# Лабораторная работа №1 по предмету Методы Вычислений

Автор: Томашевич Константин, Вариант 8.

## Часть 1

С помощью обычного rand() было бы довольно сложно и костыльно генерировать десятичные дроби з 13 знаками после запятой, RAND\_MAX же ~32000, да и rand() не такой уж и рандомный, как хотелось бы. Поэтому я решил использовать генератор случайных чисел Mersenne Twister, который используется в GMP. Код его я, конечно же, стырил, ибо конкретно он не часть лабы. Добивался наличия хотя бы 13 значащих цифр после запятой я таким образом:

1. gen <- random number in [-4, 4].
2. If fractional of gen \* 10^13 lower than 10^-4 then gen <- gen + 10^-13.
3. return gen.

## Часть 2

Пожалуй, наименее интересная часть лабораторной работы. Метод Гаусса-Жордана пишется довольно просто и топорно, но работает при этом чрезвычайно быстро :sarcasm:. Вот зачем нужно было выдумывать всякие гмресы с арнольди, если можно просто потратить всего в раз в 70 (в случае 256х256 матриц) больше времени и просто решить ? Впрочем, недавно учёные нашли Илью, который умеет считать за . Возможно, это и есть будущее алгебры, только Илья пока что отказывается делиться этим секретным алгоритмом.

Теперь по делу: число обусловленности для всех матриц колеблется в интервале [4.5; 5], что не так уж и плохо выглядит в общем случае, но вот если вспомнить, что , всё выглядит уже не так радужно. Но, к счастью, непреодолимое желание приблизиться к числу обусловленности испытывала только евклидова норма разницы реального решения и решения через МНК, с остальными как-то обошлось.

Заметим, что средний элемент матрицы таки совсем не ноль. Но мы же генерировали её рандомно! Нас обманули и мерсен твистер не работает? Таки не обманули, просто диагонали всё-таки заполнялись не рандомно (диагональное доминирование делали же), вот и не ноль.

## Часть 3

Наконец-то мы перешли к реальному решению СЛАУ. И вот у нас метод Гауса с выбором максимального элемента по всей матрице. По правде говоря, с написанием этого метода я запарился больше всего, ибо тут и порядок решений нужно в конце восстановить, и постоянно перекидывать строки и столбцы в процессе. Запарно и дебажить пришлось.

Благодаря тому, что матрицы мы, всё-таки, генерируем довольно хорошие для этого алгоритма, евклидова норма разности решений колеблется в промежутке , что довольно таки точно. Для эксперемента я попробовал отключить выбор максимального элемента по матрице, в результате чего максимальная норма поднялась до , то есть в 1000 раз! Тем самым я убедился, что такая противная операция, как выбор максимального элемента по матрице, не просто имеет право на существование, а ещё и даёт очень даже хороший результат. Но всё-таки насколько попаболь была её дебажить...

Теперь рассмотрим время выполнения, оно составляет в среднем 40мс на матрицах 256х256, что почти в 3.5 раза быстрее поиска обратной матрицы по Гаусу-Жордану. Но стоп, мы же выполняем почти те же операции! Да ещё и играем со строками и колонками (и шрифтами), строки-то можно поменять местами за О(1), а вот столбцы только за О(n). Чего тогда так быстро? А штука просто в том, что в Гаусе у нас справа один столбец b, а не целая матрица. То есть разница в скорости очевидна.

## Часть 4

После написания и дебага обычного Гауса, написать было уже не настолько сложно. Да, я думал, что LU надо тоже делать в выбором максимального элемента по матрице. Но после адского Гауса с той же болью уже не было так сложно.

Сначала поговорим про евклидову норму разности решений. Впрочем, про неё говорить, возможно, и не стоит, так как она до последней цифры совпадает с той же нормой у обычного Гауса. И удивляться тут нечему – так как мы делаем фактически то же самой, только «запоминаем» ходы Гауса в объединённую LU-матрицу (так-как свойства матриц L и U позволяют совместить их в одну для хранения) и перестановки строк и столбцов в матрицы и (которые в памяти можно хранить просто как вектора перестановки), потом просто повторяем все нужные действия с данным вектором b.

А вот что насчёт времени? строится за 39мс, на 1мс быстрее, чем решается СЛАУ па Гаусу. Почему? Но ведь при построении мы не делаем обратного хода. На решение СЛАУ с построенным разложением уходит всего 1мс, что довольно логично, ведь мы просто мучаем один столбец b. Никакой магии, конечно, не происходит, ведь 39+1=40мс – тот же Гаусс. Зато можно решать потом много СЛАУ на очень большой скорости, благодаря уже полученному разложению.

## Часть 5

Метод квадратного корня, полученный потём поиска производной от разложения Холецкого (образно говоря), позволяет довольно эффективно ускорить вычисление решения СЛАУ, матрица коэффициентов которого симметрична. Принцип очень даже прост: находим разложение Холецкого , дальше тоже, что и с LU. D тоже удобно хранить как вектор, кстати. И разложение можно также пускать на конвейер, как и LU.

Точность получилась такой же, как и у Гаусса, а именно евклидова норма находится в промежутке . Но тут следует заметить, что при поиске разложения Холецкого, в отличии от того же поиска LU, никаких лайфхаков по типу выбора максимального элемента по матрице не используется, так как это просто нарушило бы симметричную структуру матрицы. Но, тем не менее, разложение Холецкого и без этого хорошо справляется с точностью решения.

По времени решения метод квадратного корня также показал себя достаточно хорошо: 25мс против 40мс Гауссовских. Ну что, шах и мат? Нет, ведь метод квадратного корня специально создан для быстрого решения СЛАУ с **симметричными** матрицами коэффициентов, с другими его и провести, собственно, невозможно. Так что со своим делом этот метод справляется очень даже хорошо, а для всего остального можно и Гаусса использовать, при желании.

## Часть 6

Перейдём к методу релаксаций. С его доказательством я не особо знаком, ибо «на лекциях не обучался» (и в гимназиях тоже). В следствие вышеописанных факторов чрезвычайной академической неграмостности этот чрезвычайно интересный метод решения СЛАУ кажется мне магией. А ещё его и написать было очень даже проще, не то что долбанный Гаусс с выбором максимального элемента.

Сначала рассмотрим точность решения через евклидову норму разности: это будет промежуток . Собственно, поразительно точно для итерационного метода. Они же созданы для быстрого и не совсем точного получения решения путём жертвопринашения конспектов одногрупников.

Перейдём к вопросу времени. И вот тут ещё один шокирующий факт: всего 8 долбанных миллисекунд! Что это такое? Разрыв шаблонов и победа неформалов? Может где-то на фоне JS-сники с радостными воплями пишут свою ОС, которая будет «лучше линукса»? Они-то может и пишут, но с методом релаксации другая фигня. А именно штука в том, что он работает только на специфическом классе матриц (как и все класические итерационные методы), а сгенерированные по данному нам правилу матрицы вообще близки к идеальному случаю для этого метода, что вполне себе объясняет такой крутой результат. То есть метод релаксации, конечно, крут, но он именно что «широко известен(применяем) в узких кругах».

## Часть 7

## Часть 8

## Часть 9

## Часть 10