**Федеральное государственное автономное образовательное учреждение высшего образования**

**“Санкт-Петербургский политехнический университет Петра Великого”**

Кафедра “Прикладная математика”

Дисциплина “Численные методы”

Отчет по учебной практике

Студента Хрипункова Д.В.

Группы 3630102/80001

Руководитель практики: Добрецова С.Б.

Санкт-Петербург 2019 г.

1. Формулировка задачи

Рассмотреть матричное уравнение вида Ax = b (систему линейных алгебраических уравнений) и найти приближённое решение методом простых итераций с заданной точностью ε.

По полученным данным построить зависимости погрешности ||x\* - x|| и невязки ||Ax – b|| от приближения ε для проверки сходимости метода, ||x\* - x|| и ||Ax – b|| от определителя матрицы det(A) при det(A) → 0 и зависимость числа итераций от приближения решения ε и начального приближения x0. По зависимостям построить графики в среде MATLAB.

2. Постановка задачи

В данной работе решается задача нахождения решения СЛАУ вида

Которое можно преобразовать в матричный вид как *Ax = b*, где b = (b1, b2, …, bn) – вектор свободных членов, x = (x1, x2, …, xn) – вектор неизвестных, – вещественная матрица коэффициентов размером n x n.

Чтобы существовало решение такой системы, необходимо и достаточно, чтобы определитель матрицы A не был равен нулю. Решение находится итерационным методом (прямых итераций), то есть алгоритм работает, пока не приблизится к необходимому решению с заданной точностью.

3. Алгоритм метода

Путь у нас дано матричное уравнение вида *Ax = b* и нам нужно найти его прибилжённое решение с точностью ε. Эту систему можно преобразовать к эквивалентной системе вида *x = Bx + c*, где x – тот же вектор неизвестных, B и c – новые матрица и вектор, получаемые из A и b. Из этой системы можно получить итерационную формулу *x(k+1)= Bx(k) + c*, начинающуюся с некоторого вектора x(0). Алгоритм метода можно описать так:

1. По формуле ci = bi / Aii находим вектор *c*
2. По формуле Bij = -Aij / Aii, при i ≠ j и Bij = 0 при i = j
3. Берём вектор x0 = c
4. По итеративной формуле *x(k+1)= Bx(k) + c* находим вектор решений до тех пор, пока ||x\* - x(k)|| > ε

4. Анализ задачи

Для применения метода простых итераций необходимо и достаточно чтобы все собственные числа матрицы *B* были по модулю меньше 1. В этом случае итерационная последовательность *x(k+1)= Bx(k) + c* будет сходиться к единственному вектору решений x\* системы *Ax = b*. Для достижения этого условия для матрицы *B* в исходной матрице *A* должно присутствовать диагональное преобладание. Также справедливы оценки погрешности и . Первую можно использовать при проведении k-ой итерации, а вторую до начала счёта.

5. Тестовый пример

Возьмём СЛАУ вида *Ax = b* с данными входными параметрами:

, а ожидаемое решение .

Подставив входные данные, можно убедиться в правильности системы.

Теперь попробуем получить вектор *x* методом простых итераций и определим начальные данные для итеративной формулы:

1. По формуле ci = bi / Aii

c1 = b1 / A11 = 9.8 / 7 = 1.4

c2 = b2 / A22 = 41 / 8 = 5.125

c3 = b3 / A33 = 10.6 / 5 = 2.12

1. По формуле Bij = -Aij / Aii, при i ≠ j и Bij = 0 при i = j рассчитаем значения B:

B11 = 0

B12 = A12 / A11 = 0.4 / 7 = 0.05714

B13 =A13 / A11 = 0.4 / 7 = 0.05714

B21 = A21 / A22 = 0.2 / 8 = 0.025

B22 = 0

B23 = A23 / A22 = 0.4 / 8 = 0.05

B31 = A31 / A33 = 0.1 / 5 = 0.02

B32 = A32 / A33 = 0.1 / 5 = 0.02

B33 = 0

1. Возьмём вектор x0 = c
2. Сделаем 4 итерации по итеративной формуле *x(k+1)= Bx(k) + c*

x(1) = Bx(0) + c =

x(2) = Bx(1) + c =

x(3) = Bx(2) + c =

x(4) = Bx(3) + c =

Решение оказалось очень точным при малом количестве итераций.

6. Контрольные тесты

Для тестов будем использовать матрицу размером 12х12, элементы которой создаются с учётом диагонального преобладания с помощью псевдорандомной функции rand() с сидом, заданным по умолчанию, из библиотеки stdlib.h на языке си. В каждой строке матрицы сначала вычисляются все элементы, кроме диагонального, как случайные числа меньшие единицы, а потом диагональный элемент считается как

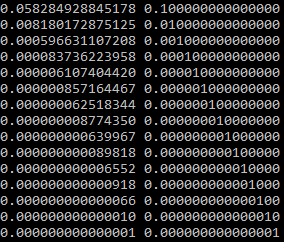
Весь анализирующий функционал реализован на языке си в visual studio 2019, а графическая интерпретация в пакете MATLAB.

При рассмотрении всех зависимостей от приближения решения ε оно будет изменяться от 10-1 до 10-15 с домножением на 10-1 на каждом шаге.

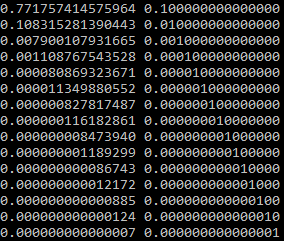
При рассмотрении зависимостей от определителя матрицы det(A) будем брать неизменное приближение ε = 10-14 и изменять определитель матрицы, деля на каждом шагу первый столбец матрицы A на и вектор b на то же число, чтобы сохранить вектор решений прежним. Для того, чтобы в первой строке оставалось диагональное преобладание, перед началом исследования домножим первый элемент матрицы А на столько раз, сколько будет итераций изменения определителя.

При рассмотрении зависимости от начального приближения возьмём его как вектор решения x\* и будем умножать на число k. K берём как число 100 и на каждом шагу делим на 1.1.

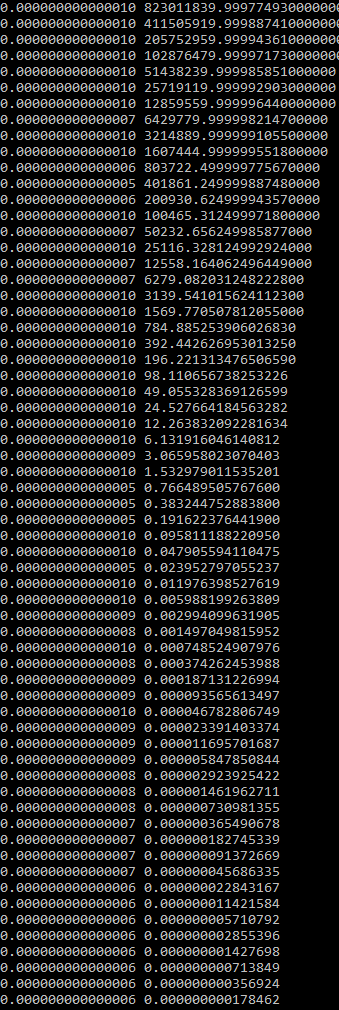
Пример работы программы, где первый столбец – ошибка вычисления ||x\* - x||, а второй - приближение решения ε:



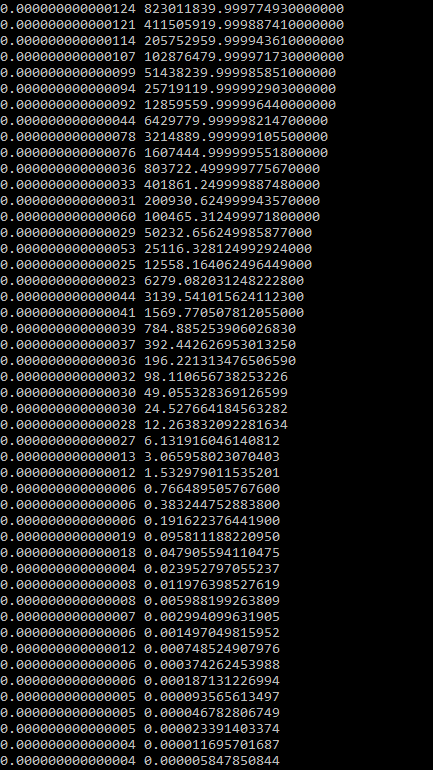
Пример работы программы, где первый столбец – невязка ||Ax - b||, а второй - приближение решения ε:



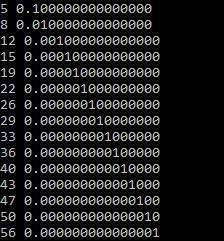
Пример работы программы, где первый столбец - ошибка вычисления ||x\* - x||, а второй – определитель матрицы A:



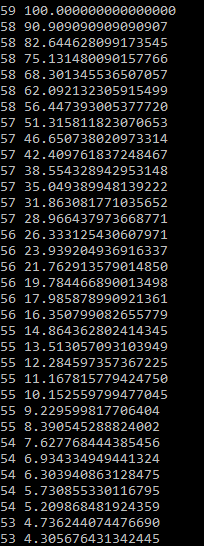
Пример работы программы, где первый столбец - невязка ||Ax - b||, а второй – определитель матрицы A:



Пример работы программы, где первый столбец – число итераций N, а второй – приближение решения ε:



Пример работы программы, где первый столбец – число итераций N, а второй – начальное приближение x0:



7. Модульная структура программы

double\* CreateMatrix(int size) – функция создания квадратичой матрицы с заданным размером. Возвращает полученную матрицу

double\* CreateVector(int size) – функция создания вектора с заданным размером. Возвращает полученный вектор.

double\* VectSub(double\* v1, double\* v2, int size) – функция вычисления вектора разности двух векторов. Принимает на вход два вектора и их размер. Возвращает вектор разности.

double FindNorm(double\* vect, int size) – функция вычисления нормы вектора. Принимает на вход вектор и его размер. Возвращает вещественное значение нормы.

double\* CreateBetaVector (double \*b, double \*A, int size) – создаёт вектор, обозначенный как вектор *c* в итерационной формуле. На вход получает вектор свободных членов *b*, исходную матрицу A и их размерность. Возвращает вектор *c.*

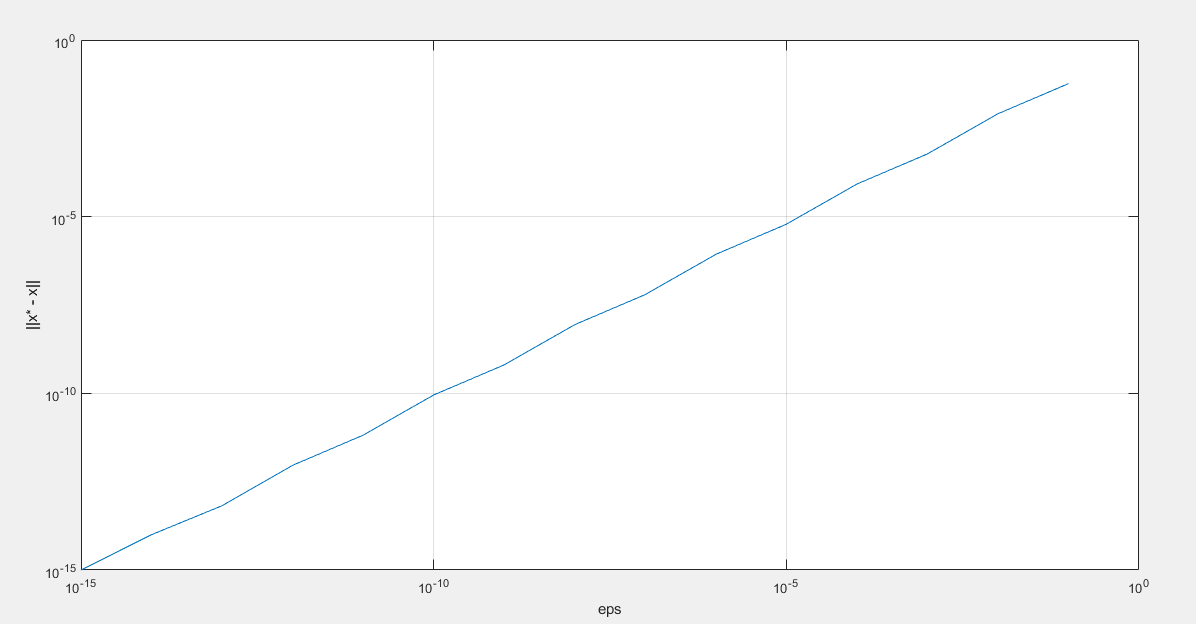
double\* CreateAlphaMatrix (double \*A, int size) – создаёт матрицу, обозначенную как матрица *B* в итерационной формуле. На вход получает исходную матрицу *A* и её размерность. Возвращает матрицу *B*.

double\* CreateXVector (double\* beta, double\* alpha, double\* x, int size) – считает новое значение вектора *x* по итерационной формуле. На вход принимает вектор *c*, матрицу *B*, текущий вектор *x* и их размерность. Возвращает новое значение вектора *x*.

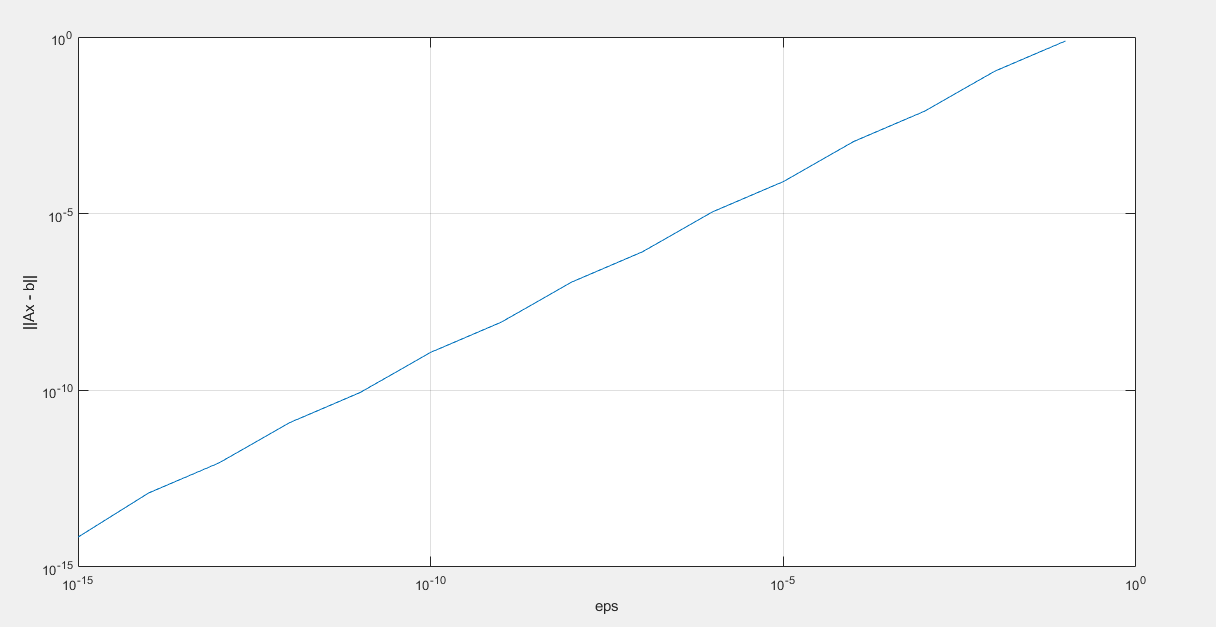
double GetDet(double\* A, int size) – считает определитель матрицы по определению через перестановки , где Nk – количество инверсий в k-ой перестановке, а j1(k), j2(k), …, jn(k) – k-ая перестановка множества j. Принимает на вход матрицу *A* и её размерность. Возвращает значение определителя.

8. Численный анализ

Графики зависимостей построенные в MATLAB в логарифмических масштабах с помощью loglog(x, y):

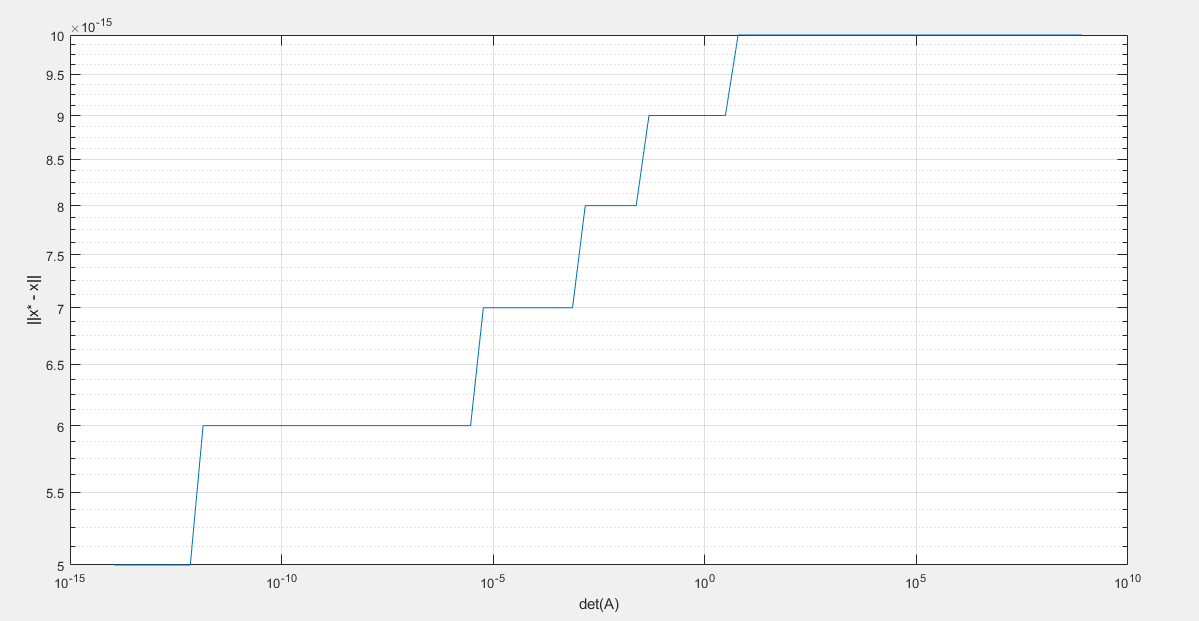


Зависимость вычислительной ошибки ||x\*-x|| от приближения решения ε

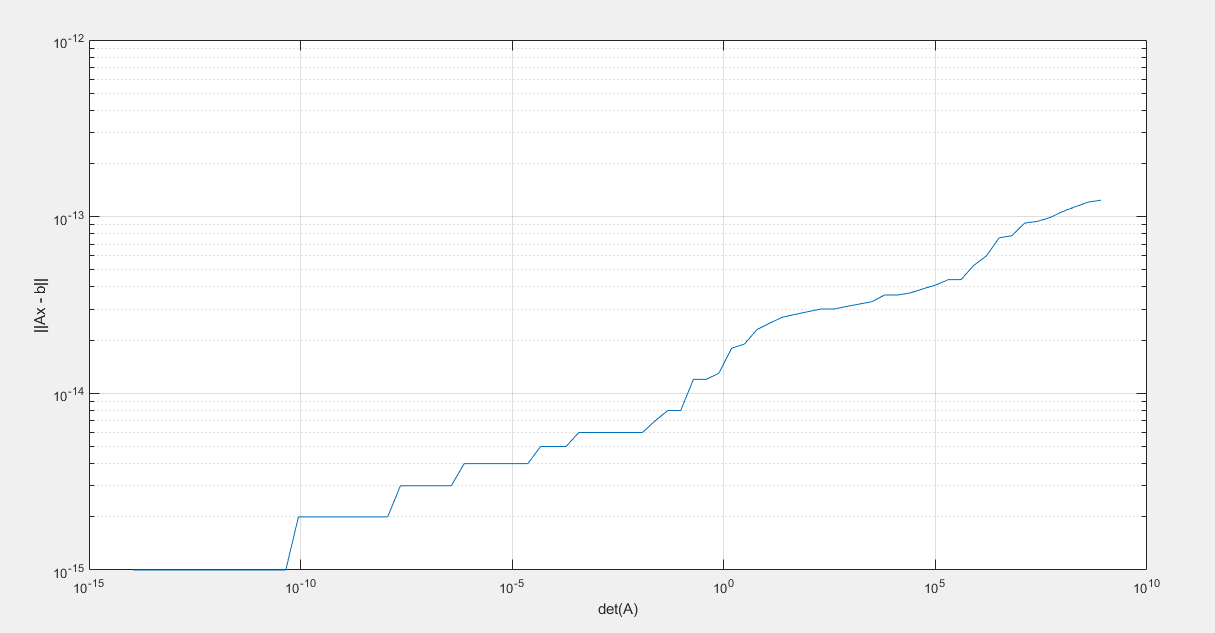


Зависимость невязки ||Ax-b|| от приближения решения ε

Эти два графика дают зависимость очень близкую к линейной. График невязки идёт чуть выше графика вычислительной ошибки, но различие незначительно.

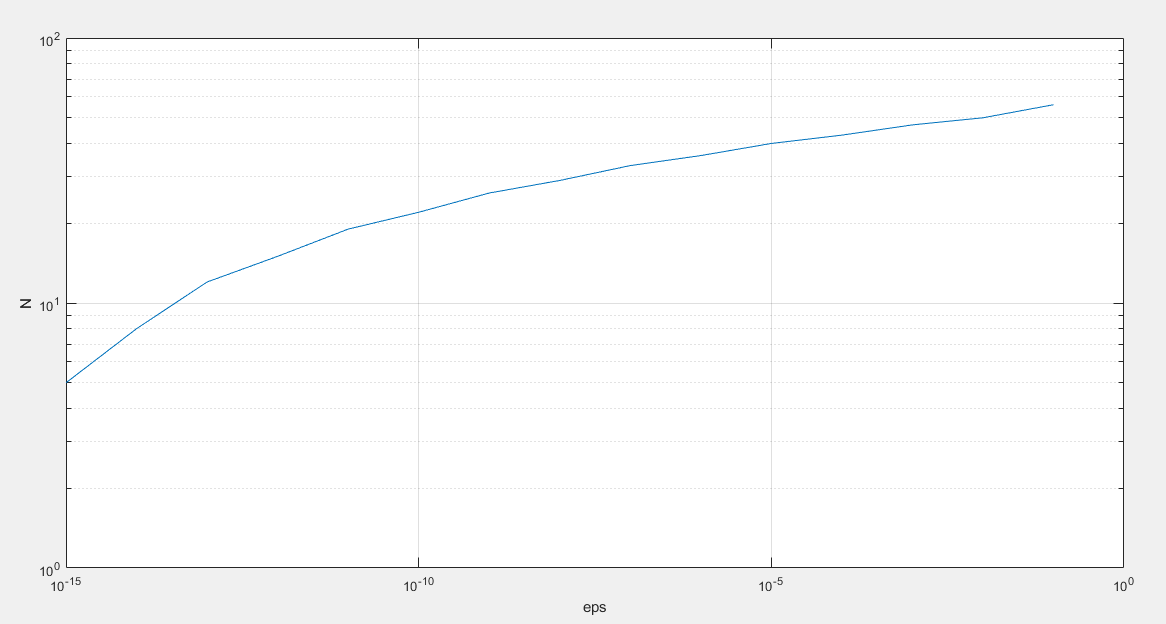


Зависимость вычислительной ошибки ||x\*-x|| от определителя исходной матрицы det(A)



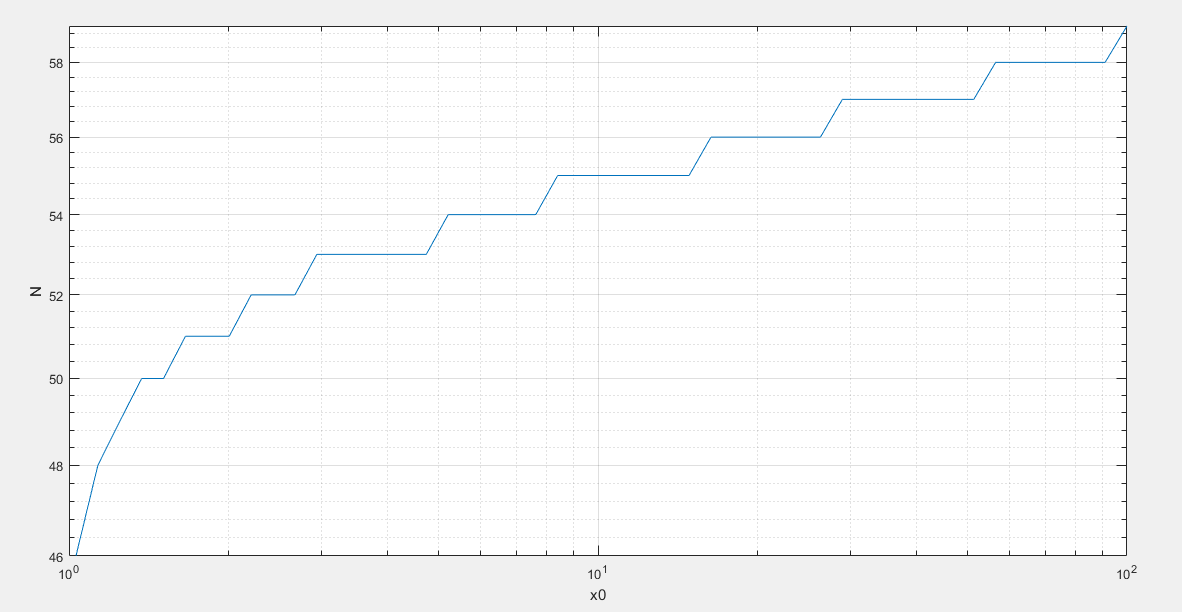
Зависимость невязки ||Ax-b|| от определителя исходной матрицы det(A)

Зависимость показывает, что при увеличении определителя ступенчато увеличивается и вычислительная ошибка вектора решений. Невязка тоже увеличивается, но более плавно и эту зависимость можно сравнить с линейной.



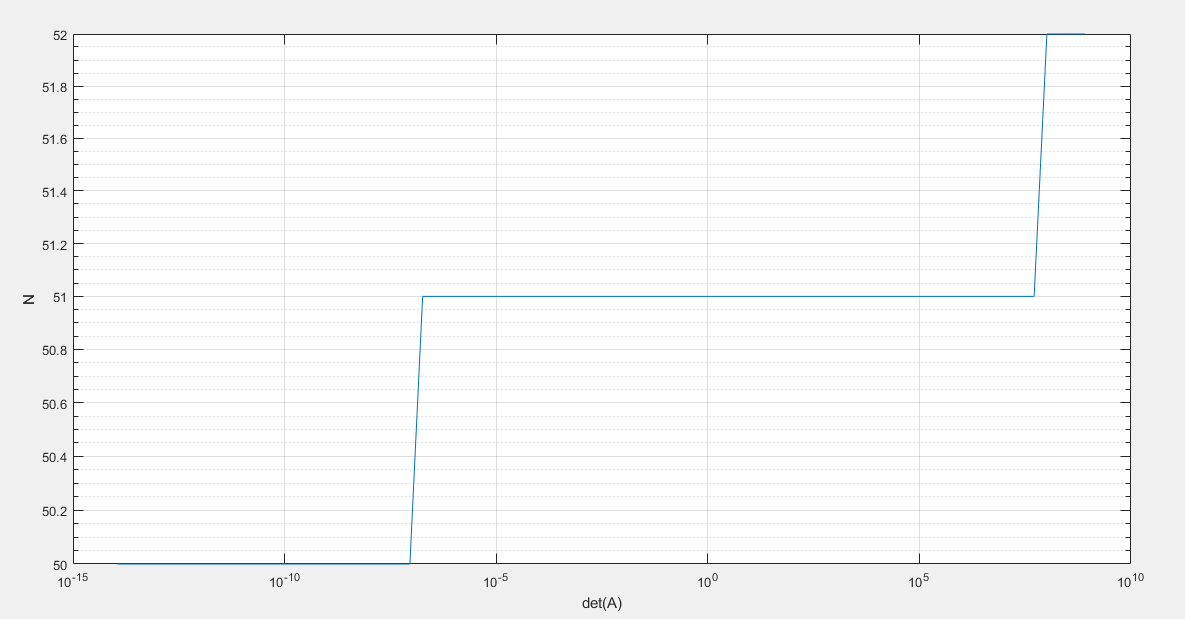
Зависимость числа итераций N от приближения решения ε

При увеличении приближения решения ε число итераций полупараболически возрастает.



Зависимость числа итераций N от начального приближения x0

Зависимость числа итераций от начального приближения x0 более ступенчата и так же растёт при увеличении начального приближения.



Зависимость числа итераций N от определителя исходной матрицы det(A)

По графику видно, что число итераций меняется незначительно при достаточно больших изменениях определителя.

9. Вывод

В целом метод простых итераций показал себя как довольно точный метод, т.к. вычислительная ошибка всегда заметно ниже необходимого приближения, а значение невязки в допустимых пределах.

При этом вычислительная ошибка не сильно зависит от определителя матрицы, т.к. изменяется в порядке 10-14, когда определитель может изменяться вплоть до 108. Также вычислительная ошибка может быть одной для целого ряда порядков определителя. Невязка в это же время чуть больше зависит от определителя матрицы, но её порядок варьируется от 10-12 до 10-15 при порядке определителя так же до 108. На промежутке порядков определителя от 10-15 до 10-11 невязка даёт одно и то же значение.

Однако, метод является очень затратным по итерациям и их число может достигать более 60 при большой требуемой точности или плохом начальном приближении. Даже при наилучшем приближении, но высокой точности ответа, алгоритм затрачивает большое количество итераций. При малой точности ситуация становится лучше, но метод всё ещё является достаточно затратным.