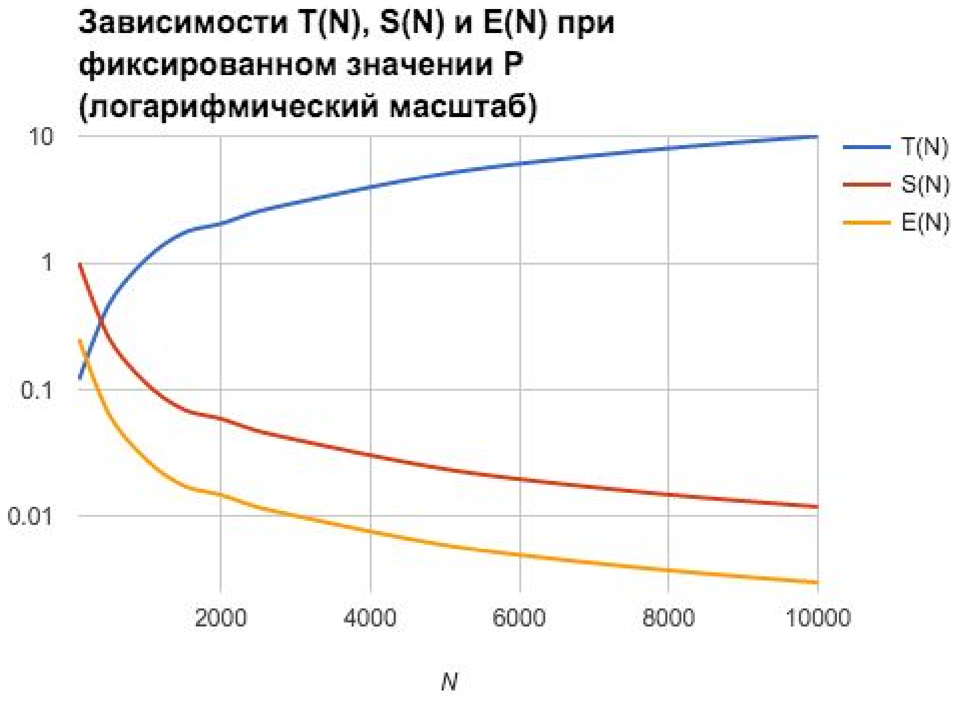
# Задание #2. Методы Монте-Карло

Горемыкин Александр Олегович, ifresh.wp@gmail.com

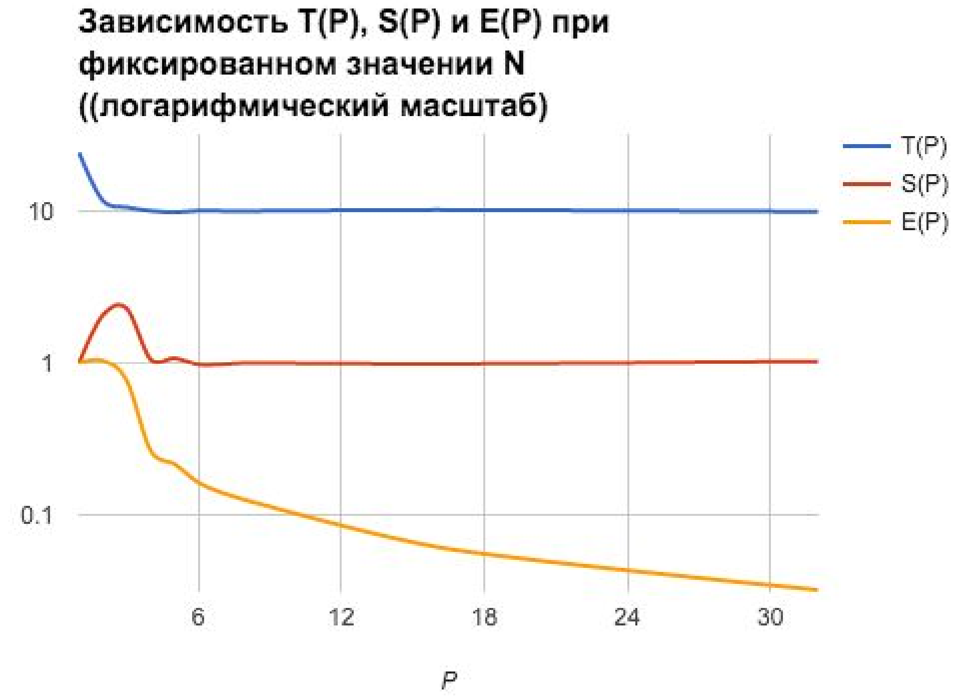
---

**ЗАМЕЧАНИЕ: тесты проводились на локальной системе:**
**Core i5@2.6 (2 cores, hyper-threading up to 4), 8 GB RAM**

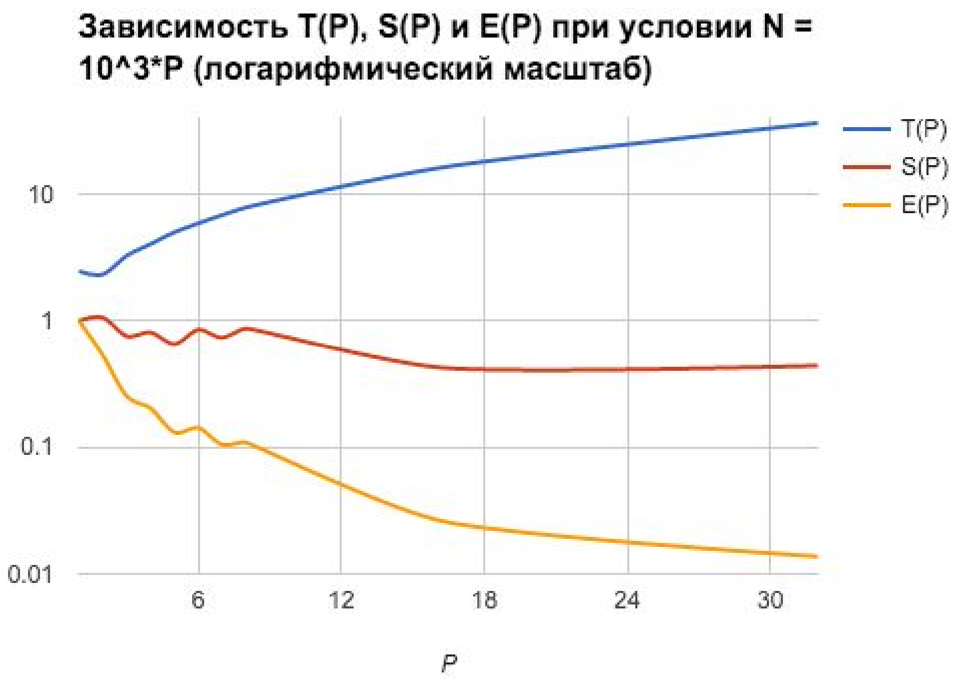
---



Зависимости T(N), S(N) и E(N) при фиксированном значении P (логарифмический масштаб)



Зависимость T(P), S(P) и E(P) при фиксированном значении N ((логарифмический масштаб)



-----------------------------------------------------------------------------------------------------------------

**github:** https://github.com/sanllier/Practice_Ershov/tree/master/01

```
#include <fstream>
#include <ctime>

#include "helpers.h"
#include "parparser.h"

//--------------------------------------------------------------------------------

struct World {
	long leftLimit;
	long rightLimit;
	float probability;

	World()
		: leftLimit(0)
		, rightLimit(0)
		, probability(0.0f)
	{}
};

class Particle {
public:
	struct WalkResult {
		float time;
		bool rightLimitReached;

		WalkResult()
			: time(0.0f)
			, rightLimitReached(false)
		{}
	};
```

```
private:
        World m_world;

        float m_lifeTime;
        long m_steps;

public:
        Particle(const World& world)
                : m_world(world)
                , m_lifeTime(0.0f)
                , m_steps(0)
        {}

        WalkResult walkFrom(long from) const {
                WalkResult temp;
                long currentPosition = from;
                float rValue = 0.0f;

                while (currentPosition > m_world.leftLimit && currentPosition <
m_world.rightLimit) {
                        float rValue = rand() / float(RAND_MAX);
                        currentPosition += rValue < m_world.probability ? 1 : -1;
                        ++temp.time;
                }

                temp.rightLimitReached = currentPosition == m_world.rightLimit;
                return temp;
        }
};

//--------------------------------------------------------------------------------

int main(int argc, char** argv) {

        parparser parser(argc, argv);

        World world;

        world.leftLimit = parser.get("a").asLong();
        world.rightLimit = parser.get("b").asLong();
        long inititalPosition = parser.get("x").asLong();
        world.probability = parser.get("p").asFloat();
        long particlesNumber = parser.get("N").asLong();
        string outFile = parser.get("o").asString();
        string statFile = parser.get("s").asString();

        srand(time(0));

        //--------------------------------------------------------------------------------

        MPICHECK(MPI_Init(&argc, &argv));
        //--------------------------------------------------------------------------------

        int commSize = 0;
        int rank = 0;
        MPICHECK(MPI_Comm_size(MPI_COMM_WORLD, &commSize));
        MPICHECK(MPI_Comm_rank(MPI_COMM_WORLD, &rank));


        long localIterationsNumber = particlesNumber / long(commSize);
        if (rank < particlesNumber % commSize) ++localIterationsNumber;

        Particle particle(world);
        long totalTime = 0.0f;
        long rightLimitReachedTimes = 0;

        const double startTime = MPI_Wtime();

        for (long i = 0; i < localIterationsNumber; ++i) {
                Particle::WalkResult tempRes = particle.walkFrom(inititalPosition);
                totalTime += tempRes.time;
                rightLimitReachedTimes += tempRes.rightLimitReached ? 1 : 0;
        }

        MPI_Barrier(MPI_COMM_WORLD);
        const double endTime = MPI_Wtime();

        //-------------------------------------------------------------------------------

        float buf[2] = {totalTime / float(particlesNumber), rightLimitReachedTimes /
float(particlesNumber)};
        float total[2];

        MPICHECK(MPI_Reduce(buf, total, 2, MPI_FLOAT, MPI_SUM, MASTER, MPI_COMM_WORLD));

        if (rank == MASTER) {
                auto printHeader = [argc, argv, commSize](ofstream& str) {
                        str << "-----------------------TASK 1-----------------------\n";
                        for (int i = 0; i < argc; ++i) {
                                str << argv[i] << " ";
                        }
                        str << " on " << to_string(commSize) << " procs.\n\n";
                };


                ofstream oStr(outFile.empty() ? "output.txt" : outFile, ofstream::out);
                printHeader(oStr);
                oStr << "Average time: " << total[0] << "\n";
                oStr << "Right limit reaching probability:"  << total[1] << "\n\n";
                oStr << "----------------------------------------------------\n";
                oStr.close();


                ofstream statStr(statFile.empty() ? "stat.txt" : statFile, ofstream::out);
                printHeader(statStr);
                statStr << "Total time: " << to_string(endTime - startTime) << "\n\n";
                statStr << "----------------------------------------------------\n";
                statStr.close();
        }

        //-------------------------------------------------------------------------------
        MPICHECK(MPI_Finalize());
        return 0;
}
```