ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему: Численные методы линейной алгебры

Выполнил:
Студент группы БПМИ211 _____ П.С. Шайдурова
Подпись И.О.Фамилия

25.05.2023
Дата

Принял:
Руководитель проекта Дмитрий Витальевич Трушин
Имя, Отчество, Фамилия
доцент, к.ф.-м.н.
Должность, ученое звание
ФКН НИУ ВШЭ
Место работы (Компания или подразделение НИУ ВШЭ)

Дата проверки 29.05 2023 9 _____
Оценка (по 10-ти бальной шкале) Подпись

Москва 2023

Содержание

Аннотация

Основной целью проекта является изучение вычислительно-устойчивого алгоритма по поиску сингулярного разложения матрицы, а также имплементация и тестирование библиотеки на языке C++ для работы с матрицами, включающая в себя ранее изученный алгоритм.

1 Введение

Численные методы в линейной алгебре позволяют решать матричные задачи с помощью приближенных прикладных алгоритмов. Две главные задачи, которые исследователи пытаются решить с помощью таких алгоритмов — это нахождение быстрых и устойчивых алгоритмов по поиску решений систем линейных уравнений или собственных значений матрицы.

Задача по поиску сингулярного разложения заключается в том, чтобы для любой матрицы комплексных чисел $A \in M_{n\times m}(\mathbb{C})$ найти матрицы $U \in M_{n\times n}(\mathbb{C}), V \in M_{m\times m}(\mathbb{C}), \Sigma \in M_{n\times m}(\mathbb{R})$, что $A = U\Sigma V^*$, где $UU^* = E$, $VV^* = E$, иными словами U и V — унитарные матрицы и Σ $---$ это диагональная матрица, на диагонали которой стоят в порядке убывания сингулярные значения матрицы: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0$.

В теоритической части изучен вычислительно-устойчивый алгоритм для поиска SVD. Во-первых, нужно бидиагонализовать матрицу [5, Part 2. Theorem 1] [4, Chapter 4], с помощью домножения ее с двух сторон на унитарные, получив матрицу B. Затем мы будем неявно работать с симметричной матрицей B^tB [6, Chapter 1.3]. Внутри будут использованы повороты Гивенса [4, Chapter 4]. Это будет сделано на основе QR алгоритма со сдвигом Уилкинсона [7, Lecture 29]. Само QR разложение будет найдено с помощью устойчивого алгоритма, который основан на отражения Хаусхолдера [4, Chapter 4]. В процессе будет также использована устойчивая модификация алгоритма Грама-Шмидта [7, Lecture 8].

Основной целью работы стала библиотека для работы с матрицами, в которой и реализован алгоритм по поиску сингулярного разложения, представленный в работе. В отчете также отображен дизайн архитектуры. К библиотеке написаны автоматические тесты, которые проверяют все крайние случаи. А также автоматические тесты, которые дают понять о зависимости времени работы программы от размеров матрицы. Результаты тестов на производительность отображены в отчете.

Вся работа будет размещена в github-репозитории [2].

2 Описание функциональных и нефункциональных требований к программному проекту

Необходимо реализовать библиотеку для работы с матрицами.

2.1 Функциональные требования

Библиотека должна предоставлять пользователю классы `Matrix` и `Vector`, которые будут являться абстракциями для матриц и векторов над полем $\mathbb{C}$, соответственно. Для этих классов должны быть реализованы базовые действия над ними: сложение, вычитание, сопряжение, домножение на скаляр, поэлементное умножение, матричное умножение (алгоритмы умножения матриц не являются предметом изучения этой работы, поэтому достаточно наивной имплементации за $\mathcal{O}(n^2m)$). Комплексные числа должны представляться классом `Complex`, все необходимые операции для него также должны быть реализованы. Для комплексных чисел также должен быть написан парсер. Его аргумент — строка $a \pm bi$, где $b \geq 0$, написанная без пробелов (если $a = 0$, то писать знак перед мнимой частью необязательно, как и a). Возвращаемое значение — прочитанное комплексное число.

Основной частью библиотеки будет являться реализация алгоритма поиска SVD (и вспомогательных функций), который должен быть эффективным и вычислительно устойчивым. На его вход будет подаваться единственная матрица A произвольного размера $m \times n$, а в качестве результата он должен возвращать три матрицы, образующих искомое разложение A. Аналогично, будет реализована функция, позволяющая получить по матрице ее QR-разложение, а также функция, производящая ортогонализацию Грама-Шмидта для матриц полного ранга.

К библиотеке должны быть написаны функции для автоматического тестирования всех ее основных методов на заранее подготовленных тестах.

2.2 Нефункциональные требования

- Программа должна быть написана на языке `C++` и соответствовать стандарту `c++17`
- Программа должна компилироваться компилятором, предоставляемым `GNU GCC`, запущенным с опциями `-Wall -Wextra -Werror`
- Программа не должна использовать сторонних библиотек, производящих вычисления над матрицами
- Необходимо использовать `git` в качестве системы контроля версия и `GitHub` для удаленного репозитория. История коммитов должна отображать процесс разработки проекта, все названия должны быть на английском языке
- Все комментарии должны быть на английском языке.
- Для сборки будет использоваться `Cmake` версии не ниже `3.17`
- Код должен соответствовать Google C++ Style guide [3]
- Использование библиотеки GoogleTest для написания автоматических тестов
- На компьютерах, где будет запускаться приложение или будет использоваться библиотека, должно быть хотя бы 4 гигабайта оперативной памяти

3 Теоритическая часть

3.1 Основные определения

Определение 1. Эрмитово-сопряженная матрица – это матрица $A^* \in M_{n\times m}(\mathbb{C})$, полученная из данной матрицы $A \in M_{m\times n}(\mathbb{C})$, после транспонирования и сопряжения каждого ее элемента, то есть $A^* = \overline{A^t}$

Определение 2. Унитарная матрица – это матрица $A \in M_{n\times n}(\mathbb{C})$, для которой верно $AA^* = A^*A = E$.

Определение 3. Норма вектора $v \in \mathbb{C}^n$ – это $\sqrt{\sum\limits_{i=1}^{n} \overline{v_i} \cdot v_i}$, где $v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$. Обозначается как $\|v\|_2$ или $\|v\|$.

3.2 Формальная постановка задачи

Пусть дана матрица $A \in M_{n\times m}(\mathbb{C})$. Нужно найти:

1. Матрица $U \in M_{n\times n}(\mathbb{C})$ – унитарная.
2. Матрица $V \in M_{m\times m}(\mathbb{C})$ – унитарная.
3. Матрица $\Sigma \in M_{n\times m}(\mathbb{C})$, причем $\Sigma = \begin{pmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{pmatrix}$, где вне диагонали стоят нули и вещественные числа $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq 0$.

Такие, что $U\Sigma V^* = A$ – сингулярное разложение матрицы (SVD).

3.3 Вспомогательные алгоритмы

3.3.1 Алгоритм ортонормирования Грама-Шмидта

Пусть дана система из n линейно независимых[1] векторов - $a_1, a_2, \ldots, a_n \in \mathbb{C}^m$. Нужно найти ортонормированный базис $v_1, v_2, \ldots, v_n \in \mathbb{C}$ пространства образованного данными векторами.

Рассмотрим момент, когда $v_1, v_2, \ldots, v_k$ $(k \geq 0)$ - уже ортогонализованы, то есть $\forall i \leq k : \ \|v_i\| = 1$ и $\forall i, j \leq k, i \neq j : v_i^* v_j = v_j^* v_i = 0$. Хотим добавить к этой системе вектор v_{k+1}. Рассмотрим процесс:

1. $v_{k+1}^{(0)} = a_{k+1}$
2. $i = 0, 1, \ldots, k$: $v_{k+1}^{(i)} = v_{k+1}^{(i-1)} - v_i v_i^* v_{k+1}^{(i-1)} = v_{k+1}^{(i-1)} - v_i^* v_{k+1}^{(i-1)} v_i$
3. $v_{k+1} = \frac{v_{k+1}^{(k+1)}}{\left\| v_{k+1}^{(k+1)} \right\|}$

В конце нужно нормировать v_{k+1}.

Отмечу, что в отличие от классической версии алгоритма Грама-Шмидта такая модификация вычислительно-устойчива, хотя сама реализация почти не отличается от классической версии.

Приведем псевдокод модифицированного алгоритма Грама-Шмидта.

Algorithm 1: Gram–Schmidt process

Input : Integer number n and n vectors a_i, $i = 0, 1, \cdots, n-1$, where each a_i is in $\mathbb{C}^m$ and a_i are linear independent.

Output: n vectors v_i, $i = 0, 1, \cdots, n-1$, where each element is a vector in $\mathbb{C}^m$ and v_i are orthonorma l basis of space $< a_0, a_2, \cdots, a_{n-1} >$.

```
v = a
for k ← 0 to n − 1 do
    v[k] = v[k]/abs(v[k])
    for i ← k + 1 to n − 1 do
        v[i] = v[i] − v[k]* · v[i] · v[k]
    end for
end for

return v
```

Количество операций процесса ортонормирования Грама-Шмидта системы из n векторов размером m есть $\sum\limits_{k=1}^{n} \sum\limits_{i=1}^{k-1} \mathcal{O}(m) = \mathcal{O}(n^2 m)$.

3.3.2 Отражения Хаусхолдера

Пусть дан вектор $x \in \mathbb{C}^n$. Рассмотрим унитарную матрицу $\mathrm{P} = \mathbb{E}_n - 2 \cdot u \cdot u^*$ со свойством, что

$$Px = x - u(2u^* x) = \alpha \cdot e_1 = \left.\begin{pmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{pmatrix}\right\} n \text{ times},$$

причем $\alpha \in \mathbb{R}$.

Выберем

$$u = \frac{x - \rho \|x\| e_1}{\|x - \rho \|x\| e_1\|},$$

где $|\rho| = 1$. Для устойчивости, выберем $\rho = \mathrm{sgn}(x)$ в вещественном случае и $\rho = -e^{i\phi}$, где $x = |x|e^{i\phi}$ в комплексном случае. При $x = 0$, ρ можно выбирать произвольно.

[1] Случай зависимых векторов будет разобран ниже в алгоритме нахождения QR разложения

Заметим, что если применить полученный P к матрице с первым столбцом x, то первый столбец матрицы занулится, кроме левого верхнего элемента. Обобщая, чтобы обнулить элементы $A_{l,k}, A_{l+1,k}, \ldots, A_{n,k}$ ($1 \le l \le n$ и $1 \le k \le m$) матрицы $A \in M_{n\times m}(\mathbb{C})$ нужно применить слева оператор

$$P = \begin{pmatrix} \mathbb{E}_{l-1} & 0^T \\ 0 & \mathbb{E}_{n-l+1} - 2uu^* \end{pmatrix},$$

где

$$u = \frac{x - \rho\|x\|e_1}{\|x - \rho\|x\|e_1\|}$$

и $x = \begin{pmatrix} A_{l,k} \\ A_{l+1,k} \\ \vdots \\ A_{n,k} \end{pmatrix}$, ρ определено также, как и выше.

Абсолютно аналогичные рассуждения работают со строками, если применять отражения Хаусхолдера справа.

Асимптотика одного построения отражения (то есть нахождения u) есть $\mathcal{O}(n)$.

3.3.3 QR разложение

Пусть дана матрица $A \in M_{n\times m}(\mathbb{C})$. Нужно найти унитарную матрицу $Q \in M_{n\times n}(\mathbb{C})$ и верхнетреугольную матрицу $R \in M_{n\times m}(\mathbb{C})$ такие, что $A = QR$.

Несложно заметить, что в алгоритме Грама-Шмидта вектор $v_j \in \langle v_1, v_2, \ldots, v_{j-1}, a_j\rangle$, тогда $a_j \in \langle v_1, v_2, \ldots, v_j\rangle$. Значит $[a_1, a_2, \ldots, a_n] = [v_1, v_2, \ldots, v_n] \cdot R$, где R - верхнетругольная матрица.

Но у произвольной матрицы A столбцы могут быть не линейно независмы. Тогда во время алгоритма Грама-Шмидта вместо v_i получится 0, если этот вектор лежит в $\langle v_1, v_2, \ldots, v_{i-1}, a_i\rangle$, а тогда и a_i лежит в $\langle v_1, v_2, \ldots, v_i\rangle$. Через v_i не нужно считать следующие вектора.

В процессе ортонормирования Грама-Шмидта обозначим за $r_{i,k+1} = v_i^* v_{k+1}^{(i-1)}$, если v_i получилось ненулевым, а иначе $r_{i,k+1} = 0$. Кроме того, $r_{i,i} = \left\|v_{k+1}^{(k+1)}\right\|$. Тогда из алгоритма следует, что $a_k = \sum\limits_{i=1}^{k} v_i \cdot r_{i,k}$ и $(r_{i,j})$ – искомая верхнетреугольная матрица.

Получилось, что $A = [v_1, v_2, \ldots, v_n] \cdot R_0$, но могут быть нулевые вектора среди $v_1, v_2, \ldots, v_n$, а значит матрица не унитарная. Просто уберем все нулевые вектора из матрицы $Q_0 = [v_1, v_2, \ldots, v_n]$, а соответсвенно и все нулевые строки из матрицы R_0. Получатся матрицы $Q_1 \in M_{n\times k}(\mathbb{C}), R_1 \in M_{k\times m}(\mathbb{C})$, где $A = Q_1 \cdot R_1$.

Только что найденные матрицы будут не того размера. Дополним их:

$$R = \left.\begin{pmatrix} & R_1 & \\ \hline 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{pmatrix}\right\} n - k \text{ times}$$

Кроме того, пусть $Q_1 = [v_1, v_2, \ldots, v_k]$. Дополним вектора $v_1, v_2, \ldots, v_k$ до базиса R^n. Для этого добавим все вектора стандартного базиса, а затем ортонормируем полученные $n+k$ векторов алгоритмом Грама-Шмидта, не изменяя первые k векторов. Все полученные вектора и будут составлять матрицу $Q \in M_{n\times n}(\mathbb{C})$.

Обощая все выше сказанное, приведем псевдокод:

Algorithm 2: QR decomposition

Input : Integer numbers n, m and matrix $A \in M_{n\times m}(\mathbb{C})$.
Output: Unitary matrix $Q \in M_{n\times n}(\mathbb{C})$ and upper-diagonal matrix $R \in M_{n\times m}(\mathbb{C})$ where $A = QR$.

```
Q = [], R = [[0]]

for k ← 0 to n − 1 do
    Q[k] = A[k]                                  // A[i] is a i-th column of A.
end for
```

Algorithm 2: QR decomposition(continued)

```
for k ← 0 to n − 1 do
    if abs(Q[k]) ≠ 0 then
        R[k][k] = abs(Q[k])
        Q[k] = Q[k]/abs(Q[k])
    else
        continue
    end if

    for i ← k + 1 to n − 1 do
        if abs(Q[i]) ≠ 0 then
            R[k][i] = Q[k]* · Q[i]
            Q[i] = Q[i] − R[k][i] · Q[k]
        end if
    end for
end for
/* Delete columns of matrix Q that consist only of zeros.                    */
eraseNullColumns(Q)
/* Delete rows of matrix R that consist only of zeros.                       */
eraseNullRows(R)
/* Add zero rows in matrix R to the bottom of the matrix so that there are n rows in total. */
addNullRows(R, n)
/* Complements vectors to an orthonormal basis of ℂ^n, if it possible.       */
completeOrthonormalBasis(Q, n)
return Q, R
```

Асимптотика равна асимптотике Грама-Шмидта, то есть $\mathcal{O}(mn^2)$.

Но в коде будут использоваться отражения Хаусхолдера. Будем просто занулять столбцы в данной матрице. Напишем псевдокод:

Algorithm 4: QR decomposition with Householder reflections (continued)

Input : Integer numbers n, m and matrix $A \in M_{n\times m}(\mathbb{C})$.
Output: Unitary matrix $Q \in M_{n\times n}(\mathbb{C})$ and upper-diagonal matrix $R \in M_{n\times m}(\mathbb{C})$ where $A = QR$.

```
Q = 𝔼_n
for i ← 0 to min(n − 1, m − 1) do
    /* function leftReflection(A, l, k) return Householder reflection (E_n − 2uu*) that zeroes
       A_{l+1,k}, A_{l+2,k}, ..., A_{n,k}                                    */
    P = leftReflection(A, i, i)
    Q = P · Q
    A = P · A
end for

return Q*, R
```

Время работы такого алгоритма есть $\mathcal{O}(n \cdot n^2)$, так как построение одного отражения Хаусхолдера работает за $\mathcal{O}(n)$, а затем полученный вектор умножается на матрицу размеров $n \times m$ и $n \times n$.

3.3.4 Бидиагонализация Голуба-Кахана

Пусть дана матрица $A \in M_{n\times m}(\mathbb{C})$. Нужно привести матрицу A в бидиагональную матрицу

$$B = \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ & \alpha_2 & \beta_2 & & \\ & & \ddots & \ddots & \\ & & & \alpha_{k-1} & \beta_{k-1} \\ & & & & \alpha_k \\ & & & & \end{pmatrix} \in M_{n\times m}(\mathbb{R}),$$

где $k = \texttt{min}(n, m)$, и $A = UBV^*$, где $U \in M_{n\times n}(\mathbb{C}), V \in M_{m\times m}(\mathbb{C})$ - унитарные матрицы.

Воспользуемся опять отражениями Хаусхолдера. Рассмотрим соответсвующий процесс на примере матрицы 4×5.

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix} \xrightarrow{P_1^*\cdot} \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix} \xrightarrow{\cdot Q_1} \begin{pmatrix} \times & \times & 0 & 0 & 0 \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix} \xrightarrow{P_2^*\cdot} \begin{pmatrix} \times & \times & 0 & 0 & 0 \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix} \xrightarrow{\cdot Q_2}$$

$$A \qquad P_1^*A \qquad P_1^*AQ_1 \qquad P_2^*P_1^*AQ_1$$

$$\xrightarrow{\cdot Q_2} \begin{pmatrix} \times & \times & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix} \xrightarrow{P_3^*\cdot} \begin{pmatrix} \times & \times & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{pmatrix} \xrightarrow{\cdot Q_3} \begin{pmatrix} \times & \times & 0 & 0 & 0 \\ 0 & \times & \times & 0 & 0 \\ 0 & 0 & \times & \times & 0 \\ 0 & 0 & 0 & \times & \times \end{pmatrix}$$

$$P_2^*P_1^*AQ_1Q_2 \qquad P_3^*P_2^*P_1^*AQ_1Q_2 \qquad P_3^*P_2^*P_1^*AQ_1Q_2Q_3$$

Причем P_1, P_2, P_3 и Q_1, Q_2, Q_3 – это отражения Хаусхолдера, которые зануляют соответствуюзие столбцы/строки.

Заметим, что элементы на диагонали и субдиагонали получаются вещественными в силу определния отражений Хаусхолдера.

Тогда псевдокод будет следующим:

Algorithm 5: Golub-Kahan bidiagonalization

Input : Integer numbers n, m and matrix $A \in M_{n\times m}(\mathbb{C})$.
Output: Matrices $U \in M_{n\times n}(\mathbb{C})$, $V \in M_{m\times m}(\mathbb{C})$ and vectors α, β, such that $A = UBV^*$ where B is bidiagonal with diagonals equal to α and β

$V = \mathbb{E}_n, U = \mathbb{E}_m$
$diagonal = []$
$subdiagonal = []$
for $i \leftarrow 0$ **to** $\texttt{min}(n-1, m-1)$ **do**
  /* function $leftReflection(A, l, k)$ return matrix of Householder reflection $(E_n - 2uu^*)$ that zeroes $A_{l+1,k}, A_{l+2,k}, \ldots, A_{n-1,k}$ and new number in place of $A_{l,k}$ */
  $P, \alpha = \texttt{leftReflection}(A, i, i)$
  $diagonal.\texttt{push}(\alpha)$
  $V = P \cdot V$
  $A = P \cdot A$
  if $i + 1 < m$ **then**
    /* function $rightReflection(A, l, k)$ return matrix of Householder reflection $(E_n - 2u^*u)$ that zeroes $A_{k,l+1}, A_{k,l+2}, \ldots, A_{k,m-1}$ and new number in place of $A_{k,l}$ */
    $Q, \beta = \texttt{rightReflection}(A, i, i+1)$
    $subdiagonal.\texttt{push}(\beta)$
    $U = U \cdot Q$
    $A = A \cdot Q$
  end if
end for
return $diagonal, subdiagonal, V^*, U$

Время работы алгоритма такое же, как и в случае QR разложения (с учетом того, что делаются преобразования и слева, и справа), то есть $\mathcal{O}(\texttt{min}(n, m) \cdot (n^2 + m^2))$

3.3.5 QR алгоритм со сдвигом

Пусть дана симметричная тридиагональная матрица $A \in M_{n\times n}(\mathbb{R})$. Хотим найти ее разложение $A = QDQ^t$, где $Q \in M_{n\times n}(\mathbb{R})$ – унитарная матрица и матрица $D \in M_{n\times n}(\mathbb{R})$ – диагональная матрица, состоящая из собственных значений матрицы A.

Если матрица A - это уже диагональная матрица, то $Q = E$ и $D = A$.

Иначе рассмотрим итеративный процесс. Пусть $A^{(0)} = A$. Нужно найти $A^{(k)}$ из $A^{(k-1)}$. Пусть

$$\mu_k = A_{nn}^{(k-1)} - \frac{\operatorname{sgn}(\sigma)\left(A_{n,n-1}^{(k-1)}\right)^2}{|\sigma| + \sqrt{\sigma^2 + \left(A_{n,n-1}^{(k-1)}\right)^2}},$$

где $\sigma = \frac{1}{2}\left(A_{n-1,n-1}^{(k-1)} - A_{nn}^{(k-1)}\right)$ и $\operatorname{sgn}(0) = 1$. Найдем QR разложение матрицы $A^{(k-1)} - \mu_k E = Q^{(k)}R^{(k)}$. Тогда $A^{(k)} = R^{(k)}Q^{(k)} + \mu_k E$.

Несложно заметить, что

$$A^{(k)} = \left(Q^{(k)}\right)^t \left(A^{(k-1)} - \mu_k E\right) Q^{(k)} + \mu_k E.$$

Откуда

$$A^{(k-1)} = Q^{(k)} A^{(k)} \left(Q^{(k)}\right)^t.$$

Такими действиями можно восстановить изначальное разложение для A.

Теперь если в $A_{(k)}$ для какого-то j верно, что $A_{j,j+1}^{(k)}$ достаточно близко к нулю, то скажем, что $A_{j,j+1}^{(k)} = A_{j+1,j}^{(k)} = 0$, чтобы получить $A^{(k)} = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$, а затем продолжим алгоритм для $A_1 = Q_1 D_1 Q_1^t$ и $A_2 = Q_2 D_2 Q_2^t$, найдя их разложения, а затем найти решения и для матрицы $A_{(k)}$, сделав $Q^{(k)} = \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}$ и $D^{(k)} = \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix}$

Algorithm 6: QR algorithm with Wilkinson shift

Input : Integer numbers n, symmetric tridiagonal matrix $A \in M_{n\times n}(\mathbb{R})$ and float number ε – precision.
Output: Unitary matrix $C \in M_{n\times n}(\mathbb{R})$ and diagonal matrix $D \in M_{n\times n}(\mathbb{R})$ where $A = CDC^t$.

```
C = 𝔼_n                                                    // E_k is a unit square matrix k × k
while !isDiagonal(A) do
    /* Calculate Wilkinson shift                                                          */
    σ = (A[n−1][n−1] − A[n][n])/2
    μ = A[n][n] − (sign(σ) · (A[n][n−1])²)) / (abs(σ) + √(σ² + (A[n][n−1])²))

    Q, R = QRDecomposition(A − μ · 𝔼_n)
    A = RQ + μ𝔼_n
    C = C · Q

    for k ← 0 to n − 1 do
        if A[k][k+1] < ε then
            A[k][k+1] = 0
            A[k+1][k] = 0
            A_1, A_2 = splitMatrix(A, k+1, n)        // Split matrix to sum of two symmetric tridiagonal
             matrices where the first one is a upper left corner matrix (k+1) × (k+1) and the second
             one is a down right corner matrix (n−k+1) × (n−k+1)
            C_1, D_1 = QRAlgorithm(k+1, A_1, ε)                          // recursive call
            C_2, D_2 = QRAlgorithm(n−k+1, A_2, ε)                        // recursive call
            A = mergeMatrix(D_1, D_2)                                    // A = (D_1 0; 0 D_2)
            C_3 = mergeMatrix(C_1, C_2)                                  // C_3 = (C_1 0; 0 C_2)
            C = C · C_3
            break
        end if
    end for
end while
return C, A
```

В этом алгоритме можно использовать $\mu_k = 0$ — в этом случае получим классическую версию QR-алгоритма (без сдвига). Однако в этом случае сходимость будет очень медленной или даже отсутсвовать. Если же брать μ_k как описано выше, то алгоритм будет иметь доказуемую кубическую сходимость (квадратичную в худшем случае), то есть элементы на диагонали будут кубически сходиться к собсвтенным значениям матрицы. Отметим отдельно, что на практике алгоритм всегда сходится. Тогда, так как одна итерация алгоритма работает за $\mathcal{O}(n^3)$, то весь QR-алгоритм будет занимать $\mathcal{O}(n^3|\log\varepsilon|)$ времени.

3.3.6 Вращения Гивенса

Пусть дан вектор $x \in \mathbb{R}^n = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ и два индекса i и j $(1 \le i, j \le n)$. Нужно найти такую унитарную матрицу $G \in M_{n\times n}(\mathbb{R})$, что $G^t x = \begin{pmatrix} \widetilde{x_1} \\ \widetilde{x_2} \\ \vdots \\ \widetilde{x_w} \end{pmatrix}$, где $\widetilde{x_k} = \begin{cases} \text{произвольное}, & k = i \\ 0, & k = j \\ x_k & \text{иначе} \end{cases}$.

Будем использовать матрицу

$$G = \begin{pmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & c & \dots & s & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & -s & \dots & c & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{pmatrix},$$

которая отличается от единичной в четырех индексах ($G_{i,i} = c$, $G_{j,j} = c$, $G_{i,j} = s$ и $G_{j,i} = -s$). Тогда должно быть выполнено, что

$$\begin{pmatrix} c & -s \\ s & c \end{pmatrix} \cdot \begin{pmatrix} x_i \\ x_j \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}.$$

Исходя из того, что G — унитарная матрица (то есть $c^2 + s^2 = 1$) найдем

$$c = \frac{x_i}{\sqrt{x_i^2 + x_j^2}} = \frac{1}{\sqrt{1 + \frac{x_j^2}{x_i^2}}} = \frac{x_i}{x_j}\frac{1}{\sqrt{1 + \frac{x_i^2}{x_j^2}}}$$

и

$$s = \frac{-x_j}{\sqrt{x_i^2 + x_j^2}} = \frac{x_j}{x_i}\frac{1}{\sqrt{1 + \frac{x_j^2}{x_i^2}}} = \frac{1}{\sqrt{1 + \frac{x_j^2}{x_i^2}}}.$$

Вычислительно устойчивое вычисление основывается на формлуах, где фигурирует отношения x_i и x_j.

Algorithm 7: Givens rotation

Input : Two real numbers a, b and real number ε — precision

Output: Two reals numbers c, s that $c^2 + s^2 = 1$ and $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}^t \cdot \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}$

```
if abs(b) < ε then
    return 1.0, 0.0
end if
if abs(b) > abs(a) then
    t = −a/b
    s = 1/√(1 + t²)
    c = ts
    return c, s
end if
```

Algorithm 7: Givens rotation(continued)

$t = -b/a$

$c = 1/\sqrt{1+t^2}$

$s = tc$

return $c,\ s$

Время нахождения c, s составляет $\mathcal{O}(1)$, но если мы хотим найти матрицу, то будет уже $\mathcal{O}(n^2)$.

Заметим, что матрицу можно применять и к столбцу какой-то матрицы, чтобы занулить определенное значение.

3.3.7 QR алгоритм для бидиагональных матриц

Пусть $J_0 = B \in M_{n\times n}(\mathbb{R})$ – квадратная бидиагональная матрица. Будем делать итерационный процесс: $J^{(i+1)} = S^{(i)t} J^{(i)} T^{(i)}$ – бидиагональная матрица, где $S^{(i)}, T^{(i)}$ – унитраные матрицы. Такая последовательность сойдется к диагональной матрице.

Рассмотрим симметричную тридиагональную матрицу $M_i = J^{(i)t} \cdot J^{(i)}$. Выше было показано, что применения QR алгоритма со сдвигом Вилкинсона диагонализует эту матрицу. Рассмотрим процесс, который математически будет аналогичен одной итерации QR алгоритма со сдвигом, но не будем явно считать матрицу M_i.

Будем делать "chasing" с помощью поворотов Гивенса для бидиагональной матрицы $J = J^{(i)}$. Первый поворот будет применен для чисел $J_{0,0}^2 - s$ (s — сдвиг Вилкинсона, можно посчитать без подсчета матрицы M_i) и $J_{0,0} \cdot J_{0,1}$, то есть

$$T_1 = \begin{pmatrix} c & s & 0 & \dots & 0 \\ -s & c & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix},$$

где $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}^t \cdot \begin{pmatrix} J_{0,0}^2 - s \\ J_{0,0} \cdot J_{0,1} \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}$.

Тогда $JT_1 = \begin{pmatrix} \times & \times & & & & \\ & \times & \times & & & \\ & & \ddots & \ddots & & \\ & & & \times & \times & \\ & & & & \times \end{pmatrix} T_1 = \begin{pmatrix} \times & \times & & & & \\ \times & \times & \times & & & \\ & & \ddots & \ddots & & \\ & & & \times & \times & \\ & & & & \times \end{pmatrix}$.

Следующие повороты Гивенса будут зануять полученные элементы, которые не лежат на главной диагонали или на диагонали над ней. Рассмотрим на примере матрицы 5×5:

$$\begin{pmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & & \times & \times & \\ & & & \times & \times \\ & & & & \times \end{pmatrix} \xrightarrow{S_1^t \cdot} \begin{pmatrix} \times & \times & \times & & \\ & \times & \times & & \\ & & \times & \times & \\ & & & \times & \times \\ & & & & \times \end{pmatrix} \xrightarrow{\cdot T_2} \begin{pmatrix} \times & \times & & & \\ & \times & \times & & \\ & \times & \times & \times & \\ & & & \times & \times \\ & & & & \times \end{pmatrix} \xrightarrow{S_2^t \cdot} \begin{pmatrix} \times & \times & & & \\ & \times & \times & \times & \\ & & \times & \times & \\ & & & \times & \times \\ & & & & \times \end{pmatrix} \xrightarrow{\cdot T_3}$$

JT_1 $\qquad$ $S_1^t JT_1$ $\qquad$ $S_1^t JT_1 T_2$ $\qquad$ $S_2^t S_1^t JT_1 T_2$

$$\xrightarrow{\cdot T_3} \begin{pmatrix} \times & \times & & & \\ & \times & \times & & \\ & & \times & \times & \\ & & \times & \times & \times \\ & & & & \times \end{pmatrix} \xrightarrow{S_3^t \cdot} \begin{pmatrix} \times & \times & & & \\ & \times & \times & & \\ & & \times & \times & \times \\ & & & \times & \times \\ & & & & \times \end{pmatrix} \xrightarrow{\cdot T_4} \begin{pmatrix} \times & \times & & & \\ & \times & \times & & \\ & & \times & \times & \\ & & & \times & \times \\ & & & \times & \times \end{pmatrix} \xrightarrow{S_4^t \cdot} \begin{pmatrix} \times & \times & & & \\ & \times & \times & & \\ & & \times & \times & \\ & & & \times & \times \\ & & & & \times \end{pmatrix}$$

$S_2^t S_1^t JT_1 T_2 T_3$ $\qquad$ $S_3^t S_2^t S_1^t JT_1 T_2 T_3$ $\qquad$ $S_3^t S_2^t S_1^t JT_1 T_2 T_3 T_4$ $\qquad$ $S_4^t S_3^t S_2^t S_1^t JT_1 T_2 T_3 T_4$

Причем S_i, T_i — вращения Гивенса, которые зануляют соотвествуюзий элемент. T_i зануляет элемент $\{i-1, i+1\}$ при $i > 1$. S_i зануляет элемент $\{i+1, i\}$.

Обобщая,

$$J^{(i+1)} = S_{n-1}^t S_{n-2}^t \dots S_1^t J^{(i)} T_1 \dots T_{n-2} T_{n-1}.$$

Откуда $S^{(i)} = S_1 \dots S_{n-2} S_{n-1}$ и $T^{(i)} = T_1 \dots T_{n-2} T_{n-1}$.

Отметим, что для вычислительной устойчивости и сходимости необходимы следующие действия

- Если на субдиагонали матрицы $J^{(i)}$ есть элемент по модулю меньше точности, то матрицу можно разделить на две и выполнять алгоритм для них независимо [2];
- Если на диагонали $J^{(i)}$ есть элемент по модулю меньше точности, то нужно занулить всю строку с этим элементом, чтобы матрицу можно было разбить на две. [3]

Algorithm 8: QR algorithm with Wilkinson shift to bidiagonal matrices

Input : Integer numbers n, bidiagonal matrix $A \in M_{n\times n}(\mathbb{R})$ and real number ε — precision
Output: Unitary matrices $V \in M_{n\times n}(\mathbb{R}), U \in M_{n\times n}(\mathbb{R})$ and diagonal matrix $D \in M_{n\times m}(\mathbb{R})$ where $A = UDV^t$.

$V = \mathbb{E}_n$ // $\mathbb{E}_k$ is a unit square matrix $k \times k$
$U = \mathbb{E}_n$ // $\mathbb{E}_k$ is a unit square matrix $k \times k$

while !isDiagonal(A^tA) **do**
  /* check if subdiagonal elements less than ε */
  if trySplit(A, U, V, ε) **then**
    return V, U, A
  end if
  for $i \leftarrow 0$ **to** $n-2$ **do**
    if abs$(A(i,i)) < \varepsilon$ **then**
      /* zeros all elements of row i to zero */
      zerosRow$(A, i, U, V, \varepsilon)$
      trySplit(A, U, V, ε)
      return V, U, A;
    end if
  end for

  $\mu =$ WilkinsonShift(A)
  for $i \leftarrow 0$ **to** $n-2$ **do**
    if $i = 0$ **then**
      $c, s =$ GivensRotation$(A[0][0] * A[0][0] - s, A[0][0] * A[0][1])$
      /* $A* = T$ */
      $A = (A, c, s, 0, 1)$
      /* $V* = T$ */
      $V = (A, c, s, 0, 1)$
    else
      $c, s =$ GivensRotation$(A[i-1][i], A[i-1][i+1])$
      /* $A* = T$ */
      $A = (A, c, s, i, i+1)$
      /* $V* = T$ */
      $V = (A, c, s, i, i+1)$
    end if

    $c, s =$ GivensRotation$(A[i][i], A[i+1][i])$
    /* $A = S^t * A$ */
    $A =$ multplyLeftToGivensRotation$(A, c, s, i, i+1)$
    /* $U = S^t * U$ */
    $U =$ multplyLeftToGivensRotation$(A, c, s, i, i+1)$
  end for
end while
return V, U, A

[2] Более подробно можно посмотреть в разделе с QR алгоритмом со сдвигом

[3] Более подробно, если $|J^{(i)}_{k,k}| < \varepsilon$ будем применить вращения G^t_i слева, зануляя сначала $\{k, k+1\}$ (второй элемент — $\{k+1, k+1\}$), потом $\{k, k+2\}$ (второй элемент — $\{k+2, k+2\}$) и т.д.

Количество итераций необходиомое для сходимости такое же как и у QR алгоритма со сдвигом Вилкинсона. Внутри QR алгоритма одна итерация будет занимать $\mathcal{O}(n^3)$, учитывая, что на матрицы Гивенса можно умножать за $\mathcal{O}(n)$, откуда самая дорогая операция — разделение матрицы на несколько частей.

3.4 Нахождение сингулярного разложения

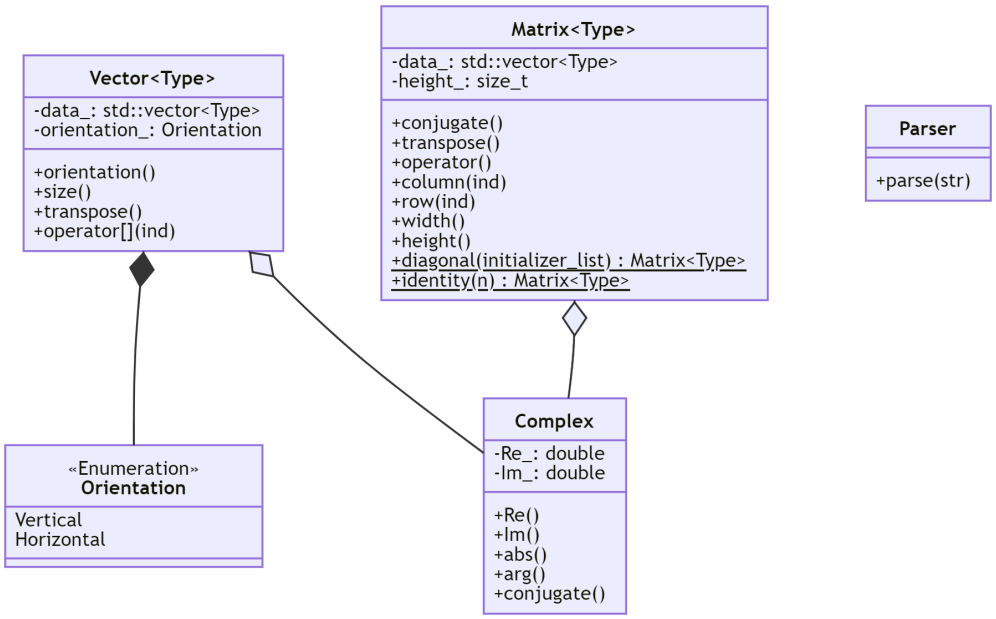
Дана матрица $A \in M_{n\times m}(\mathbb{C})$.

1. Считаем, что $n > m$, иначе сопрягаем матрицу и вызываемся рекурсивно.
2. Бидиагонализация матрицы A с помощью алгоритма бидиагонализации Голуба-Кахана. Получится $A = U_1 B V_1^t$, где $U_1 \in M_{n\times n}(\mathbb{C})$, $V_1 \in M_{m\times m}(\mathbb{C})$ и $B \in M_{n\times m}(\mathbb{R})$.
3. Делаем матрицу $B_1 \in M_{\times m}(\mathbb{R})$ путем отрезания у матрицы B последних $n - m$ нулевых строк.
4. Запускаем QR алгоритм для бидиагональных матриц и получаем $B_1 = U_2 D V_2^t$, где $U_2, V_2 \in M_{m\times m}(\mathbb{R})$ и $D \in M_{m\times m}(\mathbb{R})$ – диагональная матрица.
5. Сортируем сингулярные значения, меняя не только порядок элементов на диагонали, но и столбцы в матрицах U_2, V_2.
6. Дополняем U_2 до матрицы $M_{n\times n}(\mathbb{R})$ путем добавления блочной матрицы $\mathbb{E}_{n-m}$ к правому нижнему углу матрицы.
7. $U = U_2 \cdot U_1$ и $V = V_2 \cdot V_1$.

4 Архитектура библиотеки

4.1 Базовые классы

Библиотека предоставляет несколько базовых классов для работы с линейной алгеброй. Общая структурная диаграмма приведена ниже:



Структурная диаграмма классов библиотеки

Все классы размещены в пространстве имен `svd_computation`. Ниже приведено их более подробное описание.

4.1.1 Class Complex

Наиболее базовым из предоставляемых является класс `Complex`, позволяющий работать с комплексными числами. В качестве внутреннего типа для хранения вещественной/мнимой частей используются вещественные числа (тип `long double`). Для удобной работы с классом для него переопределены операторы ввода и вывода. Для класса реализованы все базовые арифметические операции, взятие мнимой и вещественной частей, взятие модуля и аргумента, а также:

- in-place метод `conjugate()`, берущий комплексное сопряжение к числу, и соответсвующая ему не in-place функция `conjugate(Complex)`
- Функция `sqrt(Complex)`, вычисляющая корень (возвращает значение из той ветви, где arg $\in [0, \pi)$

Пример работы с классом:

```
Complex a;
std::cin >> a; // Sample input: "1-2i"
Complex b = Complex(2, 4); // 2 + 4i
Complex c = b * conjugate(a) + b.abs();
std::cout << a << ' ' << b << ' ' << c << "\n"; // Sample output: "1-2i 2+4i -1.52786+8i"
```

4.1.2 Class Parser

Вспомогательным к предыдущему (и недоступным для пользователя) является класс `Parser` с единственным методом `parse()`, преобразующим переданную строку в объект класса `Complex`. Для успешного парсинга строка должна содержать запись комплексного числа в форме $x \pm yi$, где x, y — вещественная и мнимая части числа, соответственно. При этом если вещественная чать равна нулю, то она может быть опущена. Этот парсер используется исключительно в переопределенном операторе ввода. Возможные корректные входные строки:

`"2+3i"`, `"2+3*i"`, `3.4+5i"`, `"0.2"`, `"0.3-4.5i"`

4.1.3 Class Vector

Этот класс позволяет работать с векторами. Он является шаблонным и параметризован типом содержащихся в нем элементов `Type`. От `Type` ожидается реализация стандартных арифметических операций. Также, каждый вектор имеет *ориентацию*, которую можно передать в конструктор и получить через метод `orientation()`. Она равна либо `Vertical` (по умолчанию), либо `Horizontal` и позволяет различать вектора-столбцы и вектора-строки. В классе реализованы:

- Стандартные арифметические операции: сложение, вычитание, умножение на скаляр и др.
- Обращение по индексу через оператор `[]`
- Метод `size()`, возвращающий длину вектора.
- Перегрузки операторов ввода и вывода
- Метод `transpose()`, меняющий ориентацию вектора. Этот метод работает in-place, и в дополнение к нему есть функция `Vector<Type> transpose(Vector<Type>)`, работающая не in-place
- Статический метод `Vector standart_basis(ind, len, orientation)`, генерирующий вектор заданной длины и ориентации, где на позиции `ind` стоит единица, а на всех остальных позициях стоят нули

Помимо этого, для корректной работы с другими алгоритмами, пользователю необходимо определить функции `long double abs(Vector<Type>)` и `Type dot_product(Vector<Type>, Vector<Type>)` (для `Type=long double` и `Type=Complex` реализации уже содержатся в библиотеке). Пример работы с классом:

```
Vector<long double> v;
std::cin >> v; // Sample input: "2\n1 -1"
Vector<long double> u({1.5, 2.5}, Vector<long double>::Horizontal);
std::cout << (u + transpose(v)) * 1.5l << ' ' << dot_product(u, v) << "\n";
 // Sample output: "3.75 2.25 -1"
```

4.1.4 Class Matrix

Класс для работы с матрицами. Также как и вектор, является шаблонным по типу хранящихся в матрице элементов `Type`. Помимо стандартных арифметических операций для него ализованы:

- методы `height()` и `width()`, возвращающих количество строк и столбцов в матрице, соответственно
- методы `Vector<Type> row(ind)` и `Vector<Type> column(ind)`, возвращающих строку/столбец матрицы по индексу
- переопределенный оператор `()(row, column)` для обращения к конкретному элементу матрицы по номерам его строки и столбца
- статический метод `identity(n)`, возвращающий единичную матрицу размера $n \times n$
- статический метод `diagonal(std::vector<Type> diagonal)`, строящий матрицу, в которой на главной диагонали стоят переданные значения, а все остальные элементы равны 0
- in-place методы `transpose()` и `conjugate()`, производящие транспонирование и сопряжение матрицы. Оба метода имеют не in-place функции-аналоги `transpose(Matrix<Type>)` и `conjugate(Matrix<Type>)`

Пример работы с классом `Matrix`:

```
1 Matrix<long double> v({{-1, 1}, {1, -1}});
2 Matrix<long double> u({{2, 3}, {4, 5}});
3 std::cout << (transpose(v) * u).row(0) << "\n"; // Will print "2 2"
```

4.2 Алгоритмы

Основные алгоритмы библиотеки определены вне классов в том же пространстве имен `svd_computation`. Среди них (для краткости шаблонные параметры классов не указаны):

- `Matrix bidiagonalize(Matrix A, Matrix* left_basis, Matrix* right_basis)` — принимает матрицу A и возвращает ее бидиагонализированную форму (стоит отметить, что матрица A может быть параметризована любым типом, при этом возвращаемым значением всегда будет матрица вещественных чисел). Также функция записывает левый и правый базисы в соответствующие указатели.
- `void orthonormalize(std::vector<Vector> &system)` — ортонормирует переданную систему векторов (алгоритм принимает систему по ссылке и модифицирует ее).
- `std::pair<Matrix, Matrix> get_QR_decomposition(Matrix A)` — выполняет QR-разложение над матрицами, возвращает пару из матриц Q и R (см. описание алгоритма в теории).
- `long double left_reflection(Matrix& A,int row, int column, Matrix* left_basis)` — производит левое отражение Хаусхолдера над переданной матрицей, модифицируя ее. Аналогичная функция для правого разложения называется `right_reflection`
- `pair<long double, long double> get_givens_rotation(a, b)` — возвращает коэффициенты c и s вращения Гивенса для заданных значений x_i и x_j (см. соответствующий раздел в теории).
- `Matrix compute_svd(Matrix A, Matrix* left_basis, Matrix* right_basis)` — выполняет SVD-разложение матрицы, возвращая матрицу Σ, и записывая матрицы U и V в переданные указатели.

Времена работы всех перечисленных алгоритмах соответствуют данным в теории оценкам.

5 Тестирование библиотеки

5.1 Юнит-тесты

Все основные функции библиотеки, а также некоторые методы классов `Complex`, `Vector` и `Matrix` покрыты юнит-тестами. Для их запуска в `Cmake` есть отдельная цель `course_project_test`.

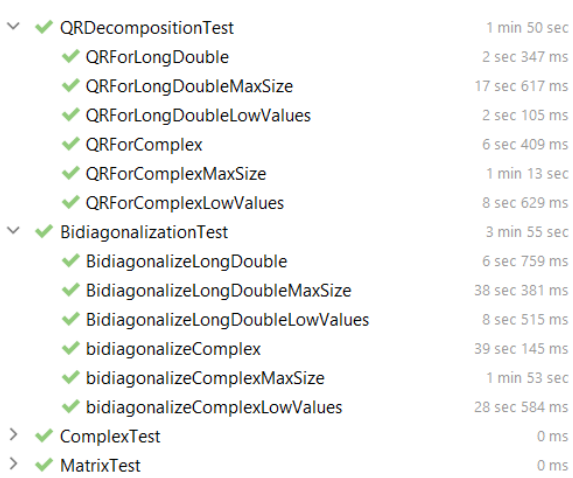


Рис. 1: Юнит тесты, 1 часть

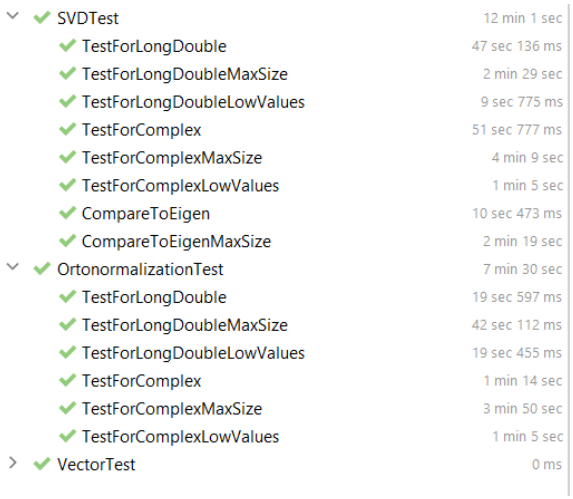


Рис. 2: Юнит тесты, 2 часть

5.2 Показатели алгоритма SVD-разложения

В алгоритме передается переменная, которая отвечает за точность — эпсилон. Провелось тестирование, которое смотрит на зависимость точности (в сравнении с исходной матрицей) от этого эпсилона[4]:

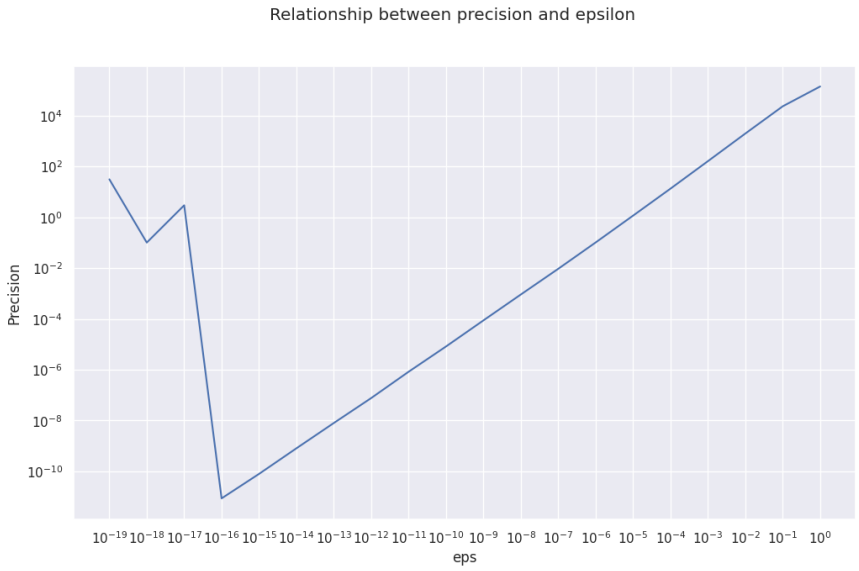


Рис. 3: Связь между выбранным эпсилоном и точностью

Для всех дальнейших тестов использовался константа точности равная 10^-16.

Кроме того, еще одна константа кода — это максимальное число операций в QR-алгоритме. Ниже приведен график зависимости точности (по сравнению с исходной матрицей) и ограничением на число итераций QR-алгоритма:

[4]Все значения в этом и других графиках построены как среднее значение от 100 запусков

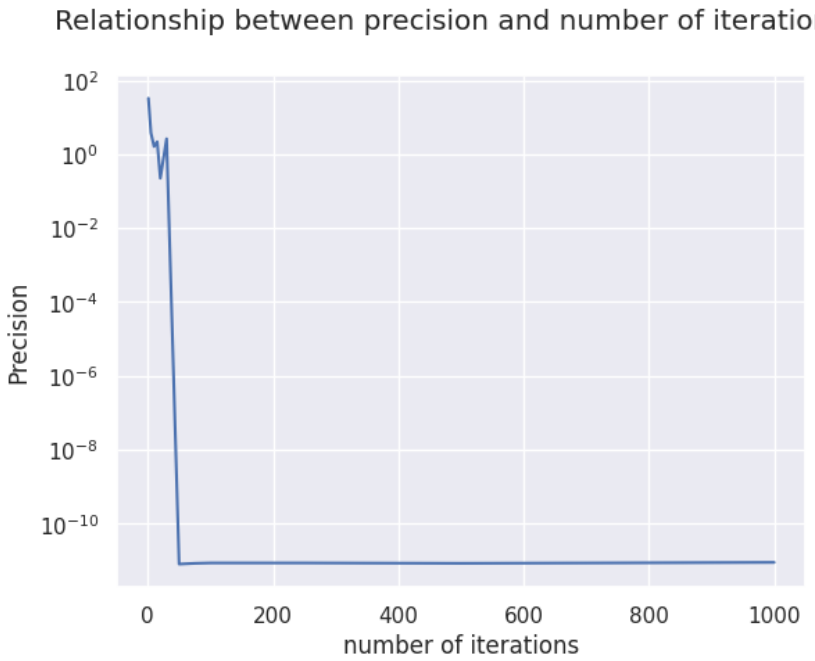


Рис. 4: Связь между максимальным количеством итераций QR-алгоритма и точностью
.

Видно, что примерно после 50 операций точность не меняется. Для дальнейших тестов значение этой костанты установленно именно 50.

Было произведено тестирование алгоритма SVD-разложения на матрицах разного размера. Алгоритм запускался на квадратных матрицах, каждый элемент которых был случайно сгенерирован из равномерного распределения на $[0, 1)$. Ниже представлен график зависимости времени работы алгоритма (в миллисекундах) от размера матриц, а также аналогичный график для встроенной библиотеки `Eigen` [1]:

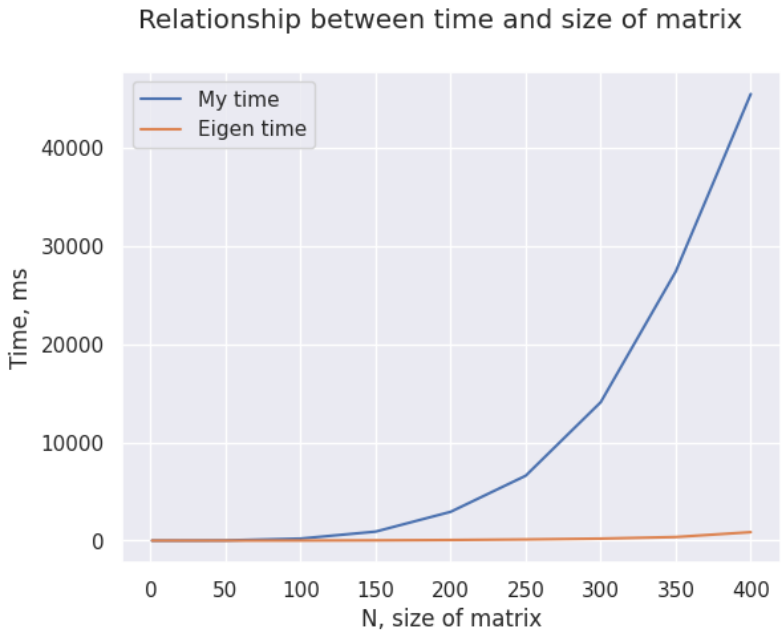


Рис. 5: Время работы моего алгоритма и встроенной библиотеки Eigen

Несмотря на довольно большую константу, мы видим что зависимость времени работы алгоритма от размера матрицы похожа на полиномиальную. Большую же разницу от времени работы библиотеки можно объяснить тем, что в библиотеке добавлены оптимизации. Целью же моей работы не было добиться лучшего времени работы.

Помимо времени работы, была замерена точность находимых алгоритмом сингулярных значений матрицы. В качестве эталонных значений использовались сингулярные значения, найденные с помощью библиотеки `Eigen` [1]. В качестве метрики рассматривалось значение $\|\Sigma - \Sigma_0\|$, где Σ — найденный нами вектор сингулярных значений, а Σ_0 — вектор сингулярных значений, найденный библиотекой `Eigen`. Тестирование на тех же данных, что и производительность, дало следующий график:

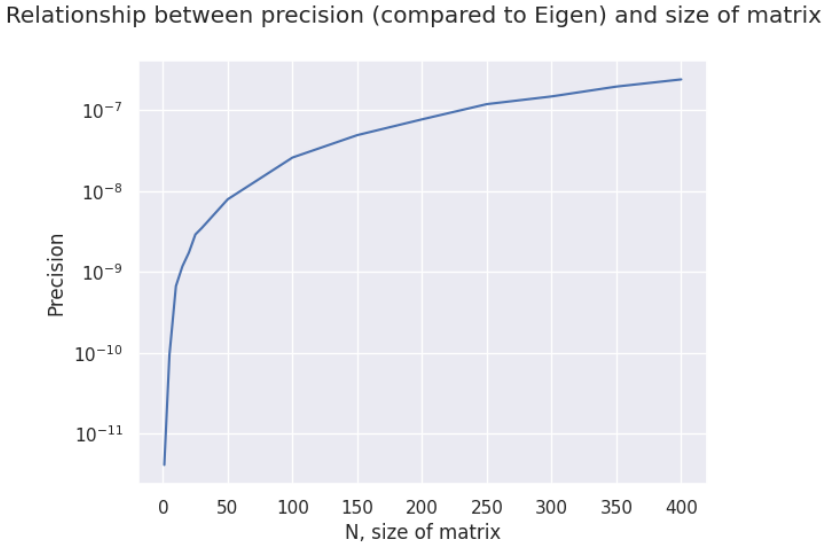


Рис. 6: Точность работы по сравнению с библиотекой Eigen

Как мы видим, даже при больших n наш алгоритм сохраняет достаточно хорошую точность. Рост ошибки при больших n связан, во-первых, с большим количеством операций в вещественных, а во-вторых с тем что наша метрика увеличивается при растущей размерности Σ.

Кроме того точность была измерена и по сравнению исходной матрицы. Сравнивалась изначальная матрица и матрица полученная после перемножения всех трех матриц в разложении. На тех же данных, что и раньше, график получился:

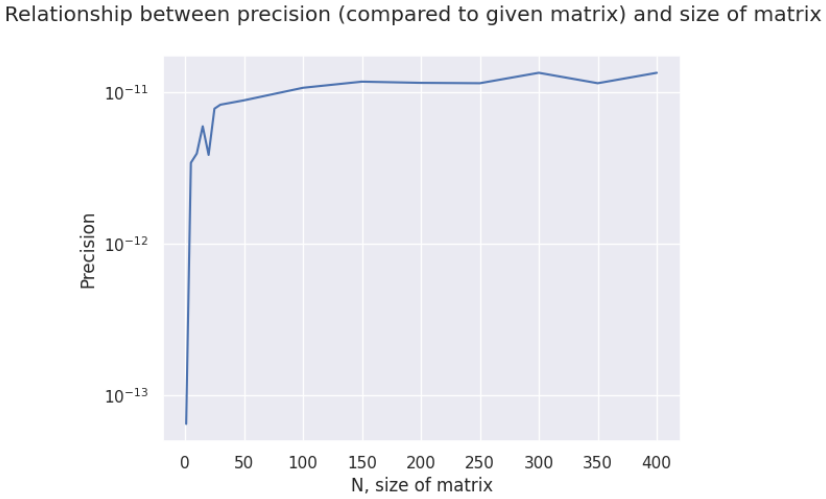


Рис. 7: Точность работы по сравнению с изначальной матрицей

Видно, что даже на больших матрицах, точность не превышает 10^{-10}.

Список литературы

[1] Eigen library. URL: https://eigen.tuxfamily.org/index.php?title=Main_Page.

[2] Github-репозиторий с кодом. URL: https://github.com/polipolinom/SVD_course_project_2_year.

[3] Google c++ style guide. URL: https://google.github.io/styleguide/cppguide.html.

[4] Dr. Peter Arbenz. Course "Numerical Methods for Solving Large Scale Eigenvalue Problems (Spring semester 2018)". 2018. URL: https://people.inf.ethz.ch/arbenz/ewp/Lnotes/?C=D;O=A.

[5] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM Journal on Numerical Analysis*, 2(2), 1965.

[6] G. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numer. Math*, 1970.

[7] L. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics, 1 edition, 1997.