И.О.Фамилия)

Преподаватель

(Подпись, дата) **В.В. Леонидов**
(И.О.Фамилия)

ВВЕДЕНИЕ

Общее техническое задание представляет собой задачу разработать программное обеспечение с графическим пользовательским интерфейсом на любом языке программирования. Входные тестовые данные для каждого варианта ДЗ должны загружаться из текстового, либо звукового файла (в зависимости от условия задания). Генерирование входных тестовых файлов допускается осуществлять с помощью скриптов Matlab. Использование библиотечных функций для реализации алгоритмов обработки сигналов не допускается (если в задании не указано иное).

Техническое задание по варианту - разработать программу, выполняющую функцию BPSK-модулятора и демодулятора. Для формирования модулированного сигнала пользователь задаёт следующие параметры: частоту дискретизации, несущую частоту, уровень добавляемого к сигналу шума, период передачи данных, количество данных и сами данные. Полученный сигнал (а также информация о частоте дискретизации, несущей частоте, периоде передачи данных) сохраняется в файл в произвольном формате и отображается на графике. Для демодуляции ранее созданного модулированного сигнала необходимо загрузить его из файла, произвести демодуляцию и отобразить полученные данные.

В данной работе представлена разработка BPSK модулятора и демодулятора дискретного сигнала. Разрабатываемая программа имеет пользовательский графический интерфейс для удобного взаимодействия с программой. Для реализации графического пользовательского интерфейса используется библиотека PyQT. Программа разработана на языке Python с использованием библиотек NumPy и Matplotlib. Язык был выбран по критерию легкости взаимодействия с ним и большого количества поддержки от сообщества в интернете.

ТЕОРИЯ

Модуляция – это процесс изменения каких-либо параметров несущего сигнала под действием информационного потока. Данный термин обычно применяют для аналоговых сигналов. Применительно к цифровым сигналам существует другой термин "манипуляция", однако его часто заменяют все тем же словом "модуляция" подразумевая, что речь идет о цифровых сигналах.

При фазовой манипуляции (ФМн, англ. PSK – phase shift keying) каждому цифровому символу сопоставляется своя начальная фаза несущего сигнала при неизменной амплитуде. Данный вид манипуляции наиболее сложен в реализации, но и наиболее помехоустойчив по сравнению с двумя другими видами манипуляции.

На рисунке 1 приведен график двоичной бинарной последовательности нулей и единиц и, соответствующий ему, график фазо-манипулированного сигнала. Низкому уровню бинарного сигнала сопоставляется начальная фаза 180 градусов, высокому уровню – фаза 0 градусов несущего сигнала синусоидального типа.

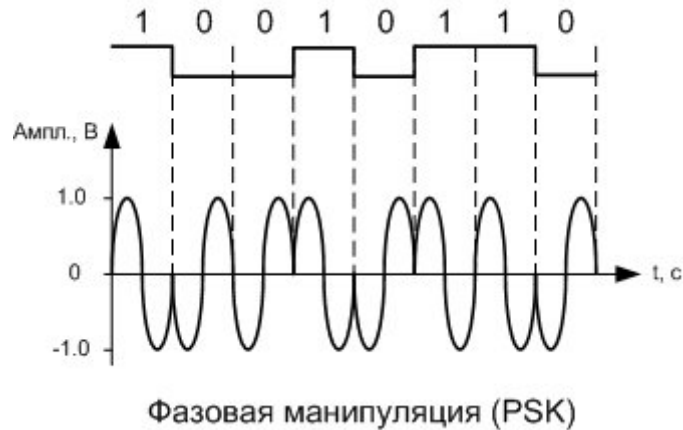


Рисунок 1 – Визуализация фазовой манипуляции

Данный тип модуляции называется двоичной фазовой манипуляцией (BPSK, binary phase shift keying), где «двоичный» относится к использованию двух фазовых смещений (одно для логической единицы и одно для логического нуля).

Сигнальное созвездие для данного вида манипуляции представлено на рисунке 2, на котором наблюдается 2 области, вокруг которых сконцентрированы точки. Эти области находятся на одной прямой и противоположны по фазе.

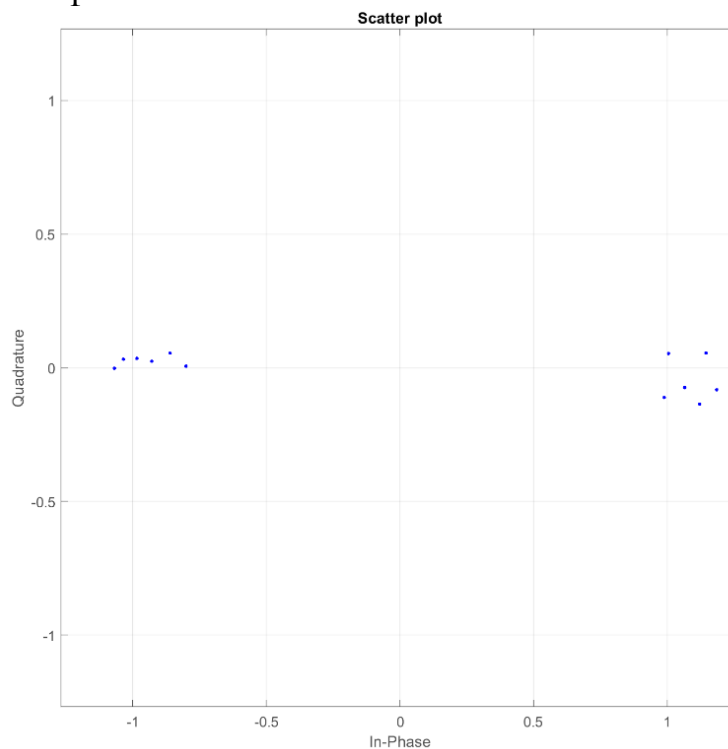


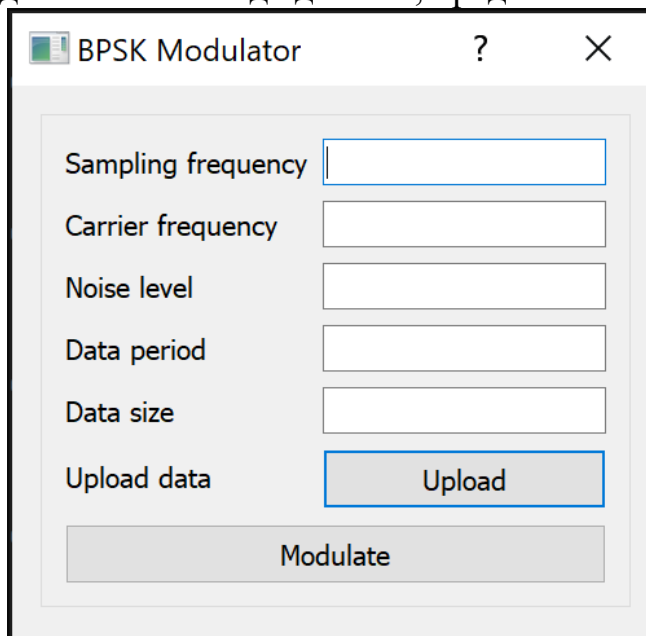
Рисунок 2 – Сигнальное созвездие BPSK

Чтобы произвести модуляцию сигнала необходимо создать несущую частоту, фаза которой будет меняться в зависимости от входных данных. Также необходимо выбрать интервал времени, который будет отвечать за кодирование одного бита информации. Далее в зависимости от значения бита информации фаза несущего сигнала остается неизменной на длину интервала одного бита, либо меняется на противоположную. Таким образом формируется модулированный сигнал.

Для демодуляции сигнала модулированный сигнал умножается поэлементно на несущую частоту, а затем пропускается через КИХ фильтр низких частот для удаления из сигнала несущей частоты. Таким образом сигнал будет выглядеть примерно как цифровой сигнал с данными. Его нужно будет пропустить через цифровой компаратор, для создания финального сигнала данных. После чего можно восстановить исходные данные в виде байт, беря значения сигнала начиная со значения интервала одного бита деленного на пополам и с шагом в значение этого интервала.

АЛГОРИТМ РАБОТЫ ПРОГРАММЫ BPSK МОДУЛЯЦИИ И ДЕМОДУЛЯЦИИ

Чтобы установить и запустить программу, необходимо исполнить в консоли команды, прописанные в файле «README.txt». Команды прописаны для ОС семейства Windows. После запуска программы пользователю выдается окно ввода данных, представленное на рисунке 3.



The image shows a software window titled "BPSK Modulator". It contains several input fields for configuration: "Sampling frequency", "Carrier frequency", "Noise level", "Data period", and "Data size". Below these is an "Upload data" label with an "Upload" button. At the bottom of the window is a large "Modulate" button.

Рисунок 3 – Пользовательский интерфейс для работы с программой

Интерфейс состоит из полей, в которые вводятся параметры для создаваемого сигнала, а также кнопки «Upload», которая позволяет загрузить файл любого формата. Бинарная составляющая выбранного файла будет использоваться как данные для модуляции. Частота дискретизации и несущая частота задается в герцах, уровень шума в

соотношении сигнал-шум, период данных задается в отсчетах, размер данных задается в количестве байт, которые необходимо использовать для модуляции.

После нажатия кнопки «Modulate» программа обработает входные данные и после построит график модулированного сигнала, который представлен на рисунке 4. Также программа сохранит данные о сигнале в файл «modulated_signal.json» в формате json, который представлен на рисунке 5.

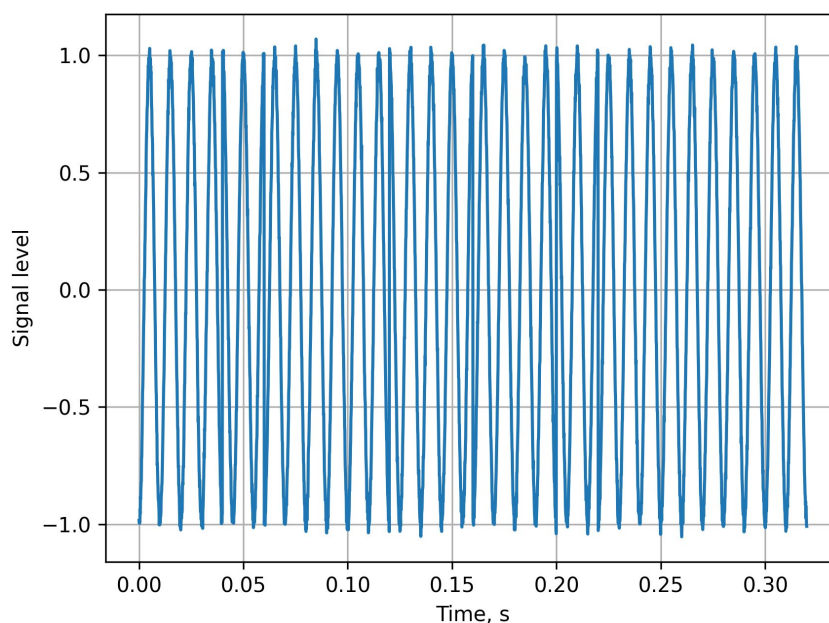


Рисунок 4 – Модулированный сигнал

```
"sampling_freq": 10000,  
"carrier_freq": 100,  
"data_period": 200,  
"modulated_signal": [  
  -0.9837015740882226,  
  -0.9938366982500275,  
  -0.9958843583650434,  
  -0.9880073978092866,  
  -0.9922860888493787,  
  -0.934807924251651,  
  -0.954113209057441,  
  -0.8820267215648343,  
  -0.8920939932320023,  
  -0.8462987022075066,  
  -0.8227181056093362,  
  -0.8163562575223406,  
  -0.7202789062238503,  
  -0.7139680763296156,  
  -0.6413806025317901,  
  -0.5896975007659877,
```

Рисунок 5 – Выходные данные модуляции

После проведения модуляции откроется окно для демодуляции, которое представлено на рисунке 6. Чтобы попасть в него, не проводя модуляцию, необходимо просто закрыть окно с модуляцией.

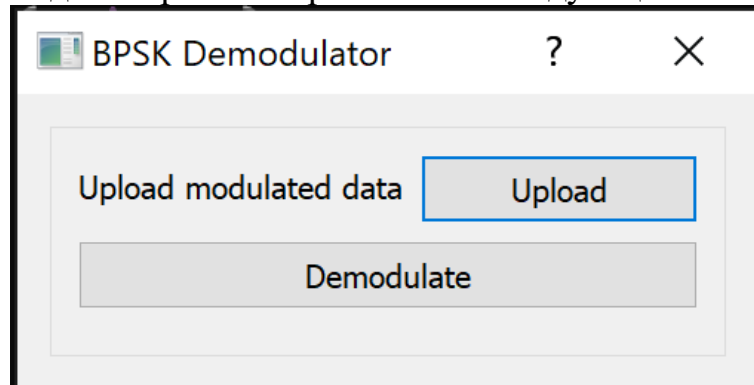


Рисунок 6 – Пользовательский интерфейс для демодуляции

Для демодуляции сигнала необходимо загрузить json файл с модулированным сигналом с помощью нажатия кнопки «Upload». Затем по нажатию кнопки «Demodulate» программа считывает данные из файла и производит демодуляцию. Результат демодуляции в виде графика представлен на рисунке 7.

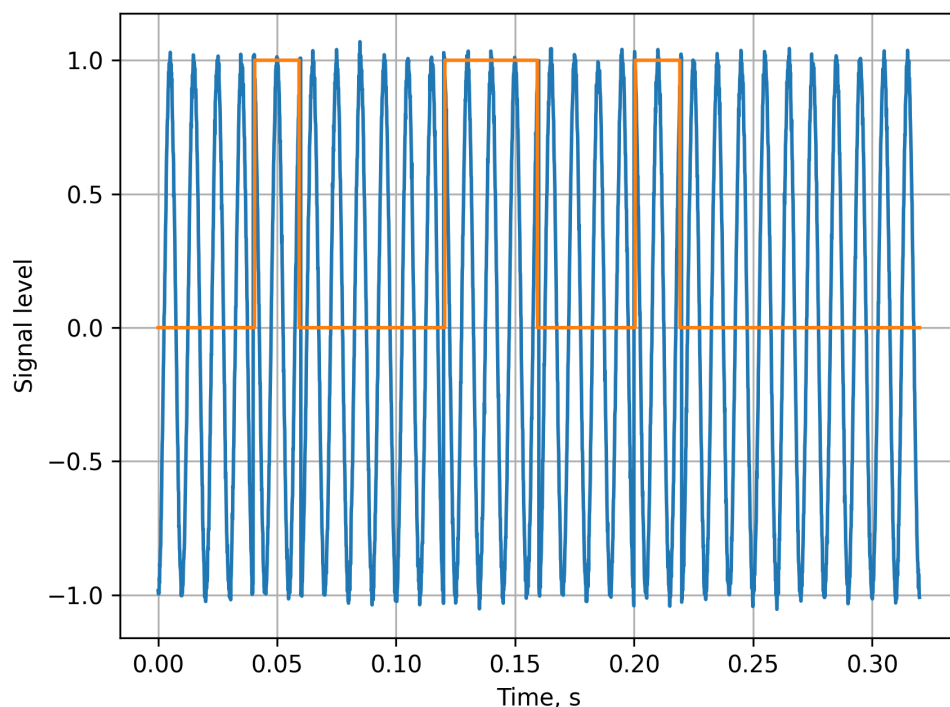


Рисунок 7 – Сигнал с демодулированными данными

После чего программа сохранит демодулированные данные в файл «demodulated_data.bin», содержание которого представлено на рисунке 8. Данные совпадают с исходными.

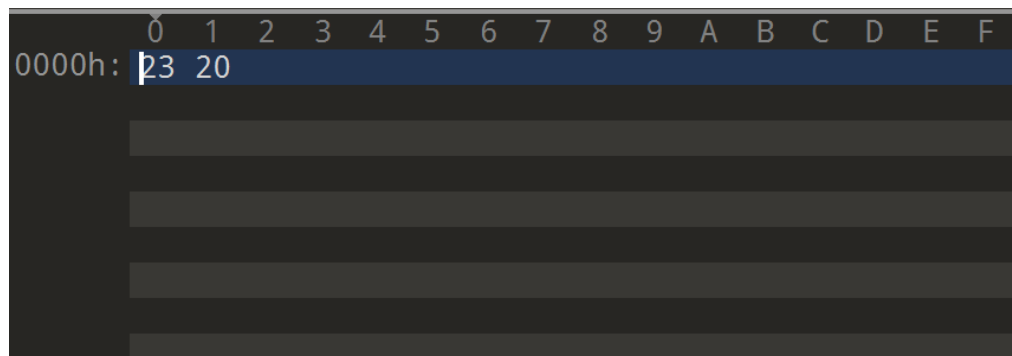


Рисунок 8 – Данные, полученные после демодуляции

ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ BPSK МОДУЛЯЦИИ И ДЕМОДУЛЯЦИИ

На вход программе подается набор данных, представленный на рисунке 9.

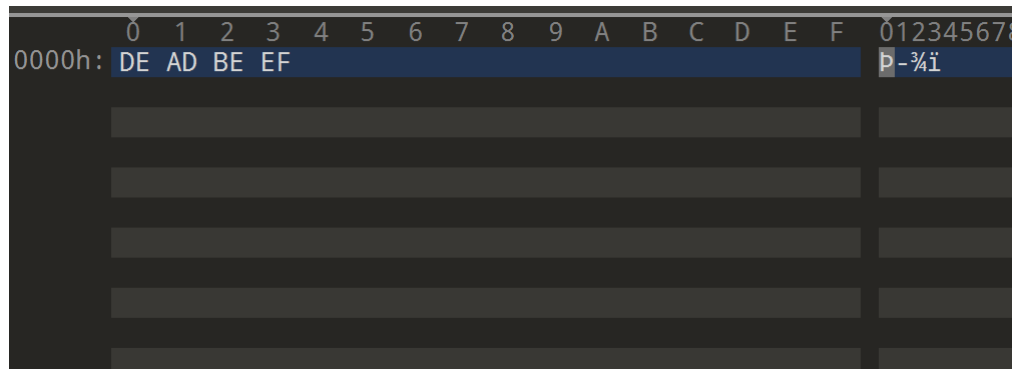


Рисунок 9 – Входные данные для модуляции

Заполняем параметры сигнала и подаем на вход файл

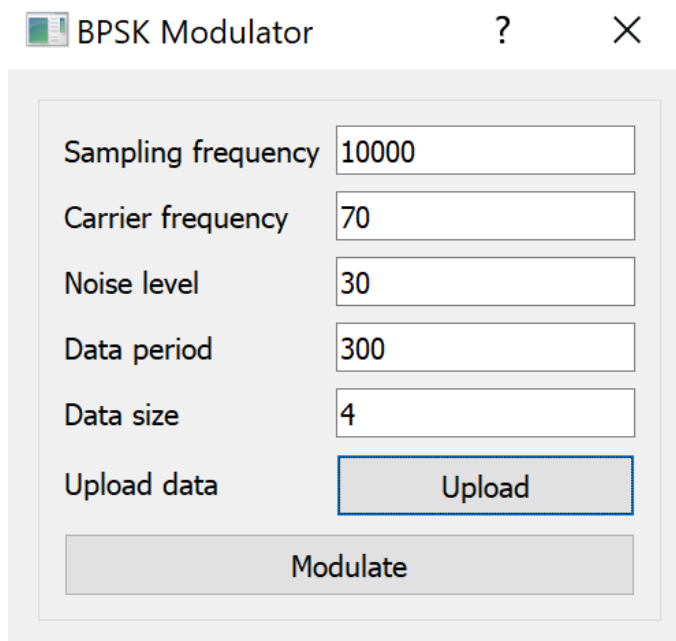


Рисунок 10 – Заполнение параметров сигнала

Получаем график модулированного сигнала и json файл с его данными

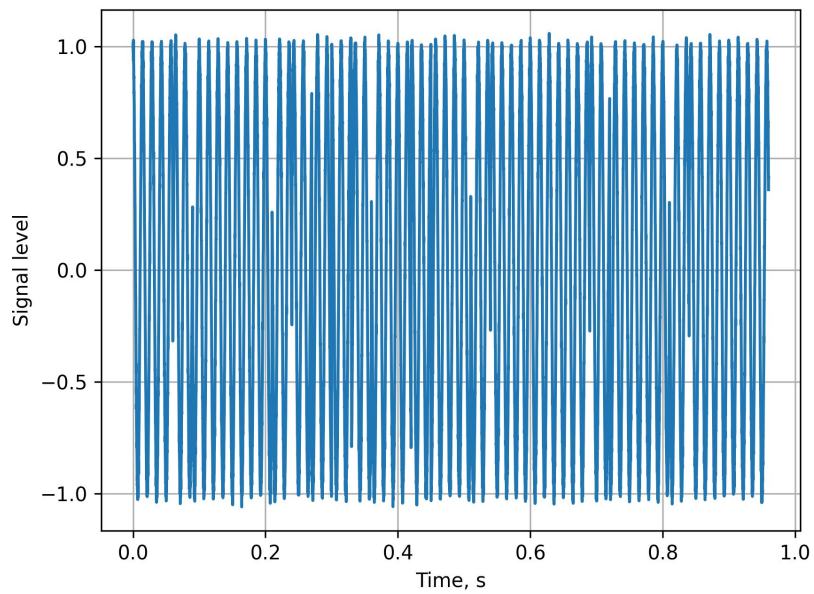


Рисунок 11 – Модулированный сигнал

```
{
  "sampling_freq": 10000,
  "carrier_freq": 70,
  "data_period": 300,
  "modulated_signal": [
    1.0130185210374751,
    1.0057940331177906,
    0.9733850384832017,
    1.0120701008154,
    0.9390679143224318,
    0.9942576514229153,
    1.0282235618834599,
    0.9576167006411463,
```

Рисунок 12 – Данные модулированного сигнала

После чего подаем json файл с данными модулированного сигнала на вход в демодулятор и получаем график демодулированного сигнала, а также данные, которые подавались на вход

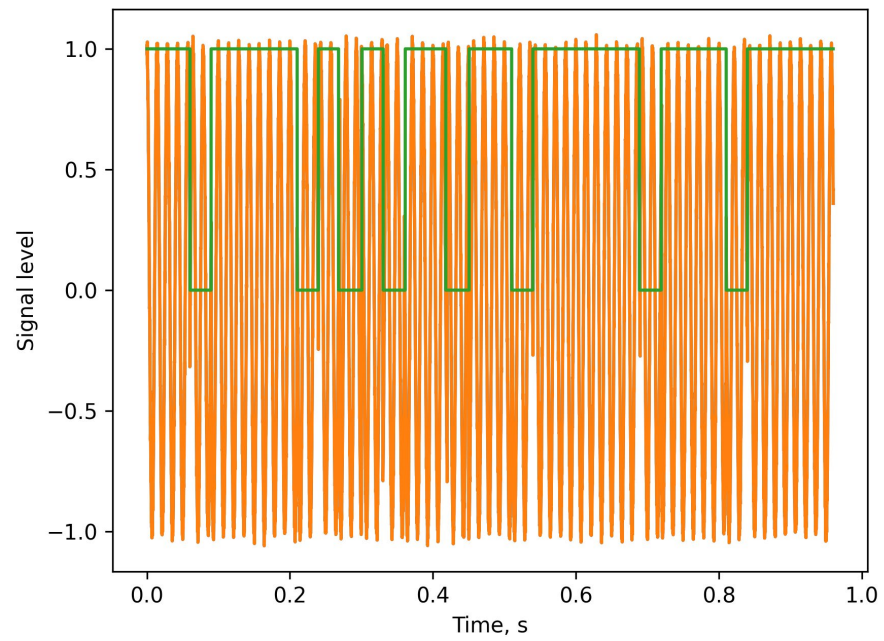


Рисунок 13 – Модулированный и демодулированный сигнал

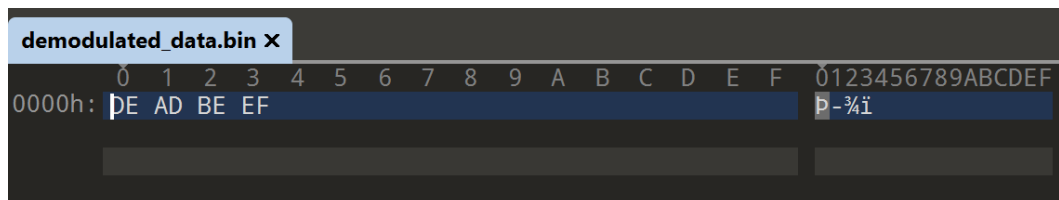


Рисунок 14 – Данные, полученные после демодуляции

ПРИЛОЖЕНИЕ

Исходный код

Файл main.py

```
import sys

import PyQt5.QtWidgets as pq

from demodulation import DemodulationWindow
from modulation import ModulationWindow


class MainWindow(pq.QMainWindow):
    def __init__(self):
        super().__init__()

        modulation_window = ModulationWindow()
        modulation_window.exec()

        demodulation_window = DemodulationWindow()
        demodulation_window.exec()

    def main() -> None:
        app = pq.QApplication(sys.argv)

        window = MainWindow()
        window.show()
        window.close()

        sys.exit(app.exec())

if __name__ == '__main__':
    main()
```

Файл modulation.py

```
import math
import json

import numpy as np
import PyQt5.QtWidgets as pq

from dataclasses import dataclass
from matplotlib import pyplot as plt

from extensions import byte_to_bin, bin_to_bpsk, BITS_IN_BYTE

plt.rcParams['figure.dpi'] = 300

@dataclass
class SourceSignal:
    sampling_freq: int
    carrier_freq: int
    noise_level: int
    data_period: int
    signal_length: float
```

```
time_points: np.ndarray
data_size: int
data: bytes
```

```
@classmethod
```

```
def raw_to_source_signal(
```

```
    cls,
    sampling_freq_i: str,
    carrier_freq_i: str,
    noise_level_i: str,
    data_period_i: str,
    data_size_i: str,
    data: bytes
```

```
) -> "SourceSignal":
```

```
    sampling_freq: int = int(sampling_freq_i)
    carrier_freq: int = int(carrier_freq_i)
    noise_level: int = int(noise_level_i)
    data_period: int = int(data_period_i)
    data_size: int = int(data_size_i)
    signal_length: float = data_period * data_size * BITS_IN_BYTE / sampling_freq
    time_points: np.ndarray = np.arange(0, signal_length, 1 / sampling_freq)
```

```
    return cls(
```

```
        sampling_freq=sampling_freq,
        carrier_freq=carrier_freq,
        noise_level=noise_level,
        data_period=data_period,
        signal_length=signal_length,
        time_points=time_points,
        data_size=data_size,
        data=data[:data_size]
```

```
)
```

```
def save_modulated_signal_to_file(self, modulated_signal: np.ndarray) -> None:
```

```
    output_file: dict[str, any] = {
        "sampling_freq": self.sampling_freq,
        "carrier_freq": self.carrier_freq,
        "data_period": self.data_period,
        "modulated_signal": modulated_signal.tolist()
    }
```

```
    json_output_file: str = json.dumps(output_file, indent=4)
```

```
    with open("modulated_signal.json", "wt") as modulated_signal_file:
        modulated_signal_file.write(json_output_file)
```

```
def add_noise(self, modulated_signal: np.ndarray) -> np.ndarray:
```

```
    modulated_signal_power: np.ndarray = modulated_signal ** 2
    modulated_signal_avg_power: np.ndarray = np.mean(modulated_signal_power)
    modulated_signal_avg_power_db: int = 10 * np.log10(modulated_signal_avg_power)
    noise_level_db: int = modulated_signal_avg_power_db - self.noise_level
    noise_level_avr_power: float = 10 ** (noise_level_db / 10)
    noise_level_power: np.float64 = np.sqrt(noise_level_avr_power)
```

```
    noise_signal: np.ndarray = np.random.normal(0, noise_level_power, modulated_signal.size)
```

```
    return modulated_signal + noise_signal
```

```
class ModulationWindow(pq.QDialog):
```

```
    def __init__(self):
```

```
        super(ModulationWindow, self).__init__()
        self.setWindowTitle("BPSK Modulator")
```

```

self.setGeometry(0, 0, 500, 400)

self.modulation_group: pq.QGroupBox = pq.QGroupBox()

self.sampling_freq: pq.QLineEdit = pq.QLineEdit()
self.carrier_freq: pq.QLineEdit = pq.QLineEdit()
self.noise_level: pq.QLineEdit = pq.QLineEdit()
self.data_period: pq.QLineEdit = pq.QLineEdit()
self.data_size: pq.QLineEdit = pq.QLineEdit()

self.upload_source_data_button: pq.QPushButton = pq.QPushButton("Upload")
self.upload_source_data_button.clicked.connect(self.get_source_signal_raw_data)

self.modulate_button: pq.QPushButton = pq.QPushButton("Modulate", self)
self.modulate_button.clicked.connect(self.modulate)

self.create_form()

main_layout: pq.QVBoxLayout = pq.QVBoxLayout()
main_layout.addWidget(self.modulation_group)

self.setLayout(main_layout)

self.input_data: bytes = bytes()
self.demodulation_window = None

def create_form(self) -> None:
    layout = pq.QFormLayout()

    layout.addRow(pq.QLabel("Sampling frequency"), self.sampling_freq)
    layout.addRow(pq.QLabel("Carrier frequency"), self.carrier_freq)
    layout.addRow(pq.QLabel("Noise level"), self.noise_level)
    layout.addRow(pq.QLabel("Data period"), self.data_period)
    layout.addRow(pq.QLabel("Data size"), self.data_size)
    layout.addRow(pq.QLabel("Upload data"), self.upload_source_data_button)
    layout.addRow(self.modulate_button)

    self.modulation_group.setLayout(layout)

def get_source_signal_raw_data(self) -> None:
    filename: str = pq.QFileDialog.getOpenFileName(self, caption='Choose File')[0]
    with open(filename, "rb") as input_data_file:
        self.input_data = input_data_file.read()

def modulate(self) -> None:
    source_signal: SourceSignal = SourceSignal.raw_to_source_signal(
        self.sampling_freq.text(),
        self.carrier_freq.text(),
        self.noise_level.text(),
        self.data_period.text(),
        self.data_size.text(),
        self.input_data
    )

    cos_points: list[float] = [2 * math.pi * x * source_signal.carrier_freq for x in source_signal.time_points]
    carrier_signal: np.ndarray = np.cos(cos_points)

    bpsk_data_values: list[int] = list()
    for byte in source_signal.data:
        binary: list[int] = byte_to_bin(byte)
        bpsk_binary: list[int] = bin_to_bpsk(binary)
        bpsk_data_values.extend(bpsk_binary)

```

```

bpsk_data_signal: np.ndarray = np.repeat(bpsk_data_values, source_signal.data_period)

modulated_signal: np.ndarray = np.multiply(carrier_signal, bpsk_data_signal)

modulated_signal = source_signal.add_noise(modulated_signal)

plt.plot(source_signal.time_points, modulated_signal)
plt.xlabel("Time, s")
plt.ylabel("Signal level")
plt.grid()
plt.ion()
plt.show()

source_signal.save_modulated_signal_to_file(modulated_signal)
self.close()

```

Файл demodulation.py

```

import json
import math
from dataclasses import dataclass

import PyQt5.QtWidgets as pq
import numpy as np
from matplotlib import pyplot as plt

from extensions import low_pass_filter, BITS_IN_BYTE

@dataclass
class ModulatedSignal:
    sampling_freq: int
    carrier_freq: int
    data_period: int
    data: np.ndarray

    @classmethod
    def data_to_modulated_signal(cls, file_data: str) -> "ModulatedSignal":
        parsed_modulated_signal: dict[str, any] = json.loads(file_data)
        return cls(
            sampling_freq=parsed_modulated_signal["sampling_freq"],
            carrier_freq=parsed_modulated_signal["carrier_freq"],
            data_period=parsed_modulated_signal["data_period"],
            data=np.asarray(parsed_modulated_signal["modulated_signal"])
        )

class DemodulationWindow(pq.QDialog):
    def __init__(self):
        super(DemodulationWindow, self).__init__()
        self.setWindowTitle("BPSK Demodulator")

        self.setGeometry(0, 0, 500, 200)

        self.demodulation_group: pq.QGroupBox = pq.QGroupBox()

        self.upload_mod_data_button: pq.QPushButton = pq.QPushButton("Upload")
        self.upload_mod_data_button.clicked.connect(self.get_modulated_signal_data)

        self.demodulate_button: pq.QPushButton = pq.QPushButton("Demodulate", self)

```

```

self.demodulate_button.clicked.connect(self.demodulate)

self.create_form()

main_layout: pq.QVBoxLayout = pq.QVBoxLayout()
main_layout.addWidget(self.demodulation_group)

self.setLayout(main_layout)

self.input_data: str = str()

def create_form(self) -> None:
    layout = pq.QFormLayout()

    layout.addRow(pq.QLabel("Upload modulated data"), self.upload_mod_data_button)
    layout.addRow(self.demodulate_button)

    self.demodulation_group.setLayout(layout)

def get_modulated_signal_data(self) -> None:
    filename: str = pq.QFileDialog.getOpenFileName(self, caption='Choose File')[0]
    with open(filename, "rt") as modulated_data_file:
        self.input_data = modulated_data_file.read()

def demodulate(self):
    modulated_signal: ModulatedSignal = ModulatedSignal.data_to_modulated_signal(self.input_data)

    signal_length: float = modulated_signal.data.size / modulated_signal.sampling_freq
    time_points: np.ndarray = np.arange(0, signal_length, 1 / modulated_signal.sampling_freq)

    cos_points: list[float] = [2 * math.pi * x * modulated_signal.carrier_freq for x in time_points]
    carrier_signal: np.ndarray = np.cos(cos_points)

    tmp_signal: np.ndarray = np.multiply(modulated_signal.data, carrier_signal)
    demodulated_signal: np.ndarray = low_pass_filter(
        tmp_signal,
        modulated_signal.carrier_freq,
        modulated_signal.sampling_freq
    )
    digital_signal: np.ndarray = demodulated_signal > 0.1

    plt.plot(time_points, modulated_signal.data)
    plt.plot(time_points, digital_signal)
    plt.xlabel("Time, s")
    plt.ylabel("Signal level")
    plt.grid()
    plt.ion()
    plt.show()

    data_bits_raw: list[int] = list()
    for offset in range(modulated_signal.data_period//2, digital_signal.size,
modulated_signal.data_period):
        data_bits_raw.append(digital_signal[offset])

    data_bits: np.ndarray = np.asarray(data_bits_raw)

    data_bytes = np.split(data_bits, data_bits.size / BITS_IN_BYTE)
    result_bytes = bytes()
    for byte in data_bytes:
        result_bytes += int("".join(str(int(x)) for x in byte), 2).to_bytes(1, "little")

```

```
with open("demodulated_data.bin", "wb") as demodulated_data_file:  
    demodulated_data_file.write(result_bytes)
```

Файл extensions.py

```
import numpy as np
```

```
BITS_IN_BYTE: int = 8
```

```
def byte_to_bin(number: int) -> list[int]:  
    bin_number: str = bin(number)[2:]  
    binary: list[int] = [int(x) for x in bin_number]  
    zero_bits_len: int = BITS_IN_BYTE - len(bin_number)  
    for bit in range(zero_bits_len):  
        binary.insert(0, 0)  
  
    return binary
```

```
def bin_to_bpsk(data: list[int]) -> list[int]:  
    return [-1 if bit == 0 else bit for bit in data]
```

```
def low_pass_filter(signal: np.ndarray, band_limit: int, sampling_rate: int) -> np.ndarray:  
    cutoff_index: int = int(band_limit * signal.size / sampling_rate)  
    fft_signal: np.ndarray = np.fft.rfft(signal)  
    fft_signal[cutoff_index + 1:] = 0  
    return np.fft.irfft(fft_signal, n=signal.size).real
```