Я закинул весь проект вижуал студио в папки и в зип. Но еду не смог его загрузить (и из-за этого я просрочил на несколько минут). Поэтому прикрепляю только исходники.

Все методы, используемые в заданиях реализованы в классе Matrix.

Сами задания реализованы в классе Task.

Тесты в папке tests; папка внутри папки с проектом в вижле.

Выводы в папке results; папка внутри папки с проектом в вижле.

В мэине есть функция экзампл которая демонстрирует работу на консоль. (закомментирована)

1. Task::task1()

Заполнение за квадрат (+ n по памяти).

Можно было сделать двумя циклами за квадрат и без доп. памяти (заполнить + посчитать суммы для диагональных элементов), но я сделал одним циклом запоминая сумму в вектор.

2. Task::task2()

Решение представляет собой матрицу (n 1) для удобства работы.

Заполнить x = O(n). Умножение матриц обычное (за куб (n m)).

3. Task::task3\_1() — найти обратную

Task::task3\_2() — найти норму

Создаю единичную матрицу и делаю копию основной (можно сократить время если не требуется сохранять оригинал основной матрицы). И повторяю для единичной все операции с основной (операции обычного гаусса). Получается 2 куба, т.к. надо идти вниз, а потом вверх. Так идет ход гаусса за куб и вниз и вверх на практике алгоритм работает относительно очень медлено.

Норма считается за квадрат очевидным образом.

4. Task::task4()

Тут и далее всегда создается копия исходной матрицы, над которой выполняются операции и которая впоследствии возвращается функцией (как в задании 3).

Гаусс выполняется за куб, но из-за поиска главного элемента увеличивается константа.

+ Еще найти само решение из треугольной матрицы квадрат.

Строки меняются за O(1).

5. Task::task5\_1() — найти LUP

Task::task5\_2() — решить LUP *x* *= b*

//я не понял, что значит LUx = b с волной поэтому делал обыный луп = б

*Алгоритм аналогичен Гауссу из 4 задания, но тут меняются столбцы. Чтобы их менять за O(1) создал вектор в котором сохранял перестановки; этот же вектор служит вектором для матрицы P из LUP.*

*В результате получается LU матрица и P вектор. Для экономии времени в LU не переставляются столбцы даже в конце и из P не делается матрица. (Для перестановки столбцов LU в правильный порядок есть отдельная функция за квадрат, также за квадраты можно получить отдельно L, U и P).*

*В решении СЛАУ LUPx = b также используется матрица LU с не переставленными столбцами (т.е. столбцы в неправильном порядке) и вектор P. За счет этого экономиться немного времени.*

L*z = b —*  U*z* *=* *b*’ — P*x* *=* *b*’’ —

6. Task::task6\_1() — найти LDLt

Task::task6\_1() — решить LDLt *x = b*

По аналогии с 5 заданием: в структуре LDLt разложения храню оптимизированные матрицу L и вектор D + использую их же для решения LDLt x = b. Для получения матриц L, D, Lt есть функции за квадрат.

Тут оптимизировано, то что не надо считать нижний треугольник из-за симметричности матрицы.

Для решения L*z = b (*иU*z = b)* можно было бы написать функцию и использовать ее в LUP и LDLt но я этого не делал (отражается на качестве кода; на времени никак не отразиться).

7.

Делю строки на диагонали и отнимаю матрицу от единичной. Далее по алгоритму пока разница между не станет . Если слишком много итераций (>3000), то метод скорее всего не сходиться, и я возвращаю -1 как кол-во итераций в качестве флага.

Трудоемкость не знаю + мне кажется у меня не очень оптимально из-за сохранения предыдущего решения + сохранения массива норм разностей для построения графика.

8. Результат в файле otchet.txt

Запускается функцией experiment() и выводит все задания в файл отчета.

В отдельном файле матрица с макс. числом обусловленности. И погрешности релаксации я добавлял в файл errors, так что там может много разных быть.

8.1. Не понял, что значит “… у всех матриц” у всех в мире??? Считал только для матриц A генерируемых в задании.

9. Результаты в файлах A1.txt и A2.txt

Для А2 ответы очень неточные, т.к. большие нормы. В релаксации больше 3 тысяч итераций.

Плохая работа связана с большим числом обусловленности матрицы A2.

Обнаружил проблему, что для А2 в LUP была норма ~. Она решалась заменой сравнения даблов обычным > сравнением через эпсилон. НО тогда в пункте 8 страдает средняя точность:

> сравнение : 8пункт ошибка ~ А2 ошибка ~.

epsilon сравнение: 8пункт ошибка ~ A2 ошибка ~10.

С чем это связано я не знаю. В программе для гаусса я оставил сравнение через >, а для LUP через эпсилон; можете в отчете по пункту 8 увидеть разницу порядков норм. Изменение варианта сравнения в гауссе ни на что не влияет.

(Сами сравнения в методах

main\_element\_by\_column (гаус, > сравнение)

main\_element\_by\_row (LUP, epsilon сравнение); класса матрицы)

10. Исследование отклонений в файле research\_A2 и research\_max\_conditional (delta это возмущение вектора b).

В графиках я выводил ошибки в файл errors, при новом запуске новые данные не затирают старые, а приписываются в конец. Т.к. для A2 очень много итераций я сохранял только первые 500 норм.

По результатам: при росте возмущения b растет и ошибка. Это понятно ведь по отклоненным данным получаются отклоненные ответы. Зачем нужно возмущать b, если как правило условие СЛАУ известно точно?

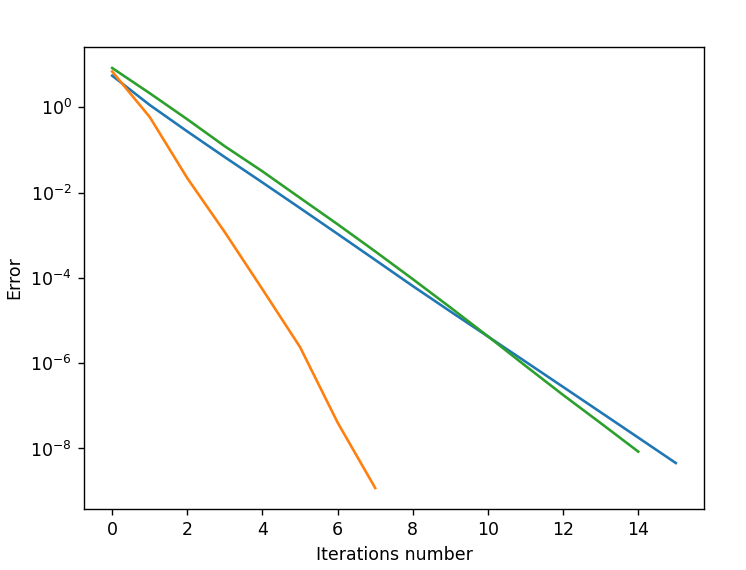
Графики строятся по данным в файле (математика не успевает скачаться до дедлайна поэтому в питоне строил (возможно неправильно); я руками вставляю в код данные из фалов и по ним строю обычный график)

Матрица с макс числом обусловленности:

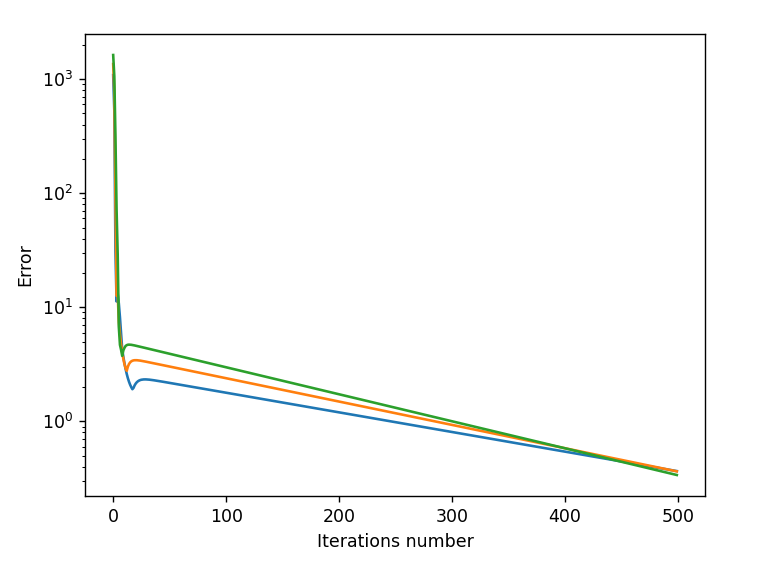
Синий: w =0.8

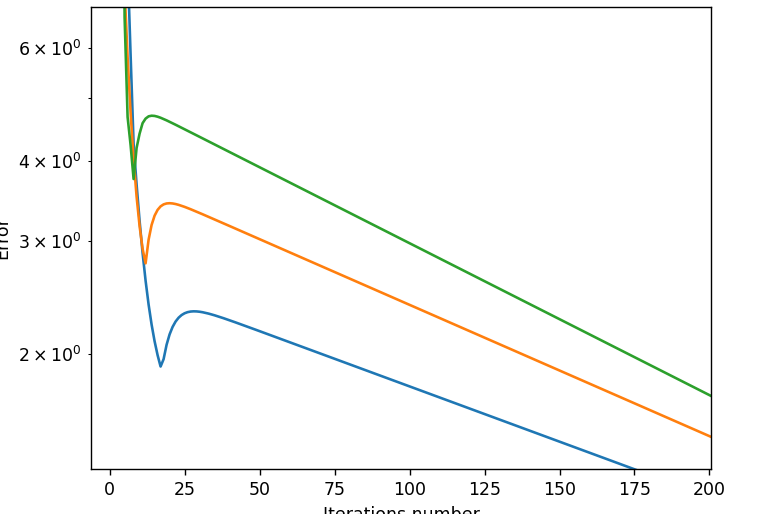
Оранжевый: w = 1

Зеленый: w = 1.2



Матрица A2 аналогично





Судя по графикам нужно выбирать коэффициент нижнего т.к. он сходиться быстрее.