МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

Направление подготовки: «Программная инженерия»

**ОТЧЁТ**

по производственной практике

**QR РАЗЛОЖЕНИЕ**

**Выполнила:** студентка группы

3824М1Пр1

\_\_\_А.Д. Коробейникова \_\_\_\_\_\_\_

Подпись

**Проверил:** д.т.н., доц.

\_\_\_\_К.А. Баркалов\_\_\_\_\_\_\_\_

Подпись

Нижний Новгород  
2025 г.

## Содержание

[1. Введение. 3](#_Toc199795784)

[2. Постановка задачи. 6](#_Toc199795785)

[3. Описание алгоритмов. 7](#_Toc199795786)

[4. Описание программы. 8](#_Toc199795787)

[5. Проверка корректности и ускорения. 9](#_Toc199795788)

[6. Заключение. 10](#_Toc199795789)

[7. Литература. 11](#_Toc199795790)

[8. Приложение. 12](#_Toc199795791)

1. Введение.

QR-разложение матрицы – представление матрицы в виде произведения унитарной (или ортогональной матрицы) Q (QTQ=E) и верхнетреугольной матрицы R с положительными диагональными элементами. QR-разложение является основой одного из методов поиска собственных векторов и чисел матрицы — QR-алгоритма. С помощью разложения можно решить линейную задачу наименьших квадратов.

Также с помощью QR-разложения можно решать системы линейных алгебраических уравнений (СЛАУ). В матричном виде эта задача описывается как (A –матрица системы, x – столбец неизвестных, b – столбец свободных членов). Если известно QR-разложение матрицы, исходная система может быть записана как . Решением этой задачи является решение треугольной системы . Эта система может быть решена с помощью обратной подстановки, поскольку R – верхняя треугольная матрица.

QR-разложение может быть получено различными методами. Проще всего оно может быть вычислено в процессе ортогонализации Грама-Шмидта. Другие методы QR-разложения основаны на отражениях Хаусхолдера и вращениях Гивенса.

## Постановка задачи.

Необходимо реализовать эффективный алгоритм QR-разложения произвольной матрицы A с помощью метода отражений Хаусхолдера. Разработать несколько версий, сравнить их работу друг с другом.

Для разработки использовать язык Си++ и технологии OpenMP для ускорения.

Сгенерировать целочисленную матрицу A размером nxn. Протестировать работу алгоритма на разных размерностях матрицы n: от небольших размеров (100 элементов), до больших объёмов данных (n>1000). Протестировать работу на различном количестве потоков. Сравнить работу алгоритма с известными реализациями в сторонних библиотеках.

Посчитать абсолютную и относительную ошибки для разложения. Критерием правильности считать близость ошибок к какому-либо малому eps, например, – такую точность знаков после запятой гарантирует тип float в Си++.

## Описание алгоритмов.

**Метод отражений Хаусхолдера**

Одним из самых распространённых методов нахождения QR-разложения является метод отражений Хаусхолдера. Пусть – n-мерный ненулевой вектор-столбец. Квадратная матрица H порядка n вида:

называется отражением Хаусхолдера или просто отражением. Вектор называется вектором Хаусхолдера. Умножению матрицы H на вектор x можно дать следующую геометрическую интерпретацию: вектор получается отражением вектора x относительно гиперплоскости, ортогональной вектору . Матрица Хаусхолдера будет симметрична и ортогональна.

Пусть . Найдем отражение H, такое, что для какого-либо α. Имеем:

и поэтому коллинеарен . Так как определен с точностью до ненулевого множителя, то можно положить Ввиду ортогональности матрицы H имеем, откуда . Поэтому

Итак, в качестве вектора Хаусхолдера можно взять вектор, вычисляемый по формуле , или любой ненулевой, ему коллинеарный.

С алгоритмом построения вектора Хаусхолдера связаны некоторые важные детали. Одна из них — выбор знака в формуле . Если x почти коллинеарен , то вектор имеет малую норму. Вследствие этого возможно появление большой относительной ошибки при вычислении множителя 2/. Эту трудность можно обойти, взяв с тем же знаком, что и первая компонента вектора x: .

Это обеспечивает выполнение неравенства и гарантирует почти точную ортогональность вычисленной матрицы H.

Другая деталь связана с выбором множителя для вектора, вычисляемого по формуле . Можно использовать такой множитель, что . Это несколько упрощает способ хранения ортогональных матриц, вычисленных по методу отражений Хаусхолдера.

Суть алгоритма QR-разложения, реализованного с помощью отражений Хаусхолдера – в последовательном домножении матрицы A слева на отражения Хаусхолдера H с целью получения матрицы R как побочного продукта и Q с помощью явных перемножений в простейшем случае.

На каждом шаге алгоритма k получаем k-ое отражение Hk, домножая на него матрицу A слева получаем , в которой k первых столбцов имеют нули под главной диагональю:

, ,

Выполнив n таких шагов, получим верхнюю треугольную матрицу . Откуда, так как матрицы Hj – ортогональные и симметричные, имеем , где .

Матрицы Hj не требуется формировать в явном виде: достаточно хранить соответствующий вектор vj, который можно записывать в j-й столбец матрицы A.

**Генерация данных**

Генерируется квадратная матрица A размером nxn. Значения для A генерируются в интервале [-10;10].

**Первая реализация**

С помощью ортогональных преобразований приведём матрицу A к треугольному виду. Возьмём первый столбец матрицы A: Построим матрицу Хасухолдера с помощью вектора и скаляров и . И, применив её к матрице, получим матрицу со столбцом нулей под первым диагональным элементом, где .

На втором этапе нужно поступить таким же образом с подматрицей матрицы, которая получается вычеркиванием в первой строки и первого столбца. Легко проверить, что это равносильно применению ко всей матрице преобразования Хаусхолдера, определяемого формулами , , , .

Таким образом, чтобы занулить n-1 строк матрицы A размером nxn, требуется выполнить n-1 шагов. На каждом из них нужно вычислять матрицу отражения H:

1. первые (k-1) позиций –нулевые.

Матрица , т.е. на каждом шаге алгоритма домножаем матрицу A слева на новую матрицу отражения и производим следующие вычисления уже над ней. При этом матрицу можно не хранить отдельно, по сути таким образом накапливается матрица R: . Для простейшей реализации матрицу Q храним и вычисляем отдельно на каждом шаге как произведение всех матриц отражения: . В результате получим .

**Проверка корректности**

Полученные матрицы Q и R необходимо перемножить и сравнить результат с исходной матрицей A.

Подсчитаем абсолютную ошибку с помощью Нормы Фробениуса (также евклидова норма, соответствующая евклидову пространству матриц относительно фробениусова скалярного произведения, или сферическая норма) представляет собой частный случай p-нормы для p = 2:, где n, m – размеры матрицы А. Получим и для оценки ошибки.

## Описание программы.

Для матрицы и вектора написаны шаблонные классы TMatrix и TVector соответственно. В простейшей реализации TVector внутри себя представлен стандартным вектором из пространства std, т.е. std::vector, для удобства работы с памятью. TMatrix представлен вектором TVector из векторов TVector (тип TVector<TVector<T>>). Внутри классов реализованы необходимые алгебраические операции сложения, вычитания и умножения матриц и векторов.

Функция main запускает вычисления для нескольких предпоготовленных тестовых данных (заданные матрицы A, Q и R размером 3x3) и девять вычислений для случайно генерируемых данных размером n. Для всех тестов выводится на экран:

1. Рамер задачи: test size: N
2. Время выполнения: time: <число в секундах>
3. Абсолютная ошибка: abs error: <вещественное число, >
4. Относительная ошибка: rel error: <вещественное число, >

Входные (матрица A) и выходные (матрицы Q и R) данные записываются в текстовые файлы для дальнейшего сравнения вычислений с релизацией QR разложения из библиотеки Scilab.

Скрипт для Scilab считывает матрицу A, подсчитывает для неё разложение и время разложения, далее считывает вычисленные мною матрицы Q и R, подсчитывает для каждой абсолютную и относительные ошибки для полученного разложения, а также для вычисленных ею матриц Q и R и моих Q и R и выводит данные на экран в формате

1. Рамер задачи: test size: N
2. Время выполнения: time: <число в секундах>
3. Абсолютная ошибка для матрицы A: abs error: <вещественное число, >
4. Относительная ошибка для матрицы A: rel error: <вещественное число, >
5. Абсолютная ошибка для вычисленной мною матрицы Q: abs error: <вещественное число, >
6. Относительная ошибка для вычисленной мною матрицы Q: rel error: <вещественное число, >
7. Абсолютная ошибка для вычисленной мною матрицы R: abs error: <вещественное число, >
8. Относительная ошибка для вычисленной мною матрицы R: rel error: <вещественное число, >

**Первая реализация**

Функция QR\_decomposition принимает на вход ссылки на матрицы A, Q и R. Копируем матрицу A в R. В цикле от 0 до n-1 подсчитываем по формулам . Подсчитываем Q как перемножение матриц отражения друг на друга, домножаем R на H слева и получаем матрицу для следующей итерации:

void QR\_decomposition(const TMatrix<T>& A, TMatrix<T>& Q, TMatrix<T>& R) {

const size\_t N = A.Size();

R = A;

for (size\_t k = 0; k < N - 1; ++k) {

auto beta = count\_beta(R, k);

auto mu = count\_mu(beta, k, R);

auto w = count\_w(R, k, beta, mu);

auto H = count\_H(beta, mu, w);

count\_Q(H, Q);

R = H \* R;

}

return;

}

float count\_beta(const TMatrix<T>& Ak, int k\_step) {

const size\_t N = Ak.Size();

double sum\_by\_k\_col = 0.0;

for (size\_t str = k\_step; str < N; ++str) {

sum\_by\_k\_col += Ak[str][k\_step] \* Ak[str][k\_step];

}

auto beta = sign(-Ak[k\_step][k\_step]) \* sqrt(sum\_by\_k\_col);

return beta;

}

float count\_mu(float beta, int k\_step, const TMatrix<T>& Ak) {

auto mu = 1 / sqrt(2.0 \* beta \* beta - 2 \* beta \* Ak[k\_step][k\_step]);

return mu;

}

TVector<T>count\_w(const TMatrix<T>& Ak,int k\_step,float beta,float mu) {

const size\_t N = Ak.Size();

TVector<T> w(N);

for (size\_t str = 0; str < N; ++str) {

w[str] = str < k\_step ? 0 : Ak[str][k\_step] \* mu;

}

w[k\_step] -= beta \* mu;

return w;

}

TMatrix<T> count\_H(float beta, float mu, TVector<T> w) {

size\_t N = w.Size();

TMatrix<T> E(N, N);

for (size\_t i = 0; i < N; ++i)

{

E[i][i] = 1.0;

}

auto H = E - w \* w \* 2.0;

return H;

}

void count\_Q(const TMatrix<T>& H, TMatrix<T>& Q) {

bool isFirst = Q.IsZero();

if (isFirst)

{

Q = H;

}

else

{

Q = Q \* H;

}

return;

}

Эта реализация понятна и проста, но долго работает. Замеры показали следующие результаты:

|  |  |  |  |
| --- | --- | --- | --- |
| Размер данных | Время работы, с | Абсолютная ошибка | Относительная ошибка |
| 100 | 1.41896 | 0.000746433 | 1.28803e-06 |
| 200 | 11.5924 | 0.00290193 | 2.5111e-06 |
| 300 | 56.9126 | 0.00631109 | 3.63063e-06 |
| 400 | 193.918 | 0.0112035 | 4.84842e-06 |
| 500 | 486.252 | 0.0174695 | 6.04093e-06 |
| 1000 | > 60 минут | > 0.1 | > 0.1 |

**Вторая реализация**

Небольшого ускорения удалось достичь благодаря замене обычного матричного умножения на блочное с размером блока 50. Также использовалось распараллеливание, сделанное прагмами OpenMP, там, где это было возможно.

Таблица с замерами для запуска с восемью потоками:

|  |  |  |  |
| --- | --- | --- | --- |
| Размер данных | Время работы, с | Абсолютная ошибка | Относительная ошибка |
| 100 | 0.103752 | 0.00059191 | 1.02058e-06 |
| 200 | 1.07141 | 0.00173145 | 1.49418e-06 |
| 300 | 7.86116 | 0.00327259 | 1.88301e-06 |
| 400 | 29.6932 | 0.00504026 | 2.17705e-06 |
| 500 | 81.4453 | 0.00717861 | 2.47992e-06 |
| 1000 | 25 минут | 0.0215255 | 3.72178e-06 |

Сравнение результатов с работой QR-разложения из Scilab (на тех же данных):

|  |  |  |  |
| --- | --- | --- | --- |
| Размер данных | Время работы, с | Абсолютная ошибка | Относительная ошибка |
| 100 | 0.0066856 | 8.885e-10 | 1.5153e-10 |
| 200 | 0.0014002 | 4.441e-10 | 5.516e-10 |
| 300 | 0.0024455 | 6.951e-10 | 6.675 e-10 |
| 400 | 0.0042798 | 1.414e-8 | 1.237e-10 |
| 500 | 0.0074805 | 1.332e-8 | 1.030e-10 |
| 1000 | 0.069125 | 3.734e-10 | 1.620e-10 |

## Проверка корректности и ускорения.

Результаты:

## Заключение.

Код работы я прикрепляю в Приложении.

## Литература.

1. Параллельные численные методы: сайт. — URL: http://www.hpcc.unn.ru/?doc=491 (дата обращения: 04.05.2025). — Текст: электронный.
2. Белов С.А., Золотых Н.Ю. Численные методы линейное алгебры – 2010 г..
3. Численные методы. Линейная алгебра и нелинейные уравнения, Вержбицкий В.М. Москва, 2005

## Приложение.