Генерация SVD происходит в файлах generate\_svd.h. Создаются вначале матрицы . Из этих матриц собирается матрица A (произведение 3 матриц).

Весь код находится в папке code:

* SVD-project v1 – версия кода, где используется только 1 тип 100 знаков после запятой из библиотеки boost, в алгоритмах со смешанной точностью используется тип double. Данная версия направлена на изучение, как размерность влияет на алгоритмы.
* SVD-project v2 d2 – версия кода, где используются boost::multiprecision::cpp\_dec\_float<Digits>, где Digits- количество знаков после запятой. Я задаю от 20 до 100 знаков. В качестве типа пониженной точности для алгоритмов использующих смешанную точность, используется boost::multiprecision::cpp\_dec\_float<Digits/2>, ровно половину знаков. Направлен на изучение влияния заданной точности на скорость работы алгоритмов.
* SVD-project v2 doulbe - версия кода, где используются boost::multiprecision::cpp\_dec\_float<Digits>, где Digits- количество знаков после запятой. Я задаю от 20 до 100 знаков. В качестве типа пониженной точности для алгоритмов использующих смешанную точность, используется тип double. Направлен на изучение влияния заданной точности на скорость работы алгоритмов.

**Предупреждение:** SVD-project v2 d2 (так как самая удачная версия по сходимости, то именно она была модифицирована, сборка проекта происходит параллельно) и SVD-project v2 doulbe (не параллельная версия), ***могут собираться очень долго!*** Параллельная версия содержит файлы test\_cases\_X\_Y.cpp, в которых объявляются типы, в общем диапазон от 20 до 100. Файлов много для того, чтобы можно было делать параллельную сборку. Без параллельной сборки, процесс может происходить до 7-8 часов, это нормально.

**Сборка:**

1. Заходим в нужную версию кода.
2. В файле CMakeLists.txt нужно поменять 2 пути:

set(CMAKE\_PREFIX\_PATH "C:/msys64/mingw64")

set(CMAKE\_IGNORE\_PATH "C:/Users/schuc/anaconda3")

Путь до mingw, 2-ой путь можно исключить. Поставил, так как был конфликт и находил библиотеки из anaconda.

1. Заходим в папку build (она должна быть пустая).
2. Я использовал сборщик ninja.
3. cmake .. -G "Ninja" -DCMAKE\_C\_COMPILER=C:\msys64\mingw64\bin\gcc.exe -DCMAKE\_CXX\_COMPILER=C:\msys64\mingw64\bin\g++.exe

Путь для gcc.exe и g++.exe прописываем свои.

1. В этой же папке прописываем: **ninja –j5.**

Количество потоков прописываем свое, мой совет подбирать вручную. Лучше отключить файл подкачки вообще. Так как, если не хватит оперативной памяти, то сборщик начнет брать из файла подкачки и сборка будет происходить около 8 часов. И смысл от параллельности теряется. 1 поток занимает около 4.5-5 гб RAM. Чем больше оперативной памяти, тем быстрее.

После у вас появляется .exe файл, его запускаем. Этот файл и есть наша программа для подсчета. После завершения программы у нас появляются таблицы, там и смотрим результаты.

**Параметры в файлах .hpp:**

* **max\_iterations** в файлах .hpp меняем по своему вкусу. Больше 10-15 нет смысла ставить, так как алгоритмы достаточно быстро сходятся кроме последнего.
* **noise\_level -** в файлах .hpp меняем по своему вкусу. Больше 1e-4 не рекомендую, так как будет недостаточно для начального приближения, все алгоритмы перестают сходиться.

**Параметры в файле testing\_all.cpp:**

* std::vector<std::pair<int, int>> matrix\_sizes\_config = {
* {50, 50},
* // {50, 40}, {100,90}, {200,180}, {500,450}, {1000,900} //
* };

Здесь прописываются размерности матрицы.

std::vector<float> sigma\_ratios\_config = {1e8f /\*, 1e3f, 1e6f, 1e9f, 1e12f, 1e15f \*/};

Здесь прописывается отношение максимального сингулярного значения к минимальному. Регулируем разреженность матрицы.

1. .

int num\_runs\_for\_averaging\_config = 1;

Количество создаваемых тестовых матриц, имеется ввиду. Если вы указали 2, то создается 2 тестовых матрицы каждой размерности, происходит расчет и результат усредняется.