Лабораторная работа №1

Солодуха Дмитрий, 13 группа

1. Самый непонятный пункт лабораторной, наверное, если не использовать нормальный генератор из новых стандартов `C++`. Так как точность у `double` больше 13 знаков, то сгенерировать необходимые матрицы и вектора можно с помощью следующего генератора:

   ```
   std::uniform_real_distribution<double>(a, b);
   ```

   Сразу же замечу, что матрицы, которые мы генерируем являются очень хорошими (из-за диагонального преобладания), поэтому некоторые не очень хорошие методы всё равно работают отлично.

2. Просто нахожу число обусловленности по определению. Метода Гаусса-Жордана написать очень просто — никаких выбор элементов делать не надо, так как мы не хотим поменять число обусловленности. Реализовал полноценный обратный ход, хотя можно было немного схитрить, но это был бы уже не базовый алгоритм. Работает достаточно долго; на тестовых прогонах время было около `271 ms`, что больше времени работы других методов, хотя и логично — мы же решаем сразу n алгебраических уравнений.

   Полученные числа обусловленности у матриц в лежали в интервале $(4.5, 5)$, как я уже написал ранее — это заслуга способа, которым мы их генерируем. То есть задача хорошо обусловлена (при таком-то $\epsilon_m = 2^{-52}$).

3. Немного модифицируем базового Гаусса. Перед тем, как занулять столбец ищем максимальный элемент и запоминаем в векторе какие строки поменяли. Думал, что будет работать гораздо медленнее стандартного алгоритма: выбора максимального за $O(k^2)$, обмен строк за $O(1)$ и обмен столбцов за $O(k)$. Но метод все равно работает быстро и вдобавок точно. При тестах наиболее оптимальным оказался выбор не по всей матрице, а по столбцу, что очевидно в силу сложности обмена столбцов.

4. При реализации построения разложения просто взял Гаусса из прошлого задания и запомнил перестановки столбцов и строк. Матрицу P хранил в виде двух векторов — один со строками, другой со столбцами. LU — в одной.

   При решении уже полученного разложения просто прохожусь два раза по треугольной матрице и попутно расставляю элементы векторов на свои места.

   Нормы получились почти такими же, как и в прошлом пункте. Минимальные отличия связаны с появившимися побочными вычислениями. По времени тоже отличий почти нет. Странным показалось, что при тестах время решения полученной системы было равным `0`. Не знаю, как компилятор мог оптимизировать решение треугольных матриц, но он явно это сделал.

5. Фактически упрощаю Гаусса из прошлых пунктов. Перестаю искать главный элемент, а вместо этого просто делю на корень. Можно было хранить только половину матрицы, пользуясь тем, что в этом методе она у нас по условию должна быть симметричной, но для этого пришлось бы нормально перелопатить начальный класс. При обращении к элементам все равно пользуюсь только верхней частью матрицы (хотя матрицу 256×256 можно и полностью закешировать). Матрицу D логично хранить в виде вектора — это и делаю.

Полученное время работы почти в 4 раза быстрее Гаусса с выбором главного элемента по всей матрице. А вот по точности они почти не отличаются. Уменьшение времени обусловлено тем, что мы изменяем только половину матрицы да и не ищем главный элемент. Жаль, что работает это только для ограниченного класса матриц — симметричных.

6. Метод релаксации пишется легче, чем считается руками. Использую векторную форму, так как матричная здесь не целесообразна. Метод выглядит быстрым. Не каких сложных операций мы не делаем за $O(n^2)$ считаем очередную итерацию.

   Точность я задавал сравнимую с предыдущими методами, поэтому так и получилось. По времени работы метод обгоняет Гаусса **только** в 1.5 раза, что в связи с предыдущими рассуждениями немного странно. Возможно это связано со слишком большой заданной точностью и не очень высокой скоростью сходимостью метода. Кстати, моя w для него не оптимальна.

7. Метод отражений реализовал полностью по алгоритму, только лишь обобщил на не квадратные матрицы, чтобы потом использовать его в других методах.

   По значениям нормы чуть проигрывает только методу квадратного корня, но и тот не сильно обогнал другие. Опять же всё дело в генерируемых матрицах, на которых всё должно считаться хорошо. А вот по времени работы проигрывает даже Гауссу почти в 1.5 раза. Всё-таки метод придуман для того, чтобы не изменять число обусловленности и в силу этого улучшать точность, а тут с этим и так всё хорошо.

   Хотя при использовании метода отражения в приложениях, то есть для получения QR-разложения он показывает себя отлично. У меня изначально МНК был написан через трансформацию Гаусса и я долго не мог понять почему моя программа зависала на GMRES. Оказалось, что я просто не мог получить необходимую точность решения из-за плохой обусловленности задачи. А с QR уже всё заработало.

8. Реализовать было очень легко — просто вызвать Хаусхолдера. Попутно при реализации ощутил боль от решения через нормальные уравнения.

   О времени работы судить сложно, потому что мы фактически обрезаем матрицу на ~50 столбцов (для моего варианта). Но в связи с моей он находится на уровне Хаусхолдера. Про норму тоже ничего особенного не скажешь. Мы всё-таки решаем задачу наименьших квадратов и было бы странно, если бы она было похожа на нормы в других методах. Поэтому она может показаться **огромной**, но на самом деле это не так. Мы ведь кучу столбцов откинули, а в моём варианте там и значения не маленькие. Вот и получили среднюю норму `14468`.

   А если использовать трансформацию Гаусса… Но лучше не использовать.

9. С GMRES'ом было неприятно. Из-за того, как я реализовал свои матрицы было сложно придумать оптимальный способ реализации алгоритма. Я, конечно, что-то попробовал, но это точно далеко от совершенства. Мне достаточно часто приходится выделять память. И хоть асимптотику это никак не портит, но все равно влияет на скорость метода.

   По времени работы в два раза обгоняет Гаусса, но и в два раз проигрывает методу квадратного корня. Если говорить о точности — просто получаем заданную. Большую точность ставить неинтересно, потому что тогда теряем суть итерационного метода.

10. Здесь повторяем ошибку прошлого пункта, но все равно ускоряем метод.

Можно было еще и нормально написать вращения, чтобы всё это работало оптимально и без пересчёта QR с прошлых итераций, но на это не особо было время.

По времени работает быстрее всех методов, по точности так же. Такая скорость связана с тем, что мы строим очень хороший базис и выкидываем ненужные итерации из прошлого пункта (фактически увеличиваем скорость сходимости), а так же простого решения задачи наименьших квадратов. Ведь в матрице Хессенберга и так почти все нули, то есть довести её до треугольной гораздо проще.

Изначально я думал, что среди всех итерационных методов выиграет метод релаксаций. Хорошие сгенерированные матрицы, простые векторные формулы — всё это должно было ему помочь. Но он всё равно проиграл GMRES'у.

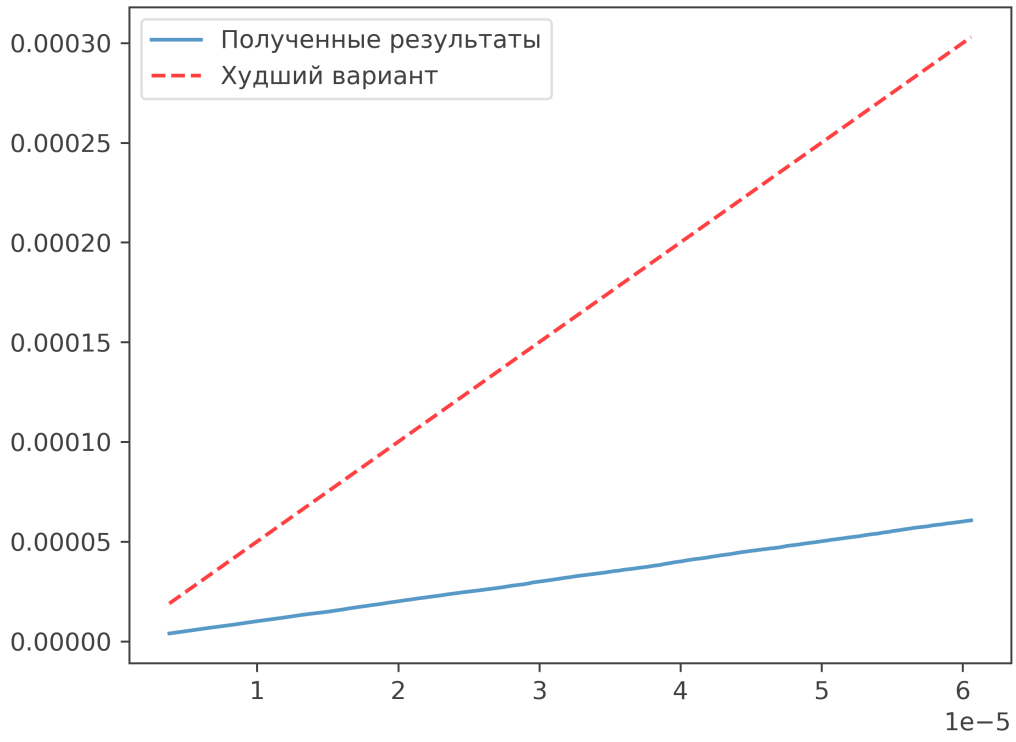
11. Отчёт по работе программы в файле `Reports\reports.txt`

12. Исследовал для матрицы с числом обусловленности $\nu(A) = 4.93768$. Если вспомнить определение числа обусловленности:

$$\delta_x \leq \nu(A)\delta_b$$

То провести анализ очень просто — относительная погрешность изменения вектора x не должна превышать относительную погрешность b умноженную на 5. На самом деле исследования результата зависит ещё и от метода, но в задании про это ничего не сказано, поэтому я взял метод Гаусса и получил, что $\delta_x \leq 2\delta_b$. То есть всё выполняется.

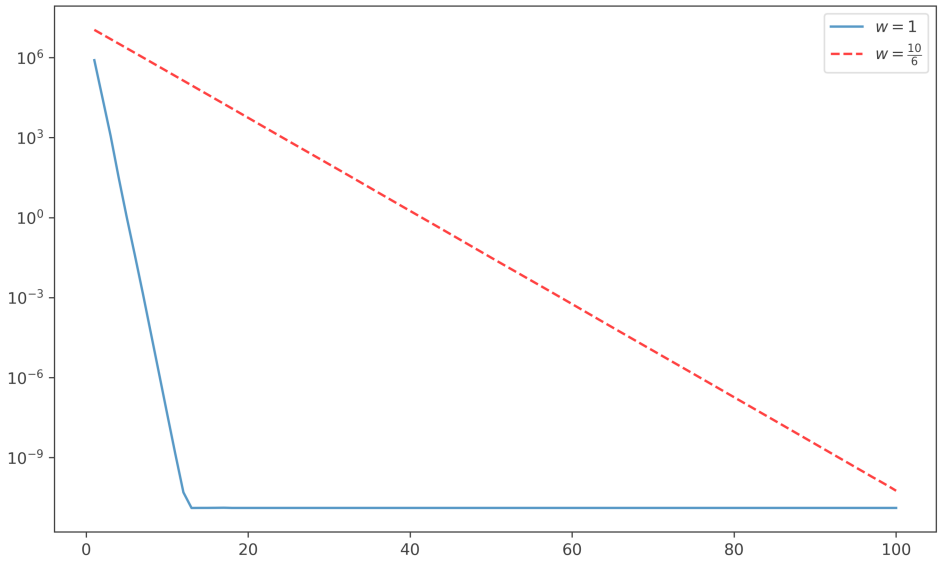
Я построил график зависимости δ_x от δ_b он получился линейным. При вычислении коэффициента пользовался МНК, кстати, только вот из `numpy`. Получил $\alpha = 1.067 \leq \nu(A)$, что подтверждает вышесказанное.



Попробовал и другие методы, но на них результаты были схожими. График по для Гаусса лежит тут: `Graphs\condition_gauss.png`.

13. Просто на каждой итерации выводим норму и строим графики. $w = 0$ не сходится, потому что значения не обновляются, поэтому убираем сразу. Для нахождения лучшего коэффициента смотрим на результаты на какой-либо итерации и берем коэффициент с наименьшей погрешностью.

    График для оставшихся w лежит тут: `Graphs\relaxation.png`.



    Из графика видно, что метод с $w = 1$ сходится лучше и уже на 17 итерации достигает максимума.
    Искать оптимальное значение для метода релаксации из условия максимума скучно, поэтому сделал это графически и получил, что $w_{min} \in (0.55; 1.5)$.

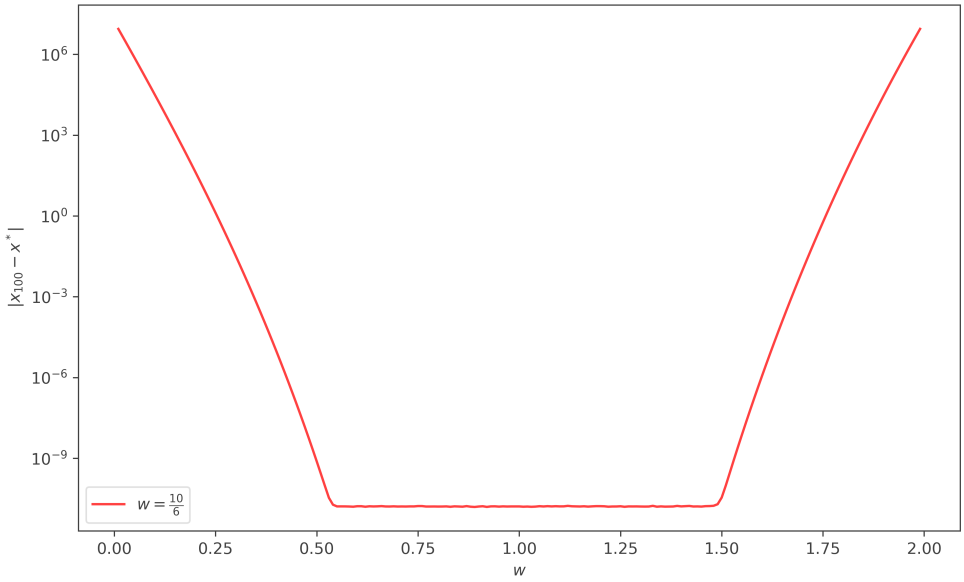


    График лежит тут: `Graphs\best_w_for_relaxation.png`.

14. Пункт на втором месте по непонятности.

Взял следующую матрицу A:

$$\begin{pmatrix} 7 & 1 & 2 & 3 \\ 1 & 15 & 3 & 6 \\ 2 & 3 & 16 & 8 \\ 3 & 6 & 8 & 20 \end{pmatrix}$$

И вектор x:

$$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

2. `6.77807`
3. `x = (1, 2, 3, 4)`
4. `x = (1.0000000000000044, 2.0000000000000044, 2.99999999999999956, 4)`
5. `x = (1.0000000000000022, 1.99999999999999956, 3.00000000000000044, 4.00000000000000178)`
6. `x = (1.0000000004906977, 2.00000000131116007, 3.00000000258097943, 3.99999999952298202)`
7. `(1.00000000000000044, 2, 2.99999999999999956, 3.99999999999999956)`
8. `x = (1.00000000000000044, 2, 2.99999999999999956, 3.99999999999999956)`
9. `x = (1.00000000000002132, 1.99999999999997158, 3.00000000000004263, 4)`
10. `x = (1.00000000000000022, 2, 3, 4)`

В правильность разложений прошу поверить.