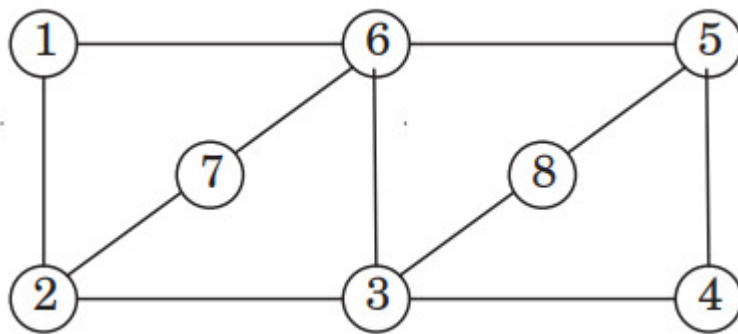


Цель работы: получение практических навыков оценки надежности вычислительных сетей.

Вариант задания: 17

Задан случайный граф  $G(X, Y, P)$ , где  $X = \{x_i\}$  – множество вершин,  $Y = \{(x_i, x_j)\}$  – множество ребер,  $P = \{p_i\}$  – множество вероятностей существования ребер. Вероятности существования ребер равны между собой и равны  $p$ . В ходе выполнения лабораторной работы необходимо выполнить следующие действия.

1. Применение простого имитационного моделирования.
2. Уменьшение множества рассматриваемых при имитационном моделировании графов.





## Описание программы

Программа выполняет вычисление вероятности наличия пути из 2 в 5 двумя способами: простым имитационным моделированием при котором генерируются вектор с помощью распределения Бернулли и ускоренном варианте данного моделирования, при котором сразу определяются определённые варианты

Для

Результаты работы программы:

Для формулы: 0 , 0.00496871272 , 0.03869566976 , 0.12333956724 , 0.26599702528 , 0.453125 , 0.65269573632 , 0.82530263116 , 0.94062116864 , 0.99175836888 , 1

Для первого варианта:

0 , 0.00498578 , 0.03856 , 0.123412 , 0.265884 , 0.452656 , 0.652899 , 0.825302 , 0.940737 , 0.991737 , 1

Для второго варианта:

0 , 0.00493111 , 0.0387289 , 0.123797 , 0.266544 , 0.452762 , 0.653028 , 0.825388 , 0.940592 , 0.991791 , 1

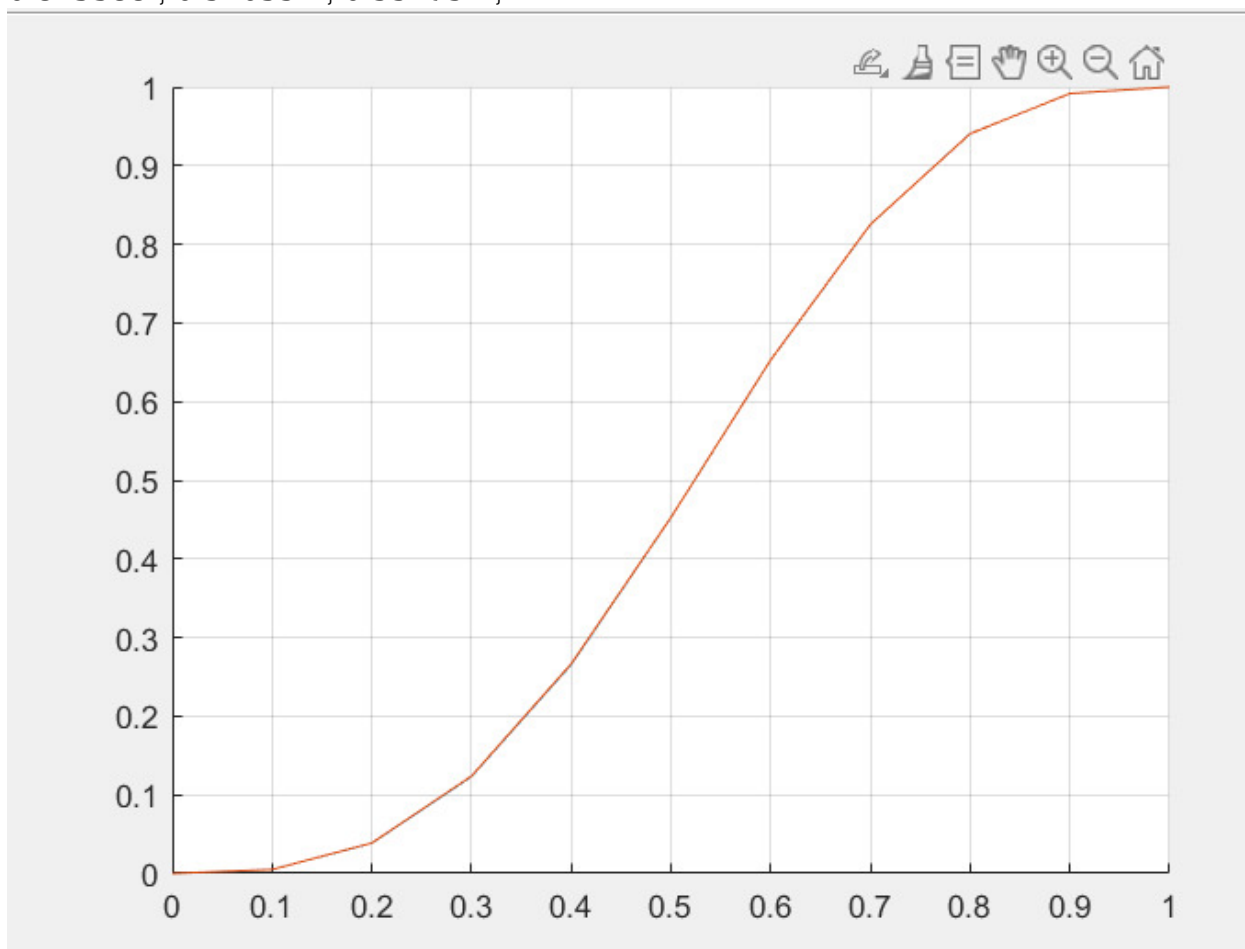


Рисунок 1 График вероятности для первого варианта

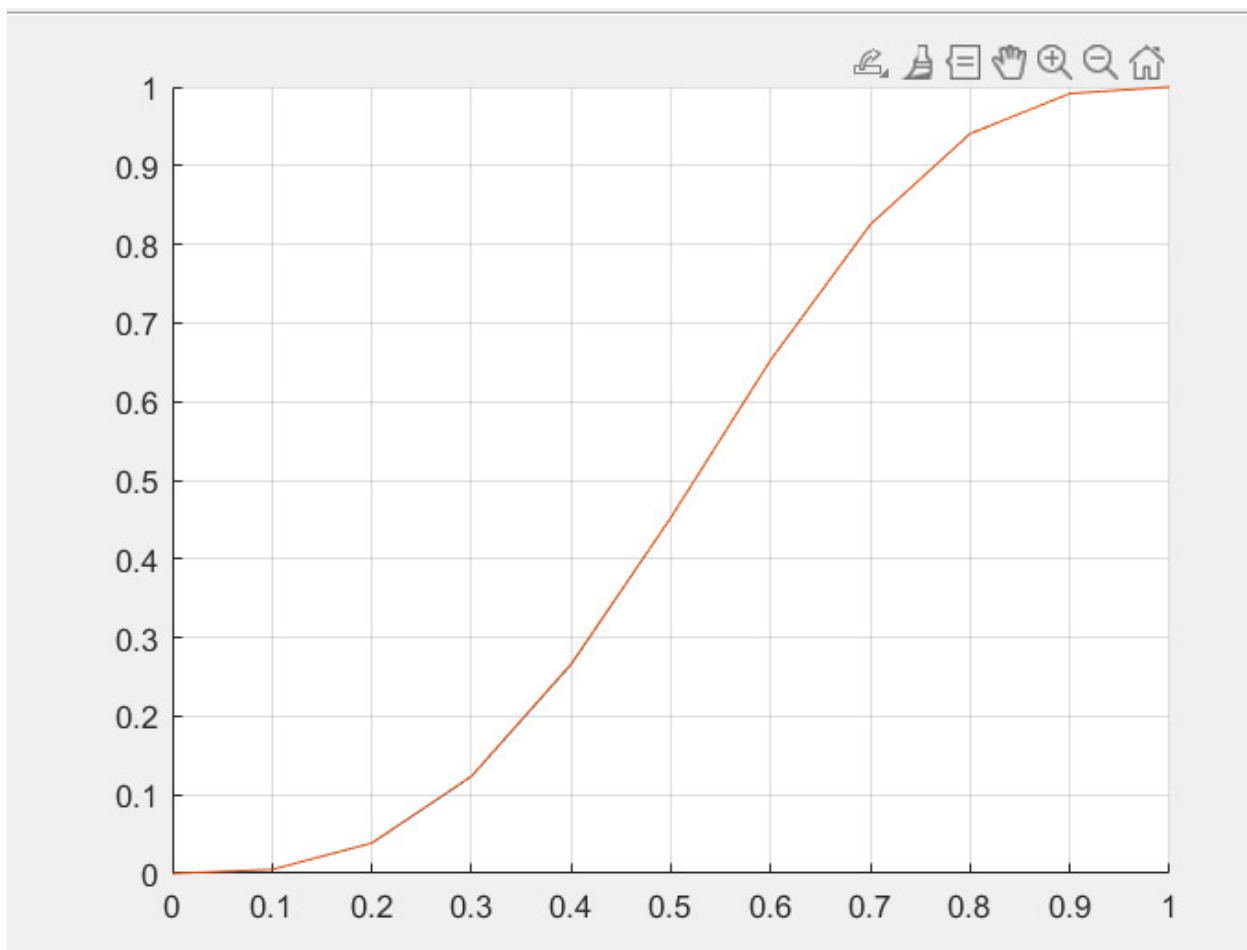


Рисунок 2 График вероятности для второго варианта

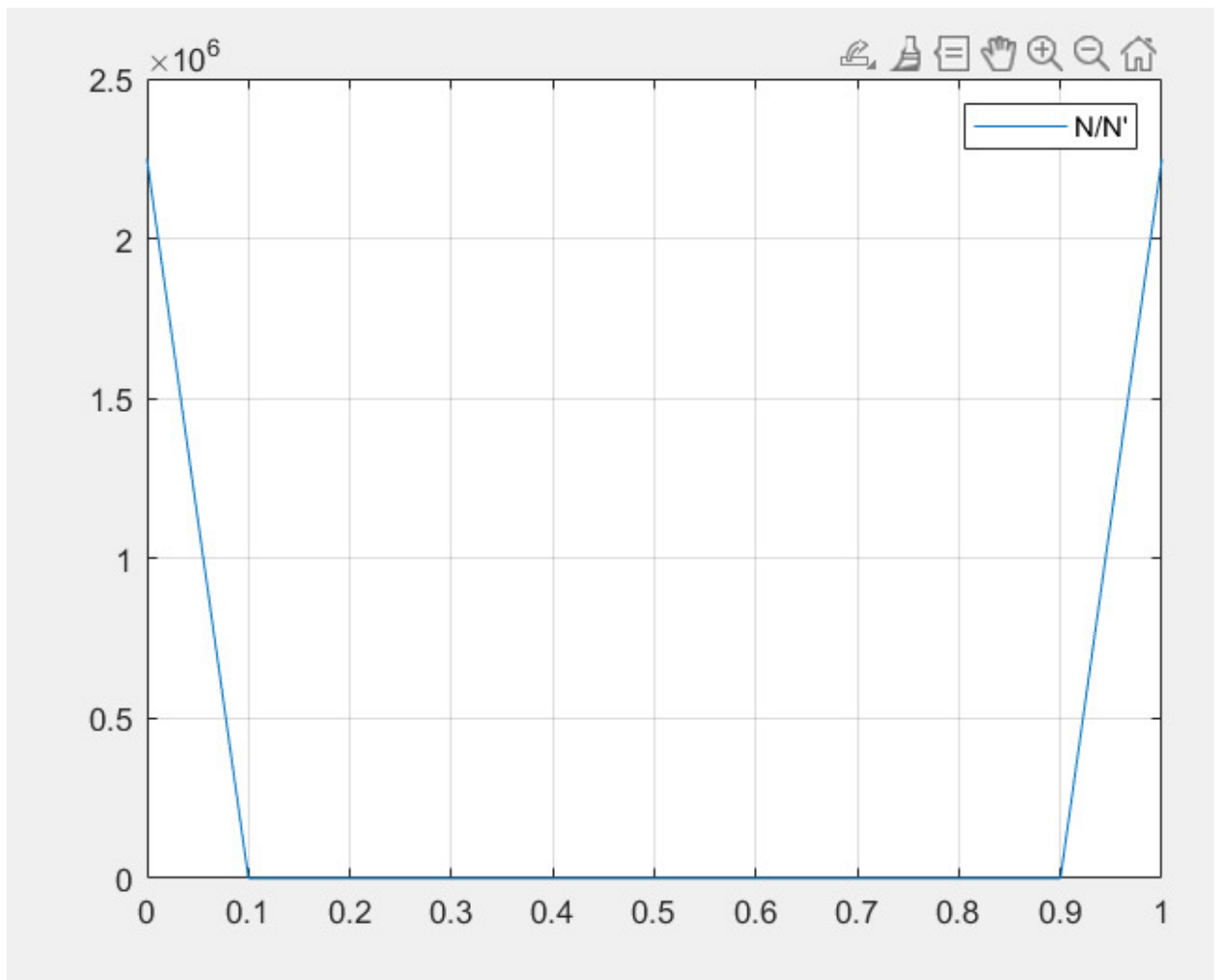


Рисунок 3 График выигрыша по количеству попыток для 2 варианта

## Выводы

Результаты первого, второго варианта и прошлой лабораторной работы сошлись с точностью , что свидетельствует о правильности расчётов и точности вычисления при использовании имитационного моделирования

## Текст программы

```
#include <iostream>
#include <vector>
#include <cmath>
#include <queue>
#include <bitset>
#include <iomanip>

struct edge {
    int a;
    int b;
};

bool bfs(std::vector<edge> &forcer);

int main() {
    std::random_device rd;
    std::mt19937_64 generator = std::mt19937_64(rd());
    std::vector<double> v0;
    std::vector<double> v1;
    std::vector<double> v2;
    std::vector<bool> flags(11, false);
    std::vector<edge> forcer;
    std::vector<edge> edges{
        {0, 1},
        {0, 5},
        {1, 2},
        {1, 6},
        {2, 3},
        {2, 5},
        {2, 7},
        {3, 4},
        {4, 5},
        {4, 7},
        {5, 6},
    };

    double epsilon = 0.001;
    double p = 0;
    for (; p <= 1; p += 0.1) {
        std::bernoulli_distribution random = std::bernoulli_distribution(p);
        double N = 2.25 / (epsilon * epsilon);
        int number = ceil(N);
        int calc = 0;
        for (int i = 0; i < number; i++) {
            for (int j = 0; j < 11; j++) {
                random(generator) ? forcer.push_back(edges[j]) : void();
            }
            calc += bfs(forcer);
            forcer.clear();
        }
        v0.push_back((double) calc / number);
        calc = 0;
        for (int i = 0; i < number; i++) {
            for (int j = 0; j < 11; j++) {
                random(generator) ? forcer.push_back(edges[j]) : void();
            }
            int p_pow = forcer.size();
            if (p_pow < 3 || p_pow > 8) {
                calc += p_pow > 8;
            } else {
                calc += bfs(forcer);
            }
            forcer.clear();
        }
    }
}
```

```

    }
    v1.push_back((double) calc / number);
}
for (double v: v0) {
    std::cout << v << " , ";
}
std::cout << std::endl;
for (double v: v1) {
    std::cout << v << " , ";
}
std::cout << std::endl;
for (double v: v2) {
    std::cout << v << " , ";
}
std::cout << std::endl;
return 0;
}

bool bfs(std::vector<edge> &forcer) {
    std::vector<std::vector<bool>> matrix;
    std::vector<bool> used(8, false);
    for (int i = 0; i < 8; i++) {
        matrix.emplace_back(8, false);
    }
    for (edge e: forcer) {
        matrix[e.a][e.b] = true;
        matrix[e.b][e.a] = true;
    }
    std::queue<int> queue;
    queue.push(1);
    while (!queue.empty()) {
        int v = queue.front();
        queue.pop();
        for (int i = 0; i < 8; i++) {
            if (matrix[v][i] && !used[i]) {
                used[i] = true;
                queue.push(i);
            }
        }
    }
    return used[4];
}

```