1) Цель работы

- разработать и отладить программу моделирования по заданному алгоритму

Вариант **III.9**
Квадратурная амплитудная модуляция:
f_0 **= 1800 Гц** — несущая частота
V_m **= 2400 Бод** — модуляционная скорость
V_i **= 12000 бит/с** — информационная скорость

q **= 32** – количество сигналов

2) Результат моделирования

При всех моделирование максимальное количество ошибок было равно **50**

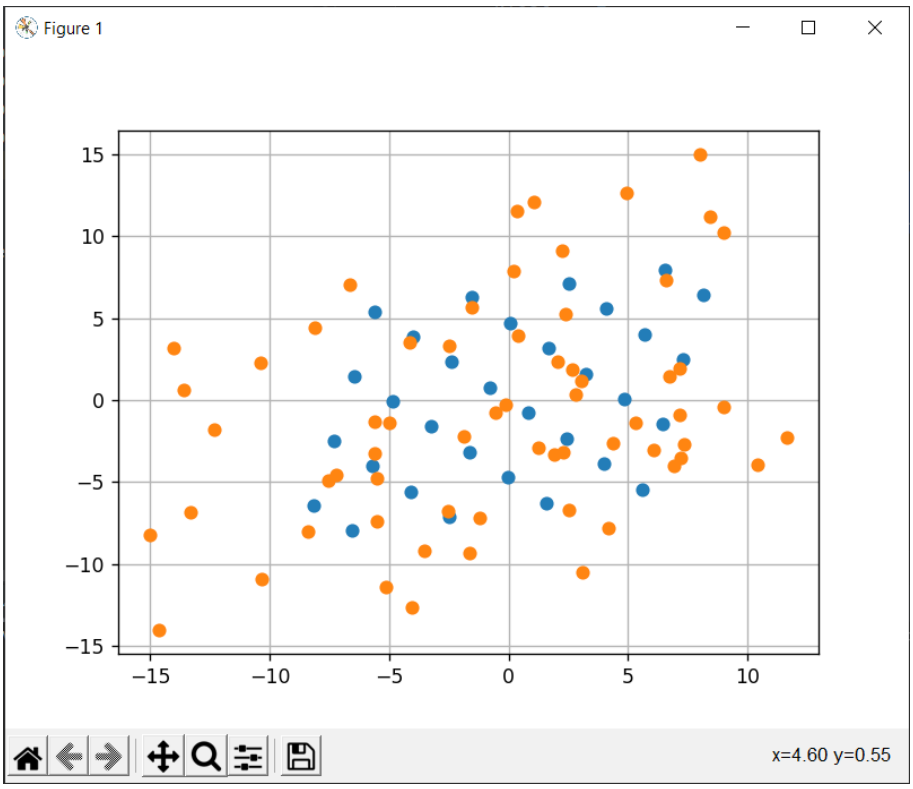

Рис. 1 — Моделирование при γ_{db} **= 0**

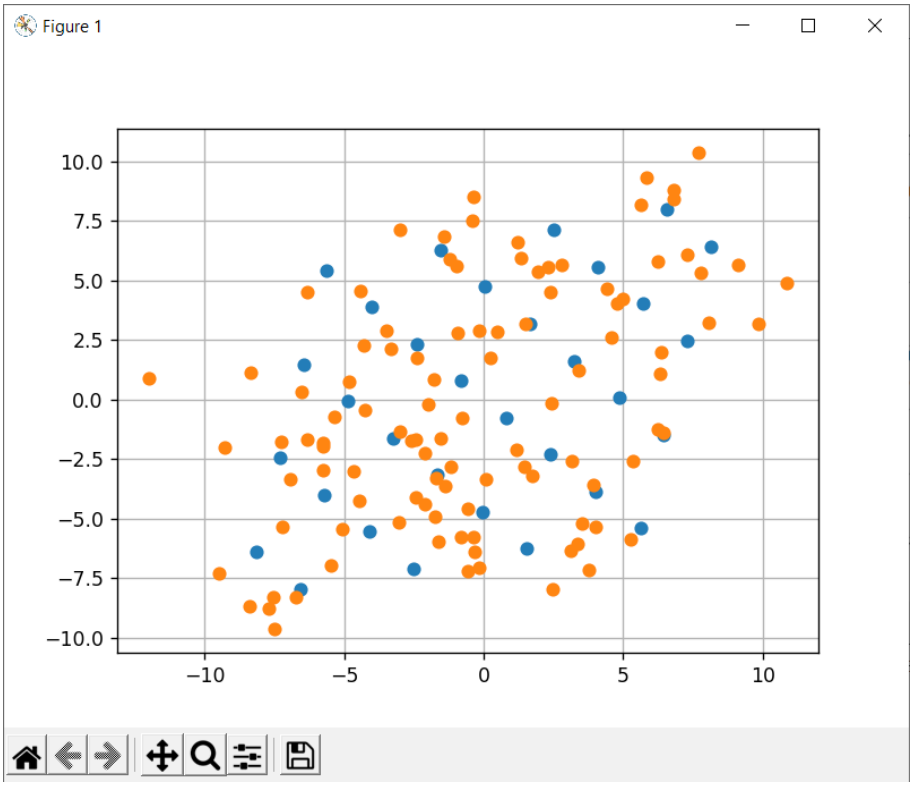

Рис. 2 — Моделирование при γ_{db} = **10**

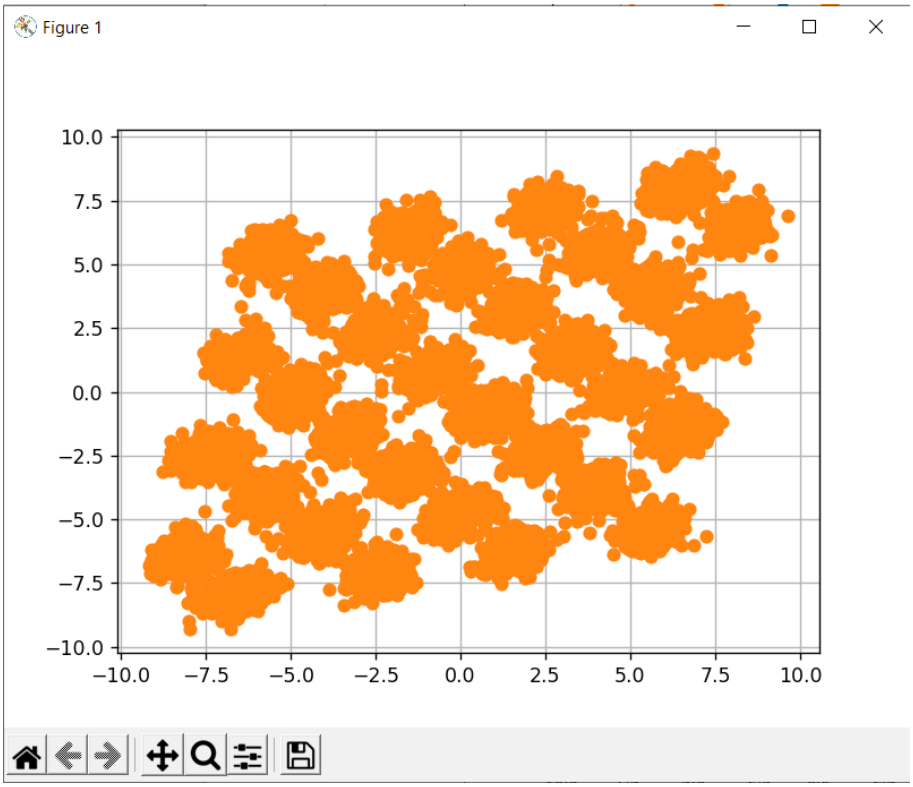

Рис. 3 — Моделирование при γ_{db} = **20**

3) Вычисление вероятность ошибки

Для квадратурно-амплитудной модуляции формула для вероятности ошибки выглядит следующим образом

$$P_e = \frac{4(\sqrt{q}-1)}{q} Q\left(\sqrt{\frac{3\overline{E}}{N_0}\frac{1}{q-1}}\right)\left(\sqrt{q}-(\sqrt{q}-1)Q\left(\sqrt{\frac{3\overline{E}}{N_0}\frac{1}{q-1}}\right)\right)$$

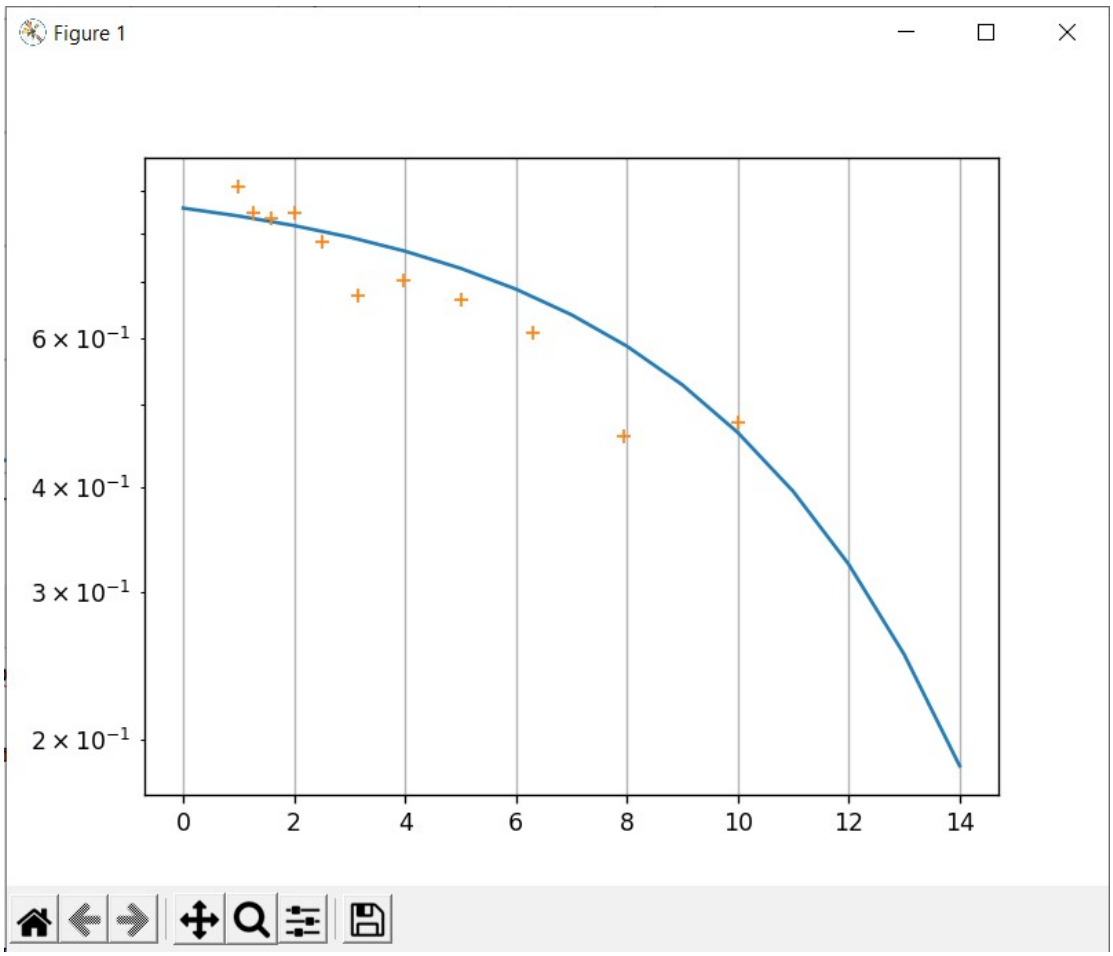

Рис. 4 — График зависимости $P_e(SNR)$ и результаты моделирования.

Теоретический расчёт частично совпал с моделированным результатом.

4) Вывод

В ходе лабораторной работы были:

- Промоделирована работа приемника дискретных сигналов в канале с аддитивным белым гауссовским шумом
- Рассчитаны вероятности ошибок в работе приемника при различных значениях SNR
- Данные вероятности ошибок были сравнены с полученными в ходе моделирования вероятностями ошибок

Листинг исходного кода на языке *Python*

```
import random

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

import O_L1 as L1
import O_L3 as L3


def doRandomStep(x, s, dt, T, f0, q, sij, sigma, NErrMax = 50):
    NErr = 0
    NTest = 0

    points = []

    while NErr < NErrMax:
        i = random.randint(0, q-1)
        r = []
        n = np.random.normal(loc=0, scale=sigma, size=len(s[i]))
        for ii in range(0, len(s[i])):
            r.append(s[i][ii] + n[ii])

        rij = L3.getBaseSij(x, [r], dt, T, f0)[0]
        i_ = -1
        minD = 1 << 30
        for ii in range(0, len(sij)):
            d = L3.evD(rij, sij[ii])
            if d < minD:
                minD = d
                i_ = ii

        if i != i_:
            NErr += 1
        points.append(rij)

        NTest += 1

    return NErr / NTest
```

```
def doRandom(x, s, dt, T, f0, q, E, sij):
    gammas = np.arange(0, 21, 4)
    ps = []

    for g in gammas:
        gt = pow(10, g / 10)
        N0 = E / gt
        sigma = np.sqrt(N0 / 2)
        ps.append(doRandomStep(x, s, dt, T, f0, q, sij, sigma))

    return [pow(10, g / 10) for g in gammas], ps

def qFun(x):
    return 0.5 - 0.5 * scipy.special.erf(x / np.sqrt(2))

def drawPr(q):
    dt = 0.1
    x = np.arange(0, 100, dt)
    p = []
    for i in x:
        qf = qFun(np.sqrt((3 * i) / (q - 1)))
        p.append((4 * (np.sqrt(q) - 1) / q)*qf * (np.sqrt(q) - (np.sqrt(q) - 1)*qf))
    L1.drawArrays(x, [p])



def QAM_L4(f0, Vmod, Vinf):
    T = 1 / Vmod
    q = pow(2, Vinf / Vmod)
    s1, s2, A = L1.getSs(q)

    dt = 1 / (f0 * 100)
    x, s = L1.getSignals(f0, T, dt, s1, s2)
    E = 0
    for i in s:
        E += pow(np.linalg.norm(i), 2)
    E /= q
    drawPr(q)
    sij = L3.getBaseSij(x, s, dt, T, f0)
    g, ps = doRandom(x, s, dt, T, f0, q, E, sij)
    L1.drawArrays(g, [ps], lineType='+')
    plt.show()

if __name__ == "__main__":
    QAM_L4(1800, 2400, 12000)
```