Краткий отчёт по решению

Васильев Павел Петрович
группа 303 ВМК МГУ

31 октября 2025 г.

Аннотация

В данном репозитории представлен алгоритм решения системы линейных алгебраических уравнений методом Холецкого на языке программирования C++. Были проведены тесты для $x \in \mathcal{R}^n : \ x \in \mathcal{U}_{[-1,1]}$, а также использовались вещественные числа разного порядка точности: float16, float32, float64. Для всех тестов были вычислены максимальные и средние нормы невязки, погрешности решений и времени работы. Использована норма $|| * ||_\infty$. В работе были использованы языки C++ для вычислений и Python для анализа результатов.

Ссылка на репозиторий: https://github.com/GoodDay-lab/cholesky-decomposition

1 Постановка задачи

Перед нами поставлена задача решить следующую систему линейных уравнений:

$$A \in \mathcal{R}^{n \times n} : \ A^T = A, \ A > 0; \ b \in \mathcal{R}^n \to Ax = b \tag{1}$$

2 Почему не Гаусс?

Одним из первых алгоритмов для решения подобных систем можно назвать алгоритм Гаусса, в сущности, это нахождение LU разложения для матрицы A. Данный алгоритм неустойчив при небольшом значении диагонального элемента матрицы (близком к машинному нулю), к примеру:

$$A = \begin{bmatrix} 10^{-8} & 1.0 \\ 1.0 & 1.0 \end{bmatrix}$$

$$\tilde{L} = \begin{bmatrix} 1.0 & 0 \\ 10^{-8} & 1.0 \end{bmatrix} \quad \tilde{U} = \begin{bmatrix} 10^{-8} & 1.0 \\ 0 & -10^{-8} \end{bmatrix}$$

Тогда

$$\tilde{L}\tilde{U} - A = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

С другой стороны, у нас есть алгоритм Холецкого, который лишён данной проблемы, это объясняется тем, что $c_{kk} = \sqrt{a_{kk} - (\dots)}$, даже если a_{kk} близко к минимально представимому на машине типу float, деление всё равно происходит на его корень, что больше, чем изначальное число. Более подробное обоснование можно прочесть в учебнике Е.Е.Тыртышникова "Методы Численного Анализа".

3 Алгоритм Холецкого для решения СЛАУ

Вернёмся к изначальным условиям задачи (1).

Рассмотрим следующее уравнение:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & 0 & 0 \\ c_{21} & c_{22} & 0 \\ c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} c_{11} & c_{21} & c_{31} \\ 0 & c_{22} & c_{32} \\ 0 & 0 & c_{33} \end{bmatrix}$$

Попробуем вычислить коэффициенты:

$$c_{11} = \sqrt{a_{11}}, \quad c_{21} = \frac{a_{21}}{c_{11}}, \quad c_{31} = \frac{a_{31}}{c_{11}}$$

$$c_{22} = \sqrt{a_{22} - c_{21}^2}, \quad c_{32} = \frac{a_{32} - c_{31}c_{21}}{c_{22}}$$

$$c_{33} = \sqrt{a_{33} - c_{31}^2 - c_{32}^2}$$

Обобщая для любого n, получим:

$$c_{kk} = \left(a_{kk} - \sum_{j=1}^{k-1} c_{kj}^2)^{\frac{1}{2}} \right) \tag{2}$$

$$c_{ik} = \frac{\left(a_{ik} - \sum_{j=1}^{k-1} c_{ij}c_{kj} \right)}{c_{kk}}, \ \forall 1 \leq i < k \tag{3}$$

Хорошо, мы получили удобное представление $A = LL^T$ из треугольных матриц L, а для них мы можем легко решить систему уравнений строчка за строчкой со сложность $O(n^2)$. Что ж тогда давайте пользоваться принципом разделяй-и-властвуй и решим две более простые системы линейных уравнений

$$Ax = b \Longrightarrow LL^T x = b \Longrightarrow \begin{cases} Ly = b \\ L^T x = y \end{cases} \tag{4}$$

Что ж, отлично! Мы разобрались с этим алгоритмом в теории, время проверять его работу на практике.

4 Результаты

Полученные результаты представлены на графиках ниже. К примеру, один и тот же алгоритм работает по-разному для разных типов данных, наименьшая величина достигается на f32 (6), а наибольшая, как ни странно, на f16 (3).

Так же, при увеличении типа float увеличивается и качество работы алгоритма, достаточно сравнить норму невязки на f16 (1) и f64 (7).

5 Выводы

У нас получилось реализовать устойчивый алгоритм для решения системы СЛАУ методом Холецкого на языке программирования C++. Средняя скорость алгоритма составляет 12 мс для матрицы 100×100, в зависимости от типа вещественного числа. Код доступен по ссылке в аннотации.

6 Графики

Рис. 1: f16


График средней погрешности по \inf-норме
0.01000
0.00975
0.00950
0.00925
0.00900
0.00875
0.00850
0.00825
5
10
15
20
25
30
35
40
Количество тестов

Рис. 2: f16

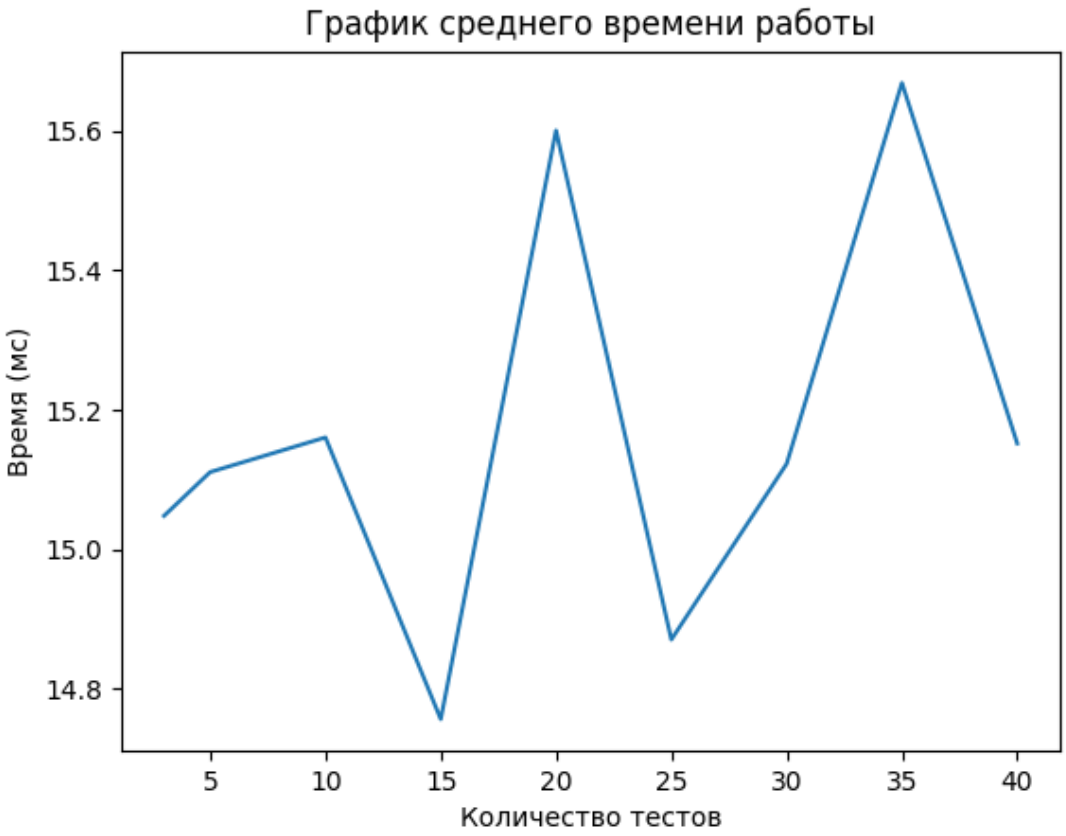
График среднего времени работы
Время (мс)
15.6
15.4
15.2
15.0
14.8
5
10
15
20
25
30
35
40
Количество тестов

Рис. 3: f16

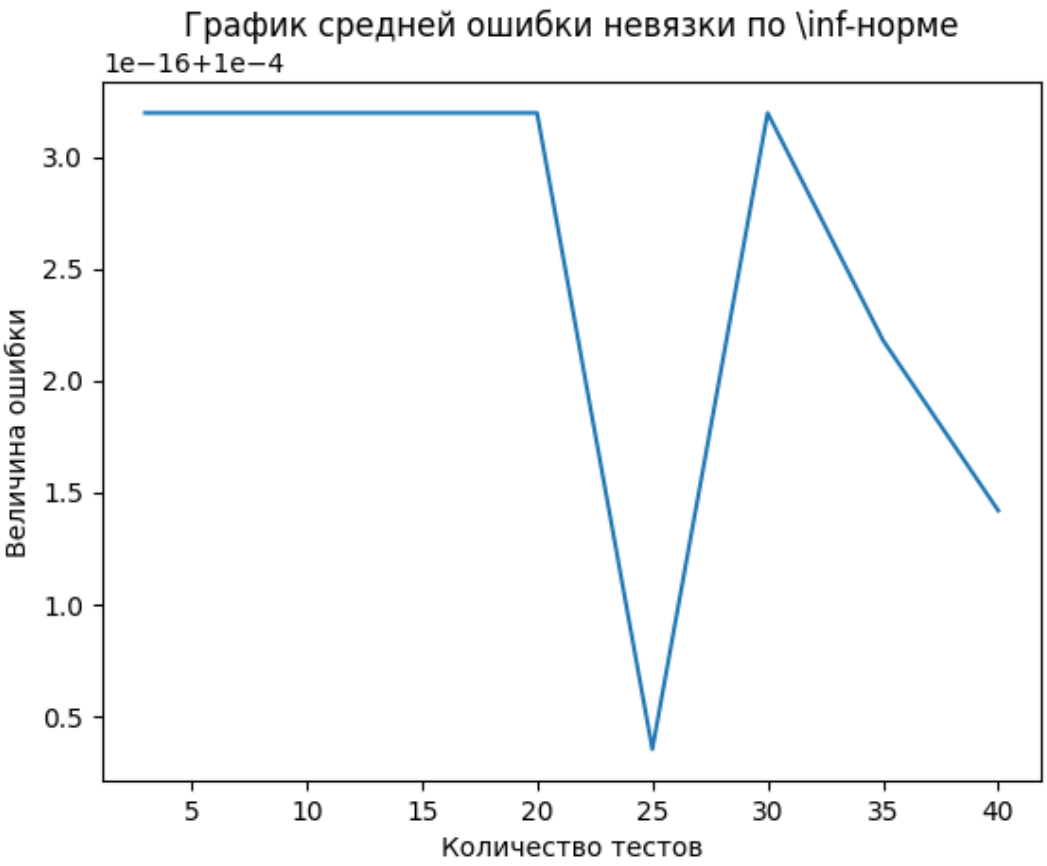
График средней ошибки невязки по \inf-норме
1e−16+1e−4
3.0
2.5
2.0
1.5
1.0
0.5
Величина ошибки
5
10
15
20
25
30
35
40
Количество тестов

Рис. 4: f32

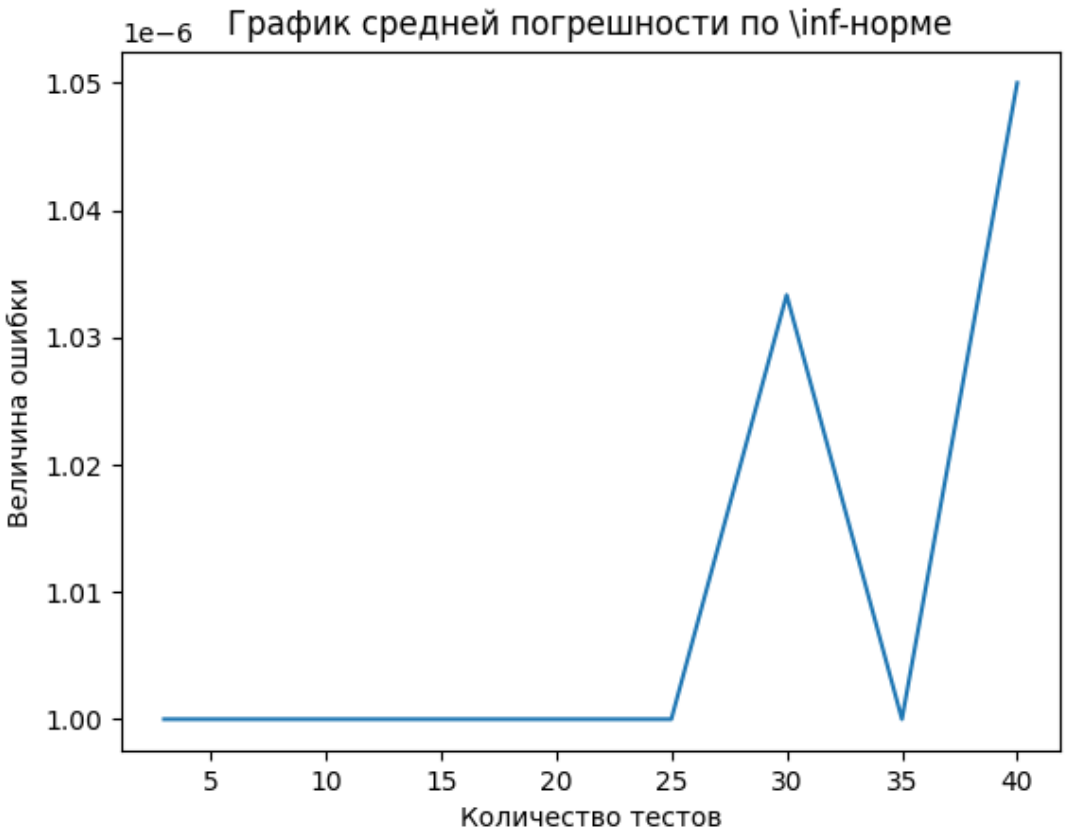
График средней погрешности по \inf-норме
1e−6
1.05
1.04
1.03
1.02
1.01
1.00
Величина ошибки
5
10
15
20
25
30
35
40
Количество тестов

Рис. 5: f32

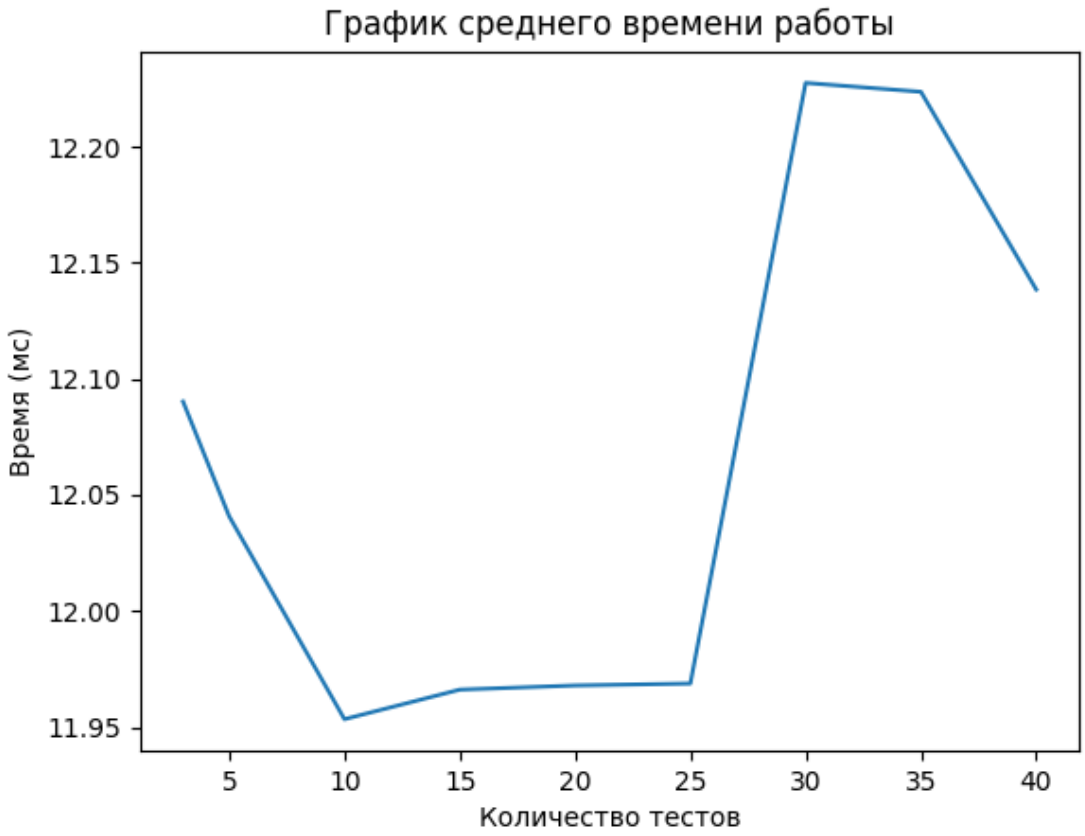
График среднего времени работы
Время (мс)
12.20
12.15
12.10
12.05
12.00
11.95
5
10
15
20
25
30
35
40
Количество тестов

Рис. 6: f32

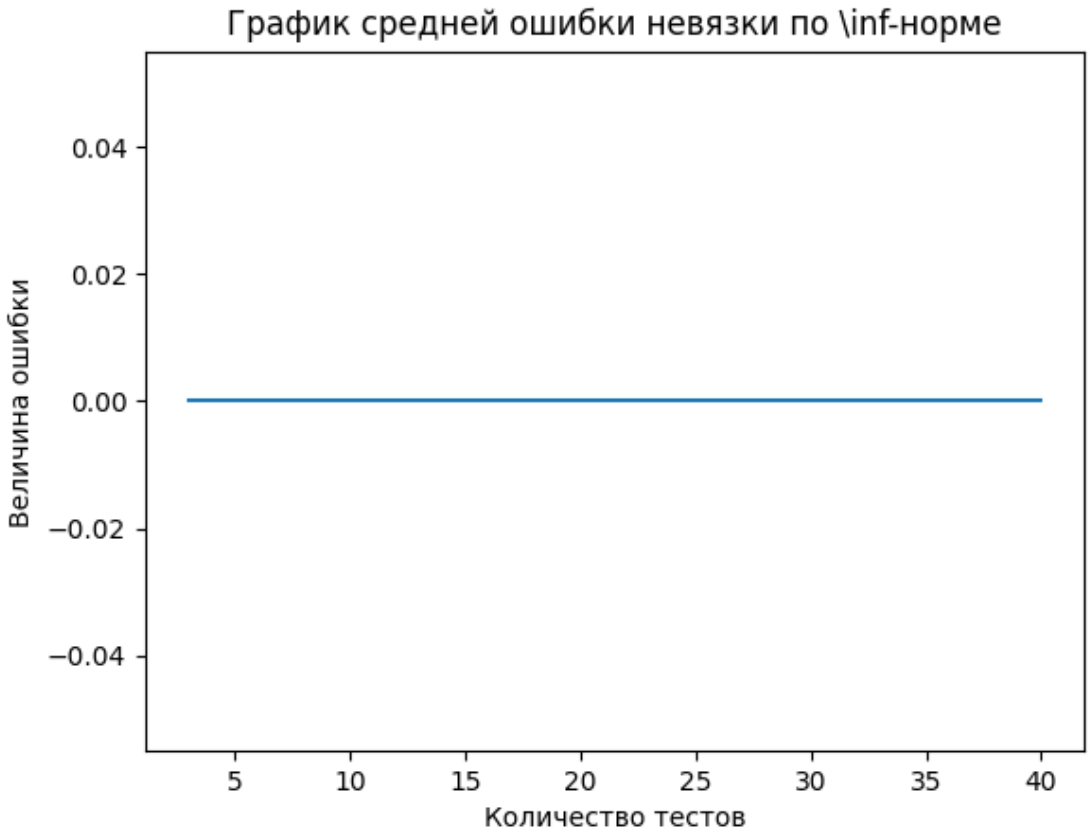
График средней ошибки невязки по \inf-норме
Величина ошибки
0.04
0.02
0.00
−0.02
−0.04
5
10
15
20
25
30
35
40
Количество тестов

Рис. 7: f64

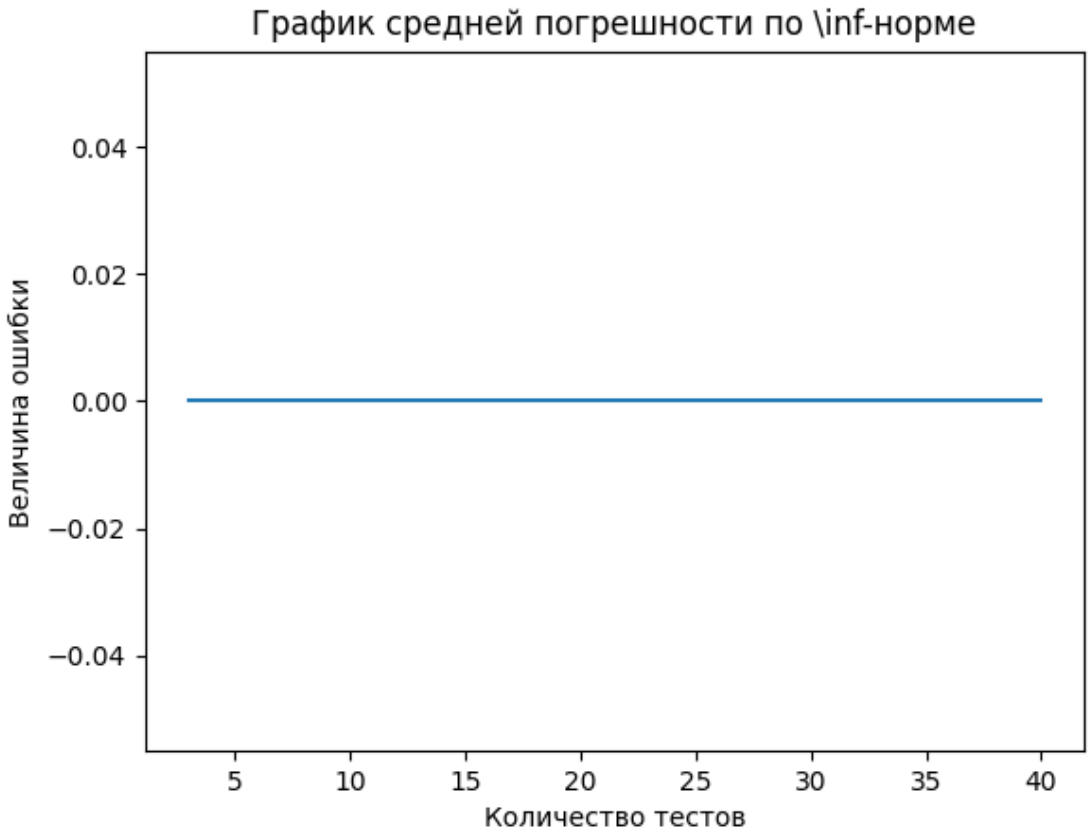
График средней погрешности по \inf-норме
Величина ошибки
0.04
0.02
0.00
−0.02
−0.04
5
10
15
20
25
30
35
40
Количество тестов

Рис. 8: f64

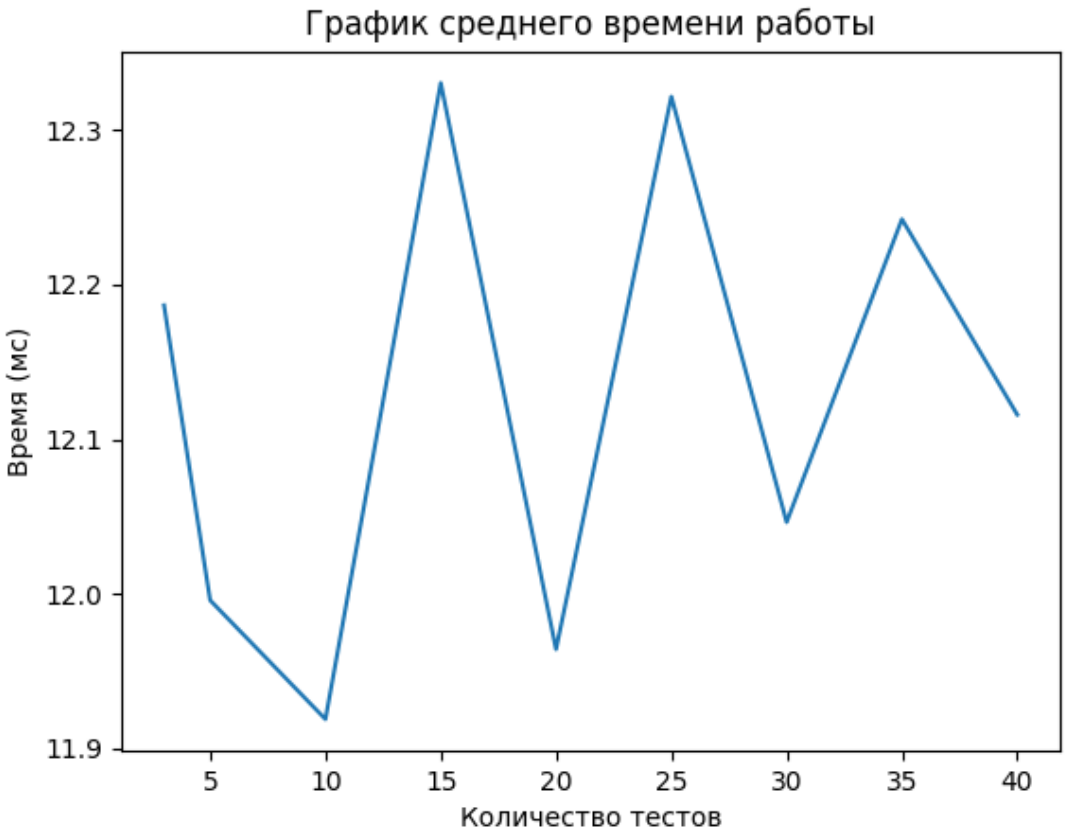
График среднего времени работы
Время (мс)
12.3
12.2
12.1
12.0
11.9
5
10
15
20
25
30
35
40
Количество тестов

Рис. 9: f64