МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных систем и технологий
Кафедра «Прикладная математика и информатика»
Дисциплина «Численные методы»

КУРСОВАЯ РАБОТА

Тема «Нахождение собственных значений матрицы QR-алгоритмом»

Выполнил студент _____ / В. Д. Шувалова /
подпись инициалы, фамилия

Курс 3 Группа ПМбд-31

Направление/ специальность 01.03.04 Прикладная математика

Руководитель зав. кафедрой ПМИ, к.т.н., доцент
должность, ученая степень, ученое звание
Кувайскова Юлия Евгеньевна
фамилия, имя, отчество

Дата сдачи:
«____»_____20____г.

Дата защиты:
«____»_____20____г.

Оценка: _____

Ульяновск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных систем и технологий
Кафедра «Прикладная математика и информатика»
Дисциплина «Численные методы»

ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)

студенту ПМбд-31 Шувалова В. Д.
группа фамилия, инициалы

Тема проекта (работы) «Нахождение собственных значений матрицы QR-алгоритмом»

Срок сдачи законченной работы «___» _____20____г.

Исходные данные к работе 1. Найти собственные значения матрицы A двумя численными методами: QR-алгоритмом и методом вращений Якоби с точностью ε. QR-разложение реализовать двумя подходами: процессом Грама-Шмидта и поворотом Гивенса. 2. Провести сравнение указанных методов решения. При задании точностей $\varepsilon_1 = 10^{-3}$ и $\varepsilon_2 = 10^{-6}$ результаты сравнения и расчетов записать в сводную таблицу. 3. Вычислить собственные векторы матрицы по найденным собственным значениям методом Гаусса, используя уравнение $Ax = \lambda x$. Указание: В программе предусмотреть ввод требуемой точности ε, а также сообщения, предупреждающие о невозможности решения указанной задачи.
(базовое предприятие, характер курсового проекта (работы): задание кафедры, инициативная НИР, рекомендуемая литература, материалы практики)

Содержание пояснительной записки (перечень подлежащих разработке вопросов)
Введение 1. Теоретическая часть 1.1. Постановка задачи 1.2. QR-разложение по процессу Грама-Шмидта 1.3. QR-разложение с помощью поворота Гивенса 1.4. QR-алгоритм 1.5. Метод вращений Якоби 1.6. Метод Гаусса для нахождения собственных векторов 2. Практическая часть 2.1. Описание работы программы для пользователя 2.2. Краткий обзор структуры и функций кода 3. Исследовательская часть 3.1. Решение с точностью $\varepsilon = 10^{-3}$ 3.2. Решение с точностью $\varepsilon = 10^{-6}$ 3.3. Сравнение численных методов Заключение Список литературы Приложение

Перечень графического материала (с точным указанием обязательных чертежей)
_____

Руководитель зав. кафедрой ПМИ _____ / Ю. Е. Кувайскова /
должность подпись инициалы, фамилия

«_____»_____20___г.

Студент _____ /_____ В. Д. Шувалова_____/
подпись инициалы, фамилия

«_____»_____20___г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ОТЗЫВ
руководителя на курсовую работу

студента Шуваловой Валерии Дмитриевны
фамилия, имя и отчество

Факультет информационных систем и технологий группа ПМбд-31 курс 3
Дисциплина Численные методы
Тема работы «Нахождение собственных значений матрицы QR-алгоритмом»

Курсовая работа Шуваловой В.Д. рассматривает решение проблемы поиска собственных значений матриц с помощью таких численных методов как QR-алгоритм и метод вращений Якоби.

Целью курсовой работы является программная реализация нахождения собственных значений матрицы при помощи QR-алгоритма и метода вращений Якоби, а также нахождение соответствующих им собственных векторов с помощью метода Гаусса.

Для достижения цели были поставлены следующие задачи: изучить научную литературу, учебные пособия и электронные ресурсы, в которых рассматриваются QR-алгоритм, метод вращений Якоби, метод Гаусса, а также QR-разложение с помощью следующих подходов: процесса Грама-Шмидта и поворота Гивенса; программно реализовать решение исходной задачи указанными численными методами и подходами; провести сравнение численных методов и сделать выводы.

В результате была разработана программа численного решения проблемы поиска собственных значений с использованием языка программирования Python версии 3.12 и среды разработки PyCharm Community Edition версии 2024. При помощи программы были получены собственные значения и собственные векторы для указанной матрицы при задании различных точностей от пользователя. В итоге для решаемой задачи все методы были достаточно точны, но самым эффективным по времени оказался метод вращений Якоби.

Тему курсовой работы Шувалова В.Д. выбрала самостоятельно, что показывает заинтересованность в решении данной проблемы. При выполнении курсовой работы сообщала, на каком этапе реализации находится, консультировалась по интересующим вопросам. Студентка привела подробный теоретический материал и продемонстрировала его знание, а также показала высокий уровень программирования на Python с помощью использования уникальных синтаксических возможностей языка и понимания оценки сложности алгоритмов.

Структура и содержание курсовой работы Шуваловой В.Д. полностью соответствуют заданию. Работа выполнена грамотно и в сроки.

Руководитель зав. кафедрой ПМИ, к.т.н., доцент _____ / Ю.Е. Кувайскова
должность, учёная степень, ученое звание подпись инициалы, фамилия

«_____»_____20____г.

Оглавление

Введение

В современной линейной алгебре нахождение собственных значений матриц является одной из ключевых задач, имеющей широкое применение в различных областях науки и техники. Собственные значения играют важную роль в различных методах обработки изображений, особенно в задачах, связанных с анализом и сжатием данных. Например, в методе главных компонент, предназначенном для сжатия изображений, собственные значения позволяют отобрать несколько наиболее информативных компонент для представления изображения, что помогает сжать изображение, сохранив его ключевые характеристики. Также собственные значения и собственные векторы находят широкое применение в обработке физических сигналов. Например, в задачах фильтрации сигналов собственные векторы могут быть использованы для представления сигналов в пространстве, где шум и полезный сигнал могут быть различимы, что позволяет применять фильтры шумоподавления. Кроме того, они также используются в квантовой механике для описания таких физических величин как энергия и импульс.

С начала XX века внимание математиков к этой задаче возросло, и с тех пор было разработано множество численных методов для вычисления собственных значений. Среди них особенно выделяется QR-алгоритм, разработанный в конце 1950-х годов независимо В. Н. Кублановской и Дж. Фрэнсисом. Численные методы позволяют значительно сократить время вычислений по сравнению с аналитическими подходами. Это особенно важно в ситуациях, требующих быстрых вычислений, например, в обработке физических сигналов, когда время между поступлением входного сигнала и получением преобразованного сигнала на выходе должно быть минимально возможным.

Целью курсовой работы является программная реализация нахождения собственных значений матрицы при помощи QR-алгоритма и метода вращений Якоби, а также нахождение соответствующих им собственных векторов с помощью метода Гаусса.

Для достижения этой цели были поставлены следующие задачи:

1. Изучить научную литературу, учебные пособия и электронные ресурсы, в которых рассматриваются QR-алгоритм, метод вращений Якоби, метод Гаусса, а также QR-разложение с помощью следующих подходов: процесса Грама-Шмидта и поворота Гивенса.

2. Программно реализовать решение исходной задачи указанными численными методами и подходами.

3. Провести сравнение численных методов и сделать выводы.

Курсовая работа состоит из трёх глав. В теоретической части описываются основные идеи процесса Грама-Шмидта и поворота Гивенса как подходов реализации QR-разложения, а также основные идеи QR-алгоритма, метода вращений Якоби и метода Гаусса. В практической части представлена реализация программы. В исследовательской части приводится сравнение численных методов.

Код программы написан на языке программирования Python версии 3.12 с использованием среды разработки PyCharm Community Edition версии 2024 и прикреплён в приложении.

1. Теоретическая часть

1.1. Постановка задачи

В данной части будут представлены основные идеи двух численных методов для нахождения собственных значений квадратных вещественных матриц, а именно: QR-алгоритм и метод вращений Якоби.

Основой QR-алгоритма является QR-разложение. Это представление матрицы A в виде произведения ортогональной матрицы Q и верхнетреугольной матрицы R:

$$A = QR.$$

Ортогональная матрица Q – это квадратная матрица с вещественными элементами, результат умножения которой на транспонированную матрицу равен единичной матрице E:

$$QQ^T = Q^TQ = E.$$

Верхнетреугольная матрица R – это квадратная матрица, у которой все элементы ниже главной диагонали равны нулю.

QR-разложение может быть реализовано различными подходами. Сначала будет рассмотрен такой подход как процесс Грама-Шмидта, далее – поворот Гивенса. Важно отметить, что QR-разложение может быть неприменимо для матриц, вектор-столбцы которых линейно-зависимы или почти линейно-зависимы в пределах заданной точности.

Метод вращений Якоби не содержит такой самостоятельной основы как QR-разложение в QR-алгоритме, а потому будет полностью рассмотрен в части 1.5.

В конце теоретической части будет рассмотрен метод Гаусса для нахождения собственных векторов, соответствующих найденным ранее собственным значениям. Для этого используется характеристическое уравнение матрицы:

$$Ax = \lambda x,$$

где λ – собственное значение, x – соответствующий ему собственный вектор. Преобразовав данное уравнение, получим СЛАУ:

$$(A - \lambda E)x = 0.$$

Именно это СЛАУ будет решаться методом Гаусса. Поскольку каждое собственное значение имеет бесконечно много собственных векторов, метод Гаусса будет находить один “красивый” собственный вектор, присваивая единицу произвольным неизвестным, так как это используется для удобства представления собственных векторов.

1.2. QR-разложение по процессу Грама-Шмидта

Сначала для квадратной матрицы n-го порядка

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$$

составляется набор вектор-столбцов:

$$a_1 = \begin{pmatrix} a_{11} \\ \vdots \\ a_{n1} \end{pmatrix}, \dots, a_n = \begin{pmatrix} a_{1n} \\ \vdots \\ a_{nn} \end{pmatrix}.$$

Затем для них строится система ортогональных векторов $b_1, b_2, \dots, b_n$. Это система векторов, где все векторы попарно ортогональны, то есть перпендикулярны. Она строится по следующим формулам:

$$b_1 = a_1,$$

$$b_2 = a_2 - proj_{b_1} a_2,$$

$$b_3 = a_3 - proj_{b_1} a_3 - proj_{b_2} a_3,$$

$$\dots,$$

$$b_n = a_n - proj_{b_1} a_n - proj_{b_2} a_n - \cdots - proj_{b_{n-1}} a_n.$$

Здесь $proj$ – это обозначение проекции, определяемой следующим образом:

$$proj_b a = \frac{\langle a, b \rangle}{\langle b, b \rangle} b$$

– проекция вектора a на вектор b, использующая $\langle a, b \rangle$ – скалярное произведение векторов a и b. Оно может быть найдено по формуле:

$$\langle a, b \rangle = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n,$$

где $a_1, a_2, \dots, a_n$ и $b_1, b_2, \dots, b_n$ – координаты векторов a и b соответственно.

Далее для $b_1, b_2, \dots, b_n$ строится система ортонормированных векторов $e_1, e_2, \dots, e_n$. Это ортогональная система векторов, где каждый вектор нормирован. Нормированный вектор – это вектор единичной длины, то есть вектор, норма которого равна единице. Он получается делением вектора на его норму:

$$e_j = \frac{b_j}{\|b_j\|}, \qquad j = \overline{1, n}.$$

Норма вектора x определяется следующим образом:

$$\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2},$$

где x_i – координаты вектора x.

Столбцы ортогональной матрицы Q формируются из найденных векторов $e_j,\ j = \overline{1, n}$.

Далее необходимо найти верхнетреугольную матрицу R. Выражение для её нахождения выводится из QR-разложения:

$$A = QR.$$

Умножим обе части этого уравнения на Q^{-1} слева:

$$Q^{-1}A = Q^{-1}QR.$$

Учитывая, что $Q^{-1}Q = E$ по определению обратной матрицы:

$$Q^{-1}A = ER.$$

Единичная матрица E играет роль единицы при умножении, поэтому:

$$R = Q^{-1}A,$$

где Q^{-1} – матрица, обратная матрице Q. Чтобы не находить обратную матрицу, можно воспользоваться свойством ортогональной матрицы Q:

$$Q^{-1} = Q^T,$$

где Q^T – транспонированная матрица Q. Тогда верхнетреугольная матрица R находится по следующей формуле: $R = Q^T A$. Так находится QR-разложение матрицы A по процессу Грама-Шмидта.

1.3. QR-разложение с помощью поворота Гивенса

Суть данного подхода заключается в том, что верхнетреугольная матрица R изначально полагается равной исходной матрице A. Затем она постепенно меняется путём множества умножений на матрицу Гивенса G слева. Такое умножение называется поворотом Гивенса. Поворот Гивенса позволяет обнулить или сделать близким к нулю один из элементов матрицы. Для получения верхнетреугольной матрицы требуется обнулить все элементы, стоящие под главной диагональю.

Для обнуления элемента a_{ij}, стоящего в i-ой строке j-ом столбце под главной диагональю, находится опорный элемент – это элемент, стоящий над a_{ij} на главной диагонали, то есть элемент a_{jj}. Далее рассчитываются косинус и синус по формулам:

$$\cos\varphi = \frac{a_{jj}}{\sqrt{a_{jj}^2 + a_{ij}^2}}, \qquad \sin\varphi = \frac{-a_{ij}}{\sqrt{a_{jj}^2 + a_{ij}^2}}.$$

Матрица G_{ij}, позволяющая обнулить элемент a_{ij}, является единичной матрицей E n-го порядка, за исключением некоторых её элементов: в i-ой строке i-ом столбце и j-ой строке j-ом столбце G_{ij} стоят элементы $\cos\varphi$, в i-ой строке j-ом столбце – элемент $\sin\varphi$, в j-ой строке i-ом столбце – элемент $-\sin\varphi$. Тогда матрица R становится равной $G_{ij}R$. Такой поворот Гивенса обнуляет элемент a_{ij} в изменённой R.

Далее необходимо найти ортогональную матрицу Q. Выражение для её нахождения выводится из QR-разложения:

$$A = QR.$$

Умножим обе части этого уравнения на R^{-1} справа:

$$AR^{-1} = QRR^{-1}.$$

Учитывая, что $RR^{-1} = E$ по определению обратной матрицы:

$$AR^{-1} = QE.$$

Единичная матрица E играет роль единицы при умножении, поэтому:

$$Q = AR^{-1},$$

где R^{-1} – матрица, обратная матрице R. Тогда ортогональная матрица Q находится по следующей формуле: $Q = AR^{-1}$. Так находится QR-разложение матрицы A с помощью поворота Гивенса.

1.4. QR-алгоритм

Важно отметить, что QR-алгоритм строго доказан для положительно-определённых симметричных матриц, что обеспечивает сходимость алгоритма к собственным значениям. Для других типов матриц алгоритм может также работать, но его сходимость не всегда гарантирована.

Суть QR-алгоритма заключается в том, что исходная матрица A на каждой итерации выполнения алгоритма постепенно приводится к матрице верхнетреугольного вида.

Изначально, на 0-ой итерации, полагается:

$$A_0 = A.$$

На i-ой итерации алгоритма вычисляется QR-разложение с помощью любого выбранного подхода:

$$A_i = Q_i R_i.$$

Затем матрица на (i+1)-ом шаге определяется следующим образом:

$$A_{i+1} = R_i Q_i.$$

Критерием остановки итерационного алгоритма является близость к нулю элементов под главной диагональю, то есть их модуль должен быть меньше заданной точности ε. Альтернативным критерием остановки алгоритма может быть близость к нулю наибольшего по модулю элемента под главной диагональю. По завершении алгоритма собственные значения исходной матрицы находятся на главной диагонали конечной матрицы.

1.5. Метод вращений Якоби

Метод вращений Якоби применяется только для симметричных матриц. Суть метода заключается в том, что исходная квадратная матрица A n-го порядка на каждой итерации выполнения алгоритма постепенно приводится к

матрице диагонального вида, то есть к матрице, у которой все элементы вне главной диагонали близки к нулю.

На каждой i-ой итерации алгоритма происходят приближения к нулю всех элементов матрицы вне главной диагонали. Так как матрица симметрична, то приближения к нулю происходят для двух симметричных элементов сразу. Для этого выбирается опорной элемент a_{jj}, стоящий в j-ой строке j-ом столбце для $j = \overline{1, n-1}$. Относительно него выбираются по два симметричных элемента для разных индексов k: a_{kj} в k-ой строке и j-ом столбце и a_{jk} в j-ой строке и k-ом столбце, где $k = \overline{j+1, n}$. Тогда для их приближения к нулю рассчитываются косинус c и синус s по формулам:

a) Если $a_{jj} = a_{kk}$:

$$\theta = \frac{\pi}{4},$$

$$c = cos\theta, \qquad s = \sin\theta.$$

b) Иначе, если $a_{jj} \neq a_{kk}$:

$$\tau = \frac{a_{jj} - a_{kk}}{2a_{jk}},$$

$$t = \frac{sign(\tau)}{|\tau| + \sqrt{1+\tau^2}},$$

$$c = \frac{1}{\sqrt{1+t^2}}, \qquad s = tc.$$

Здесь $sign(\tau)$ – это математическая функция, которая определяется следующим образом:

$$sign(\tau) = \begin{cases} 1, \tau > 0 \\ 0, \tau = 0 \\ -1, \tau < 0 \end{cases}.$$

Приближение к нулю симметричных элементов происходит в ходе двустороннего вращения, когда матрица A становится равной $J_j^T A J_j$, где J_j – матрица вращения, которая является единичной матрицей E n-го порядка, за исключением некоторых её элементов: в j-ой строке j-ом столбце и k-ой строке

k-ом столбце J_j стоят элементы c, в k-ой строке j-ом столбце – элемент s, в j-ой строке k-ом столбце – элемент $-s$.

Одной итерации алгоритма может быть недостаточно для того, чтобы элементы вне главной диагонали достаточно приблизились к нулю, поэтому итерации завершаются, когда модуль наибольшего элемента вне главной диагонали становится меньше заданной точности ε. По завершении алгоритма собственные значения исходной матрицы находятся на главной диагонали конечной матрицы.

Метод вращений Якоби похож на QR-разложение с помощью поворота Гивенса, однако имеет ряд отличий:

a) Метод вращений Якоби является полноценным численным методом для нахождения собственных значений матрицы, в то время как QR-разложение с помощью поворота Гивенса является лишь составной частью другого численного метода – QR-алгоритма.
b) Поворот Гивенса применяется в QR-разложении строго определённое количество раз – столько, сколько элементов под главной диагональю надо обнулить. В то время как метод вращений Якоби может применяться неопределённое количество итераций до достижения заданной точности ε.
c) Формулы для расчётов различны.

1.6. Метод Гаусса для нахождения собственных векторов

В данной части будет описываться суть метода Гаусса с учётом его применения для нахождения собственных векторов, соответствующих найденным ранее собственным значениям. Так как собственные значения существуют только у квадратных матриц, то метод Гаусса будет работать со СЛАУ, где количество неизвестных равно количеству уравнений в системе, тогда СЛАУ будет иметь вид:

$$\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n = b_1 \\ \cdots \\ a_{n1}x_1 + \cdots + a_{nn}x_n = b_n \end{cases}.$$

Её можно записать в матричном виде:

$$Ax = b,$$

где

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}, \qquad x = \begin{pmatrix} x_1 \\ \vdots \\ x_2 \end{pmatrix}, \qquad b = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}.$$

Матрица A называется основной матрицей, содержащей коэффициенты СЛАУ. x – вектор неизвестных переменных, b – вектор свободных членов. Метод Гаусса работает с расширенной матрицей A':

$$A' = \left(\begin{array}{cccc|c} a_{11} & \cdots & a_{1n} & & b_1 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & \cdots & a_{nn} & & b_n \end{array}\right).$$

Он состоит из двух этапов: прямого и обратного хода.

Прямой ход заключается в постепенном приведении матрицы A' к верхнетреугольному виду путём последовательного исключения сначала x_1 из второго, третьего, …, n-го уравнений, затем - x_2 из третьего, четвёртого, …, n-го уравнений и т. д. Ниже этот процесс описывается более подробно.

Чтобы исключить x_1 из всех уравнений, кроме первого, нужно убедиться, что x_1 в первом уравнении есть, то есть в первой строке матрицы A первый коэффициент a_{11} должен быть ненулевым. Иначе необходимо найти такую строку матрицы, где первый коэффициент, соответствующий неизвестной x_1, не равен нулю. Если такой строки нет, то происходит переход к рассмотрению неизвестной x_2. Если же такая строка нашлась, то она становится первой строкой матрицы, меняясь местами с найденной. Тогда во всех строках под ней, то есть во второй, третьей, …, n-ой строках, обнуляются первые коэффициенты, соответствующие неизвестной x_1. Для этого вторую, третью, …, n-ую строку нужно сложить с первой строкой, умноженной на $-\frac{a_{21}}{a_{11}}, -\frac{a_{31}}{a_{11}}, \ldots, -\frac{a_{n1}}{a_{11}}$ соответственно. Эти действия, как и обмен местами строк в матрице, являются эквивалентными преобразованиями матрицы, которые не меняют множество решений системы, которой соответствует эта матрица.

Затем, чтобы исключить x_2 из третьего, четвёртого, …, n-го уравнений, нужно убедиться, что x_2 во втором уравнении есть, то есть во второй строке матрицы A второй коэффициент a_{22} должен быть ненулевым. Иначе среди третьей, четвертой, n-ой строк необходимо найти такую строку матрицы, где второй коэффициент, соответствующий неизвестной x_2, не равен нулю. Если такой строки нет, то происходит переход к рассмотрению неизвестной x_3. Если же такая строка нашлась, то она становится второй строкой матрицы, меняясь местами с найденной. Тогда во всех строках под ней, то есть в третьей, четвёртой, …, n-ой строках обнуляются вторые коэффициенты, соответствующие неизвестной x_2. Для этого третью, четвёртую, …, n-ую строку нужно сложить с первой строкой, умноженной на $-\frac{a_{32}}{a_{22}}, -\frac{a_{42}}{a_{22}}, \dots, -\frac{a_{n2}}{a_{22}}$ соответственно.

Далее по виду получившейся матрицы A' можно определить, сколько решений имеет СЛАУ. Вообще говоря, собственному значению соответствует бесконечно много собственных векторов, то есть метод Гаусса в данной задаче всегда будет работать со СЛАУ, имеющими бесконечное множество решений, но тем не менее рассмотрим данный метод в общем виде.

Сначала определяется совместность или несовместность СЛАУ: система совместна, если имеет хотя бы одно решение, иначе она несовместна. Если в матрице A' встретится строка, где все коэффициенты a равны нулю, а свободный член b не равен нулю, то система несовместна, так как возникло ложное числовое равенство. Иначе, если в матрице не встретилась такая строка, система совместна.

Затем в случае совместности СЛАУ определяется её определённость или неопределённость: система определена, если имеет одно решение, иначе – неопределена. Стоит отметить, что неопределённая система всегда имеет бесконечное множество решений. Если в матрице A' встретится строка, где все коэффициенты a равны нулю, и свободный член b тоже равен нулю, то система неопределена, так как есть неизвестные, которые могут принимать произвольные числовые значения, потому что других уравнений квадратной

системы не хватит для их однозначного определения. Иначе, если в матрице не встретилась такая строка, система определена.

Далее запускается обратный ход метода Гаусса, который заключается в последовательном вычислении числовых значений неизвестных. В случае определённости СЛАУ обратный ход устроен просто: x_n определяется из n-го уравнения $x_n = \frac{b_n}{a_{nn}}$, x_{n-1} – из (n-1)-го уравнения, учитывая уже найденное x_n $x_{n-1} = \frac{b_{n-1} - a_{n-1n}x_n}{a_{n-1n-1}}$ и т. д.

В случае неопределённости СЛАУ обратный ход устроен сложнее: для нахождения n неизвестных понадобится n итераций. Тем неизвестным, которые могут принимать произвольные числовые значения, будет присвоена единица для красоты собственного вектора. Сначала, на 1-ой итерации, ищется строка матрицы A', из которой можно найти минимальное ненулевое количество неизвестных x, то есть в этой строке минимальное ненулевое количество коэффициентов $a \neq 0$. Если в ней содержится один коэффициент $a \neq 0$, то из неё можно однозначно определить одну неизвестную x, соответствующую этому коэффициенту. Если же коэффициентов $a \neq 0$ получилось более одного, то одной из неизвестных, соответствующих этим коэффициентам $a \neq 0$, присваивается единица. На следующей итерации действия повторяются с учетом одной найденной неизвестной и т. д.

2. Практическая часть

В данной части сначала будет представлено описание работы программы для пользователя, чтобы он смог её запустить и использовать. Затем – краткий обзор структуры и функций кода. Он полезен для тех, кто хочет разобраться в коде, использовать его в своих проектах, доработать.

2.1. Описание работы программы для пользователя

Для начала работы с программой пользователю необходимо запустить исполняемый файл qr_algorithm.exe:

Рисунок 1. Запуск программы.

После ему необходимо будет ввести точность вычислений и нажать Enter:

```
Введите точность eps от 10^-14 до 10^-1
в формате [0.0...0[ненулевое число]],
например: eps = 0.001

Для точности eps < 10^-14
решение задачи невозможно в силу
ограниченных вычислительных возможностей Python.

eps =
```

Рисунок 2. Ввод точности от пользователя.

Если пользователь введёт не число или использует запятую, то будет выведено соответствующее сообщение и приглашение к повторному вводу:

```
eps = gfhj

Вы ввели не число или использовали запятую.
Введите ещё раз:

eps = 0,001

Вы ввели не число или использовали запятую.
Введите ещё раз:

eps =

Вы ввели не число или использовали запятую.
Введите ещё раз:

eps =
```

Рисунок 3. Некорректный ввод точности от пользователя – введено не число или использована запятая

Если же пользователь введёт число вне диапазона $[10^{-14};\ 10^{-1}]$, то будет выведено соответствующее сообщение и приглашение к повторному вводу:

```
eps = 100

Вы ввели число вне диапазона [10^-14; 10^-1].
Введите ещё раз:

eps = 0.00000000000000000000000000000001

Вы ввели число вне диапазона [10^-14; 10^-1].
Введите ещё раз:

eps = 0.5

Вы ввели число вне диапазона [10^-14; 10^-1].
Введите ещё раз:

eps =
```

Рисунок 4. Некорректный ввод точности от пользователя – число вне диапазона

Нижняя граница диапазона равна 10^{-14}, так как вычислительные возможности Python ограничены, как и у любого другого языка программирования. Если превысить эту точность, то начнут накапливаться ошибки округления и погрешности вычислений, в результате программа будет работать некорректно.

Если пользователь введёт число вне формата 0.0 ... [ненулевое число], то будет выведено соответствующее сообщение и приглашение к повторному вводу:

```
eps = 0.00012

Вы ввели число не в формате [0.0...0[ненулевое число]].
Введите ещё раз:

eps = 0.00010

Вы ввели число не в формате [0.0...0[ненулевое число]].
Введите ещё раз:

eps = 0.0010001

Вы ввели число не в формате [0.0...0[ненулевое число]].
Введите ещё раз:

eps =
```

Рисунок 5. Некорректный ввод точности от пользователя – число вне формата

Приглашения ко вводу будут повторяться до тех пор, пока пользователь не введёт корректное число. Когда это произойдет, будут выведены собственные значения, посчитанные тремя подходами: QR-алгоритмом с использованием QR-разложения по процессу Грама-Шмидта, QR-алгоритмом с использованием QR-разложения с помощью поворота Гивенса, методом

вращений Якоби. Далее будут выведены собственные векторы, соответствующие найденным ранее собственным значениям.

```
eps = 0.0001

Собственные значения,
найденные QR-алгоритмом для QR-разложения
по процессу Грама-Шмидта:
L1 = 7.0000
L2 = 2.0000
L3 = -1.0000
L4 = 1.0000
L5 = 1.0000
```

Рисунок 6. Решение QR-алгоритмом с использованием QR-разложения по процессу Грама-Шмидта при $\varepsilon = 10^{-4}$.

```
Собственные значения,
найденные QR-алгоритмом для QR-разложения
по повороту Гивенса:
L1 = 7.0000
L2 = 2.0000
L3 = -1.0000
L4 = 1.0000
L5 = 1.0000
```

Рисунок 7. Решение QR-алгоритмом с использованием QR-разложения с помощью поворота Гивенса при $\varepsilon = 10^{-4}$.

```
Собственные значения,
найденные методом вращений Якоби:
L1 = 7.0000
L2 = 2.0000
L3 = -1.0000
L4 = 1.0000
L5 = 1.0000
```

Рисунок 8. Решение методом вращений Якоби при $\varepsilon = 10^{-4}$.

```
Собственные векторы:

для собственного значения L1 = 7.0000:
1.5000
1.0000
1.5000
1.0000
1.0000

для собственного значения L2 = 2.0000:
-1.0000
1.0000
-1.0000
1.0000
1.0000

для собственного значения L3 = -1.0000:
1.0000
0.0000
-1.0000
0.0000
0.0000

для собственного значения L4 = 1.0000:
0.0000
1.0000
0.0000
1.0000
-2.0000

для собственного значения L5 = 1.0000:
0.0000
1.0000
0.0000
1.0000
-2.0000
```

Рисунок 9. Собственные векторы при $\varepsilon = 10^{-4}$.

Далее будет выведена таблица, которая сравнивает алгоритмы по времени работы и количеству итераций:

```
Сравнительная таблица:
+--------------------+------------------+--------------+--------------+
| Критерий сравнения | QR - Грамм-Шмидт | QR - Гивенс  | М. вр. Якоби |
+--------------------+------------------+--------------+--------------+
|  Время в секундах  |   0.0019838810   | 0.0039615631 | 0.0009970665 |
|  Кол-во итераций   |        15        |      15      |      3       |
+--------------------+------------------+--------------+--------------+
```

Рисунок 10. Сравнительная таблица алгоритмов при $\varepsilon = 10^{-4}$.

Чтобы закрыть программу, пользователь может нажать Enter ещё раз или закрыть окно программы, нажав на крестик в углу окна.

2.2. Краткий обзор структуры и функций кода

Программа написана на языке программирования Python версии 3.12 с использованием среды разработки PyCharm Community Edition версии 2024. Для обеспечения совместимости кода можно использовать любую среду

разработки, однако версия Python не должна быть ниже 3.8, иначе некоторые синтаксические конструкции, например f-строки, будут работать некорректно.

Программа состоит из одного файла qr_algorithm.py. Сначала подключаются модули и библиотеки:

```
7   import math
8   import numpy as np
9   import prettytable
10  import time
```

Рисунок 11. Подключение модулей и библиотек.

Модуль – это файл с кодом, который содержит готовые инструменты, которые можно использовать в своих программах. Библиотека – это набор связанных модулей, предоставляющих более широкий функционал. В данной программе импортированы три модуля и одна библиотека.

Модуль math является встроенным в стандартную библиотеку Python. Он предоставляет доступ к различным математическим функциям и константам. Модуль prettytable является сторонним модулем, который предоставляет инструменты для форматирования и отображения табличных данных. Библиотека numpy является сторонней библиотекой, которая предоставляет поддержку многомерных массивов и высокоуровненые математические функции, работающие с этими массивами. В данной программе numpy использовалась для вычисления обратных матриц в QR-разложении с помощью поворота Гивенса. Модуль time является встроенным и предоставляет функции для работы с временем и датами.

Встроенные модули не требуют установки для их использования. Сторонние же модули и библиотеки нужно устанавливать. Далее будет описан процесс установки сторонних инструментов для среды PyCharm Community Edition. Для других сред разработки, например, Visual Studio Code, процесс может отличаться.

В среде разработки PyCharm Community Edition создаётся проект с выбранной версией интерпретатора Python (в данном случае версией 3.12) и с использованием виртуального окружения venv. Виртуальное окружение позволяет устанавливать сторонние модули и библиотеки. Кроме того, виртуальное окружение создаёт изолированную среду. Это значит, что библиотеки и модули устанавливаются только в venv только для этого проекта и не затрагивают другие проекты. Это позволяет избежать проблем с конфликтом разных версий инструментов в разных проектах. Для установки сторонних инструментов используется пакетный менеджер pip: в терминале прописывается команда: pip install название_библиотеки. После этого библиотеку (или модуль) можно использовать. Обычный пользователь избавлен от необходимости это делать, так как пользуется созданным исполняемым файлом qr_algorithm.exe. Но если в программе требуется что-то менять, то необходимо осуществить эту установку любым доступным способом.

Структура программы состоит из функций: главной функции main, основных функций для реализации численных методов и нескольких вспомогательных функций. Функция – это блок кода, который выполняет определённую задачу и может быть вызван в различных частях программы. Они повышают читаемость кода, разделяя его на составляющие. В начале каждой функции написаны докстринги (docstrings). Это комментарии, которые описывают суть функции и её параметры. Во всем коде в целом очень много комментариев для облегчения его понимания.

```
13    def sign(x):...
      6 usages
22    def multiply_two_matrices(matrix_a, matrix_b):...
      1 usage
38    def qr_decomposition_gram_schmidt_process(matrix_a):...
      1 usage
93    def qr_decomposition_givens_turn(matrix_a):...
      2 usages
122   def qr_algorithm(matrix_a, eps, qr_decomposition):...
      1 usage
138   def jacobi_rotation(matrix_a, eps):...
      1 usage
174   def gauss_method(matrix_a, b):...
248   def main():...
339   main()
340   input()  # чтобы консоль не закрылась
```

Рисунок 12. Схема всех функций программы.

Главная функция main() вводит от пользователя данные и выводит их ему, а также вызывает все другие функции. Именно она запускается в конце кода программы. Функции, осуществляющие QR-разложение: qr_decomposition_gram_schmidt_process() – по процессу Грама-Шмидта, qr_decomposition_givens_turn() – с помощью поворота Гивенса. Функция qr_algorithm() осуществляет QR-алгоритм, в качестве одного из своих параметров она принимает выбранное QR-разложение – qr_decomposition, которое является ссылкой либо на функцию qr_decomposition_gram_schmidt_process(), либо на функцию qr_decomposition_givens_turn(). В функции main() будет выбрано и то, и другое разложение. Функция jacobi_rotation() осуществляет метод вращений Якоби. Функция gauss_method() осуществляет метод Гаусса для вычисления собственных векторов по найденным ранее собственным значениям. Функция sign() является вспомогательной и реализует математическую функцию $sign(x)$. Функция multiply_two_matrices() является вспомогательной и

осуществляет умножение двух матриц друг на друга. Так как все матрицы в программе будут квадратными, умножение будет всегда определено.

Также стоит отметить использование в коде таких синтаксических конструкций как генераторы списков (list comprehension). Они есть в немногих языках программирования, поэтому вызывают сложности в понимании. Генераторы списков позволяют существенно сокращать код, записывая циклы, обрабатывающие списки, в одну строчку. Они имеют следующую структуру в самом простом случае: [способ формирования значения for переменная in итерируемый объект]. В результате создаётся список (list) из элементов итерируемого объекта, перебранных в цикле for.

```
eigenvalues = [matrix_a[i][i] for i in range(n)]
return eigenvalues
```

Рисунок 13. Простейший генератор списков в методе вращений Якоби.

Например, в данном случае (рис. 13) будет создан список из элементов матрицы matrix_a, стоящих на главной диагонали, так как используются одинаковые индексы [i][i], которые перебираются в цикле for с помощью функции range(n). Если бы генератор списков не использовался, код выглядел бы так:

```
eigenvalues = []
for i in range(n):
    eigenvalues.append(matrix_a[i][i])
return eigenvalues
```

Рисунок 14. Код, аналогичный коду на рисунке 8, но без использования генератора списков.

Наглядно видно, что в данном случае (рис. 14) генератор списков позволил сократить три строчки кода в одну. Таким образом, генераторы списков являются достаточно эффективным инструментом для сокращения кода.

3. Исследовательская часть

3.1. Решение с точностью $\varepsilon = 10^{-3}$

Ниже представлено решение задачи при $\varepsilon = 10^{-3}$ тремя алгоритмами: QR-алгоритмом с использованием QR-разложения по процессу Грама-Шмидта, QR-алгоритмом с использованием QR-разложения с помощью поворота Гивенса, методом вращений Якоби.

```
eps = 0.001

Собственные значения,
найденные QR-алгоритмом для QR-разложения
по процессу Грама-Шмидта:
L1 = 7.000
L2 = 2.000
L3 = -1.000
L4 = 1.000
L5 = 1.000
```

Рисунок 15. Решение QR-алгоритмом с использованием QR-разложения по процессу Грама-Шмидта при $\varepsilon = 10^{-3}$.

```
Собственные значения,
найденные QR-алгоритмом для QR-разложения
по повороту Гивенса:
L1 = 7.000
L2 = 2.000
L3 = -1.000
L4 = 1.000
L5 = 1.000
```

Рисунок 16. Решение QR-алгоритмом с использованием QR-разложения с помощью поворота Гивенса при $\varepsilon = 10^{-3}$.

```
Собственные значения,
найденные методом вращений Якоби:
L1 = 7.000
L2 = 2.000
L3 = -1.000
L4 = 1.000
L5 = 1.000
```

Рисунок 17. Решение методом вращений Якоби при $\varepsilon = 10^{-3}$.

```
Собственные векторы:

для собственного значения L1 = 7.000:
1.500
1.000
1.500
1.000
1.000

для собственного значения L2 = 2.000:
-1.000
1.000
-1.000
1.000
1.000

для собственного значения L3 = -1.000:
1.000
0.000
-1.000
0.000
0.000

для собственного значения L4 = 1.000:
0.000
1.000
0.000
1.000
-2.000

для собственного значения L5 = 1.000:
0.000
1.000
0.000
1.000
-2.000
```

Рисунок 18. Собственные векторы при $\varepsilon = 10^{-3}$.

3.2. Решение с точностью $\boldsymbol{\varepsilon = 10^{-6}}$

Ниже представлено решение задачи при $\varepsilon = 10^{-6}$ тремя алгоритмами: QR-алгоритмом с использованием QR-разложения по процессу Грама-Шмидта, QR-алгоритмом с использованием QR-разложения с помощью поворота Гивенса, методом вращений Якоби.

```
eps = 0.000001

Собственные значения,
найденные QR-алгоритмом для QR-разложения
по процессу Грама-Шмидта:
L1 = 7.000000
L2 = 2.000000
L3 = -1.000000
L4 = 1.000000
L5 = 1.000000
```

Рисунок 19. Решение QR-алгоритмом с использованием QR-разложения по процессу Грама-Шмидта при $\varepsilon = 10^{-6}$.

```
Собственные значения,
найденные QR-алгоритмом для QR-разложения
по повороту Гивенса:
L1 = 7.000000
L2 = 2.000000
L3 = -1.000000
L4 = 1.000000
L5 = 1.000000
```

Рисунок 20. Решение QR-алгоритмом с использованием QR-разложения с помощью поворота Гивенса при $\varepsilon = 10^{-6}$.

```
Собственные значения,
найденные методом вращений Якоби:
L1 = 7.000000
L2 = 2.000000
L3 = -1.000000
L4 = 1.000000
L5 = 1.000000
```

Рисунок 21. Решение методом вращений Якоби при $\varepsilon = 10^{-6}$.

3.3. Сравнение численных методов

Решения получились одинаковыми. Это произошло из-за того, что данные численные методы считают очень похоже. Если выводить ответы с большим количеством знаков в дробной части, чем заданная точность ε, то

можно увидеть, что ответы на самом деле различны. Например, выведем ответы при заданной точности $\varepsilon = 10^{-3}$ для QR-алгоритма с использованием QR-разложения по процессу Грама-Шмидта и метода вращений Якоби:

```
eps = 0.001

Собственные значения,
найденные QR-алгоритмом для QR-разложения
по процессу Грама-Шмидта:
L1 = 7.0000000000
L2 = 1.9999994278
L3 = -0.9999997139
L4 = 1.0000002146
L5 = 1.0000000715
```

Рисунок 22. Решение QR-алгоритмом с использованием QR-разложения по процессу Грама-Шмидта при $\varepsilon = 10^{-3}$ с большим количеством знаков в дробной части.

```
Собственные значения,
найденные методом вращений Якоби:
L1 = 6.9999999977
L2 = 1.9999999997
L3 = -0.9999999785
L4 = 1.0000000026
L5 = 0.9999999785
```

Рисунок 23. Решение методов вращений Якоби при $\varepsilon = 10^{-3}$ с большим количеством знаков в дробной части.

Видно, что ответы получились различны, но при округлении до 3 знаков после точки, так как вводимая точность $\varepsilon = 0.001$, они будут одинаковыми.

Тогда проведем сравнение алгоритмов по времени работы, использовав модуль time, а также по количеству внешних итераций. Внешними итерациями называются итерации внешних циклических процессов. Для QR-алгоритма это итерации, на которых происходит QR-разложение: $A_i = Q_i R_i$. Для метода вращений Якоби это итерации, на которых происходят приближения к нулю всех элементов матрицы вне главной диагонали.

```
Сравнительная таблица:
+--------------------+------------------+--------------+--------------+
| Критерий сравнения | QR - Грамм-Шмидт | QR - Гивенс  | М. вр. Якоби |
+--------------------+------------------+--------------+--------------+
|  Время в секундах  |   0.0010297298   | 0.0029947758 | 0.0009980202 |
|  Кол-во итераций   |        11        |      11      |      3       |
+--------------------+------------------+--------------+--------------+
```

Рисунок 24. Сравнительная таблица алгоритмов при $\varepsilon = 10^{-3}$.

Метод вращений Якоби оказался самым быстрым для этих входных данных, а QR-алгоритм с использованием QR-разложения с помощью поворота Гивенса – самым медленным. Также у метода Якоби минимальное количество внешних итераций, а у QR-алгоритма – одинаковое количество для различных подходов QR-разложения.

Также проведём аналогичное сравнение методов при заданной точности $\varepsilon = 10^{-6}$:

```
Сравнительная таблица:
+--------------------+------------------+--------------+--------------+
| Критерий сравнения | QR - Грамм-Шмидт | QR - Гивенс  | М. вр. Якоби |
+--------------------+------------------+--------------+--------------+
|  Время в секундах  |   0.0019922256   | 0.0049867630 | 0.0009973049 |
|  Кол-во итераций   |        21        |      21      |      3       |
+--------------------+------------------+--------------+--------------+
```

Рисунок 25. Сравнительная таблица алгоритмов при $\varepsilon = 10^{-6}$.

Метод вращений Якоби оказался самым быстрым для этих входных данных, а QR-алгоритм с использованием QR-разложения с помощью поворота Гивенса – самым медленным. Также у метода Якоби минимальное количество внешних итераций, а у QR-алгоритма – одинаковое количество для различных подходов QR-разложения.

Таким образом, для решаемой задачи самым эффективным по времени и количеству внешних итераций оказался метод вращений Якоби, самым неэффективным – QR-алгоритм с использованием QR-разложения с помощью поворота Гивенса.

Заключение

Цель и задачи данной курсовой работы выполнены. Были изучены такие численные методы для нахождения собственных значений как QR-алгоритм и метод вращений Якоби. Было рассмотрено два подхода для реализации QR-разложения: процесс Грама-Шмидта и поворот Гивенса. Также было изучено нахождение собственных векторов, соответствующих найденным собственным значениям, с помощью метода Гаусса. На языке программирования Python была написана программа, реализующая решение задачи указанными численными методами и подходами. Было проведено сравнение численных методов по результатам, времени работы и количеству итераций.

Подводя итоги, преимущество всех рассмотренных методов заключается в очень высокой точности нахождения решений, которая важна во многих сферах, особенно в обработке физических сигналов. Недостатком этих методов является большой расход памяти, требуемый для хранения данных.

Для решаемой задачи самым эффективным по времени работы оказался метод вращений Якоби. Однако он обладает существенным недостатком – его применение ограничивается только симметричными матрицами. QR-алгоритм имеет чуть более широкое применение, но он менее эффективен по времени работы. Поворот Гивенса оказался самым неэффективным по времени работы подходом для реализации QR-разложения, которое является основой QR-алгоритма. Учитывая, что условия применения поворота Гивенса и процесса Грама-Шмидта одинаковые, QR-разложение лучше реализовывать только с помощью процесса Грама-Шмидта.

Дальнейшая модификация программы может заключаться в добавлении других численных методов для нахождения собственных значений, в уменьшении расхода используемой памяти с помощью использования массивов библиотеки numpy, в добавлении графического интерфейса с помощью библиотеки customtkinter.

Список литературы

1. Авдюшев, В. А. Метод вращений Якоби // Астро. – 2013. [электронный ресурс] – URL: https://astro.tsu.ru/OsChMet/5_2.html (дата обращения: 28.10.2024).

2. Байтель, М. В поиске собственных значений (матриц) // Хабр. – 2024. [электронный ресурс] – URL: https://habr.com/ru/companies/ruvds/articles/845652/ (дата обращения: 20.10.2024).

3. Бахвалов, Н. С. Численные методы / Н. С. Бахвалов, Н. П. Жидков, Г. М. Кобельков. – М.: Лаборатория знаний, 2020. – 636 с. – ISBN 978-5-00101-836-0. – Текст : электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/126099 (дата обращения: 30.10.2024). – Режим доступа: для авториз. пользователей.

4. Вержбицкий, В. М. Основы численных методов: учебник для вузов / В. М. Вержбицкий. – М.: Высш. шк., 2002. – 840 с. – ISBN 5-06-004020-8.

5. Кувайскова, Ю. Е. Численные методы: учебное пособие / Ю. Е. Кувайскова. – Ульяновск : УлГТУ, 2024. – 206 с. – ISBN 978-5-9795-0000-0.

6. Ландовский, В. В. Численные методы : учебное пособие / В. В. Ландовский. — Новосибирск : НГТУ, 2023. — 72 с. — ISBN 978-5-7782-4904-2. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/404582 (дата обращения: 10.11.2024). — Режим доступа: для авториз. пользователей.

7. Панкратов В. А., Тверская Е. С. Проведение семинарского занятия “Метод вращений Гивенса” // Modern European Researches. 2022. №3. URL: https://cyberleninka.ru/article/n/provedenie-seminarskogo-zanyatiya-metod-vrascheniy-givensa (дата обращения: 05.11.2024).

8. Панкратов В. А., Тверская Е. С. Проведение семинарского занятия “Метод отражений хаусхолдера” // Modern European Researches. 2022. №1. URL: https://cyberleninka.ru/article/n/provedenie-seminarskogo-zanyatiya-metod-otrazheniy-hausholdera (дата обращения: 05.11.2024).

9. Семушин, И. В. Сборник лабораторных работ и контрольных, тестовых заданий по курсу “Вычислительная линейная алгебра” / И. В. Семушин, Г. Ю. Куликов. – Ульяновск: УлГУ, 2000. – 134 с. – ISBN 5-89146-139-0. – Текст : электронный // Венец: электронно-библиотечная система. – URL: https://lib.ulstu.ru/venec/2000/4_Semushin_Kulikov.pdf (дата обращения: 25.10.2024).

10. Смирнова, А. С. QR-алгоритм // АлгоВики. – 2021. [электронный ресурс] – URL: https://algowiki-project.org/ru/QR-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC (дата обращения: 18.10.2024).

Приложение

```
"""
Решение проблемы собственных значений с помощью численных методов: QR-алгоритм и метод вращений Якоби. QR-алгоритм
использует QR-разложение, которое реализовано двумя подходами: процессом Грама-Шмидта и поворотом Гивенса.
Для найденных собственных значений производится поиск соответствующих им собственных векторов с помощью метода Гаусса.
"""

import math  # входит в стандартную библиотеку
import numpy as np  # нужно для вычисления обратной матрицы
import prettytable  # нужно для красивого вывода таблиц
import time  # нужно для засечения времени работы алгоритмов


def sign(x):
    """
    Вспомогательная функция: математическая функция sign(x).
    """
    if x > 0:
        return 1
    if x < 0:
        return -1
    return 0


def multiply_two_matrices(matrix_a, matrix_b):
    """
    Вспомогательная функция: умножает матрицы matrix_a и matrix_b друг на друга и возвращает итоговую матрицу matrix_c.
    Предполагается, что умножение матриц определено: кол-во столбцов matrix_a = кол-ву строк matrix_b.
    """
    # размеры матриц matrix_a и matrix_b: кол-во строк и столбцов соответственно
    m_a, n_a, m_b, n_b = len(matrix_a), len(matrix_a[0]), len(matrix_b), len(matrix_b[0])  # предполагается n_a = m_b
    m_c, n_c = m_a, n_b  # размеры итоговой матрицы matrix_c: кол-во строк и столбцов соответственно
    matrix_c = []  # создадим матрицу matrix_c, заполненную нулями
    for i in range(m_c): matrix_c.append([0] * n_c)
    # при умножении двух матриц строки левой матрицы покоординатно умножаются на столбцы правой матрицы
    for i in range(m_c):
        for j in range(n_c):
            # элемент matrix_c[i][j] равен покоординатному умножению i-ой строки matrix_a на j-ый столбцец matrix_b
            matrix_c[i][j] = sum([matrix_a[i][k] * matrix_b[k][j] for k in range(n_a)])
    return matrix_c


def qr_decomposition_gram_schmidt_process(matrix_a):
    '''
```

```
    Функция осуществляет QR-разложение вещественной матрицы matrix_a на ортогональную матрицу matrix_q
    и верхнетреугольную матрицу matrix_r: matrix_a = matrix_q * matrix_r. Для реализации QR-разложения используется
    процесс Грама-Шмидта. Далее в комментариях будет описан ход этого процесса.
    '''
    n = len(matrix_a)  # размер матрицы, предполагается, что она квадратная
    vector_columns = [[matrix_a[i][j] for i in range(n)] for j in range(n)]  # вектор-столбцы a_1, a_2, ..., a_n
    # для вектор-столбцов a_1, a_2, ..., a_n нужно получить систему ортогональных векторов b_1, b_2, ..., b_n
    # система ортогональных векторов - это система векторов, где все векторы попарно ортогональны, т.е. перпендикулярны
    orthogonal_vectors = [vector_columns[0]]  # b_1 = a_1
    # далее b_j (2 <= j <= n) рассчитывается по следующим формулам:
    # b_2 = a_2 - proj_b_1_a_2; b_3 = a_3 - proj_b_1_a_3 - proj_b_2_a_3; ...
    # b_n = a_n - proj_b_1_a_n - proj_b_2_a_n - ... - proj_b_n-1_a_n
    # proj_b_a - проекция вектора a на вектор b, proj_b_a = scal_a_b / scal_b_b * b
    # scal_a_b - скалярное произведение векторов a(x1, y1, z1) и b(x2, y2, z2), scal_a_b = x1 * x2 + y1 * y2 + z1 * z2
    for j in range(1, n):  # в списках индексация с нуля => старт j с 1, а не с 2
        a_j = vector_columns[j]
        projections_b_j = []  # посчитаем для b_j проекции proj_b_1_a_j, proj_b_2_a_j, ..., proj_b_j-1_a_j
        for i in range(j):
            b_i = orthogonal_vectors[i]
            scal_a_j_b_i = sum([a_j[k] * b_i[k] for k in range(n)])
            scal_b_i_b_i = sum([b_i[k] * b_i[k] for k in range(n)])
            scal_mult = scal_a_j_b_i / scal_b_i_b_i
            proj_b_i_a_j = [scal_mult * elem for elem in b_i]
            projections_b_j.append(proj_b_i_a_j)
        b_j = [a_j[i] - sum([projections_b_j[k][i] for k in range(j)]) for i in range(n)]
        orthogonal_vectors.append(b_j)
    # из ортогональных векторов b_1, b_2, ..., b_n нужно получить нормированные векторы e_1, e_2, ..., e_n
    # нормированный (единичный) вектор - это вектор единичной длины, он получается делением вектора на его норму
    # e_j = b_j / ||b_j||, где ||b_j|| - норма вектора, ||b_j|| = sqrt(sum(elem_b_j**2)) для 1 <= j <= n
    normed_vectors = []
    for j in range(n):
        b_j = orthogonal_vectors[j]
        b_j_norm = sum([elem ** 2 for elem in b_j]) ** 0.5
        e_j = [elem / b_j_norm for elem in b_j]
        normed_vectors.append(e_j)
    # теперь найдем ортогональную матрицу matrix_q и верхнетреугольную матрицу matrix_r
    # ортогональная матрица - это квадратная матрица с вещественными элементами, результат умножения которой на
    # транспонированную матрицу равен единичной матрице:
    # matrix_q * matrix_q_transposed = matrix_q_transposed * matrix_q = E
    # верхнетреугольная матрица - это матрица, у которой все элементы, стоящие ниже главной диагонали, равны 0
    # столбцы ортогональной матрицы matrix_q формируются из векторов e_j для 1 <= j <= n
    matrix_q = [[normed_vectors[j][i] for j in range(n)] for i in range(n)]
    # найдем верхнетреугольную матрицу matrix_r из выражения matrix_a = matrix_q * matrix_r
    # умножим обе части этого уравнения на matrix_q ** (-1) слева
```

```
    # тгд: matrix_q ** (-1) * matrix_a = matrix_q ** (-1) * matrix_q * matrix_r
    # учитывая, что: matrix_q ** (-1) * matrix_q = E, где E - единичная матрица => E * matrix_r =
matrix_r
    # получаем: matrix_r = matrix_q ** (-1) * matrix_a
    # чтобы не искать обратную матрицу matrix_q ** (-1), можно воспользоваться св-вом
ортогональной матрицы matrix_q:
    # matrix_q ** (-1) = matrix_q_transposed, где matrix_q_transposed - транспонированная матрица
    # тгд верхнетреугольную матрицу matrix_r получаем по ф-ле: matrix_r = matrix_q_transposed *
matrix_a
    matrix_q_transposed = [[matrix_q[i][j] for i in range(n)] for j in range(n)]
    matrix_r = multiply_two_matrices(matrix_q_transposed, matrix_a)
    return matrix_q, matrix_r


def qr_decomposition_givens_turn(matrix_a):
    """
    Функция осуществляет QR-разложение вещественной матрицы matrix_a на ортогональную
матрицу matrix_q
    и верхнетреугольную матрицу matrix_r: matrix_a = matrix_q * matrix_r. Для реализации QR-
разложения используется
    поворот Гивенса. Далее в комментариях будет описан ход этого процесса.
    """
    n = len(matrix_a)
    matrix_r = matrix_a.copy()  # матрица matrix_r будет постепенно формироваться путем
умножений исходной матрицы
    # matrix_a на матрицу Гивенса g - такое умножение g * matrix_r называется поворотом Гивенса
    g = [[1 if i == j else 0 for j in range(n)] for i in range(n)]  # g - это единичная матрица E, но: (см. ниже
цикл)
    # чтобы привести matrix_r к верхнетреугольному виду, нужно обнулять эл-ты под главной
диагональю
    for j in range(n):  # в столбце j
        for i in range(j + 1, n):  # будем обнулять элементы a_i под эл-том a_j на главной диагонали в
этом столбце
            a_j, a_i = matrix_r[j][j], matrix_r[i][j]  # a_j - опорный эл-т, a_i - обнуляемый эл-т
            c = a_j / ((a_j ** 2 + a_i ** 2) ** 0.5)  # для этого найдём косинус c
            s = (-a_i) / ((a_j ** 2 + a_i ** 2) ** 0.5)  # и синус s
            g[j][j] = g[i][i] = c  # на координатах (i; i) и (j; j) в матрице Гивенса стоит косинус c
            g[i][j], g[j][i] = s, -s  # на координатах (i; j) и (j; i) стоят синусы s: s и -s соответственно
            matrix_r = multiply_two_matrices(g, matrix_r)  # поворот Гивенса: обнуляем эл-т a_i в i строке
j столбце
            g[j][j] = g[i][i] = 1  # возвращаем g к виду единичной матрицы, так как для след эл-та g будет
другая
            g[i][j] = g[j][i] = 0
    # найдем ортогональную матрицу matrix_q из выражения matrix_a = matrix_q * matrix_r
    # умножим обе части этого уравнения на matrix_r ** (-1) справа
    # тгд: matrix_a * matrix_r ** (-1) = matrix_q * matrix_r * matrix_r ** (-1)
    # учитывая, что: matrix_r * matrix_r ** (-1) = E, где E - единичная матрица => matrix_q * E =
matrix_q
    # получаем: matrix_q = matrix_a * matrix_r ** (-1), тгд требуется найти обратную матрицу
matrix_r ** (-1)
    matrix_r_inv = np.linalg.inv(np.array(matrix_r)).tolist()  # обратная матрица matrix_r ** (-1)
    matrix_q = multiply_two_matrices(matrix_a, matrix_r_inv)
    return matrix_q, matrix_r
```

```
def qr_algorithm(matrix_a, eps, qr_decomposition):
    """
    Функция осуществляет QR-алгоритм для нахождения собственных значений: преобразует исходную матрицу matrix_a
    в верхнетреугольную путем применения QR-разложения на каждой итерации. В качестве QR-разложения используется то,
    которое передано - qr_decomposition. Далее в комментариях будет описан ход этого алгоритма.
    """
    n = len(matrix_a)
    cnt_iters = 0
    while True:  # можно также задавать количество итераций
        matrix_q, matrix_r = qr_decomposition(matrix_a)  # QR-разложение: matrix_a = matrix_q * matrix_r
        matrix_a = multiply_two_matrices(matrix_r, matrix_q)  # матрица matrix_a меняется
        cnt_iters += 1
        if max([abs(matrix_a[i][j]) for j in range(n) for i in range(j + 1, n)]) < eps:  # завершается, когда max по abs
            break  # эл-т среди эл-тов под главной диагональю близок к 0, то есть меньше заданной точности eps
    eigenvalues = [matrix_a[i][i] for i in range(n)]  # собственные значения находятся на главной диагонали
    return cnt_iters, eigenvalues


def jacobi_rotation(matrix_a, eps):
    """
    Функция осуществляет метод вращений Якоби для нахождения собственных значений: преобразует исходную матрицу matrix_a
    в диагональную. Предполагается, что матрица симметричная. Далее в комментариях будет описан ход этого алгоритма.
    """
    n = len(matrix_a)
    cnt_iters = 0
    # выполняем, пока max по abs эл-т среди внедиагональных эл-тов не достиг 0, то есть больше заданной точности eps
    while max([abs(matrix_a[ii][jj]) for ii in range(n) for jj in range(n) if ii != jj]) >= eps:
        for j in range(n - 1):  # matrix_a[j][j] - опорный эл-т, нет смысла брать опорным правый нижний
            for k in range(j + 1, n):  #  для него стремимся обнулить симметричные эл-ты matrix_a[k][j] и matrix_a[j][k]
                if abs(matrix_a[j][k]) < eps:  # т. к. они симметричны, достаточно проверки одного
                    continue
                # составим матрицу вращения matrix_j по ф-лам ниже
                if matrix_a[j][j] == matrix_a[k][k]:
                    theta = math.pi / 4
                    c = math.cos(theta)
                    s = math.sin(theta)
                else:
                    tau = (matrix_a[j][j] - matrix_a[k][k]) / (2 * matrix_a[j][k])
                    t = sign(tau) / (abs(tau) + (1 + tau ** 2) ** 0.5)
                    c = 1 / ((1 + t ** 2) ** 0.5)
                    s = t * c
                matrix_j = [[1 if ii == jj else 0 for jj in range(n)] for ii in range(n)]  # это E, но: (см. ниже)
                matrix_j[j][j] = matrix_j[k][k] = c
```

```
            matrix_j[k][j], matrix_j[j][k] = s, -s
            # составим транспонированную матрицу вращения matrix_j_transpored
            matrix_j_transpored = [[matrix_j[ii][jj] for ii in range(n)] for jj in range(n)]
            # происходит двустороннее вращение: matrix_a = matrix_j * matrix_a * matrix_j_transpored
            matrix_a = multiply_two_matrices(matrix_j_transpored, matrix_a)
            matrix_a = multiply_two_matrices(matrix_a, matrix_j)
            # благодаря которому эл-ты эл-ты matrix_a[k][j] и matrix_a[j][k] уменьшаются, но они могут не обнулиться
            # за 1 итерацию, поэтому нужен внешний цикл while
        cnt_iters += 1
    eigenvalues = [matrix_a[i][i] for i in range(n)]  # собственные значения находятся на главной диагонали
    return cnt_iters, eigenvalues


def gauss_method(matrix_a, b):
    """
    Функция осуществляет метод Гаусса для любых СЛАУ: несовместных, определённых, неопределённых. Для неопределённых
    СЛАУ некоторые произвольные неизвестные задаёт = 1 для красоты собственного вектора.
    """
    n = len(matrix_a)
    for i in range(n):  # объединяем матрицу matrix_a и вектор свободных членов b в расширенную матрицу
        matrix_a[i].append(b[i])
    # прямой ход: приводим расширенную матрицу matrix_a к верхнетреугольному виду
    for ii in range(n):  # будем обнулять неизвестную x_ii, для этого найдём строку, в к-рой x_ii != 0
        ind_row_x_ii = ii  # индекс строки, в которой x_ii != 0, изначально предполагаем, что такая x_ii в iiой строке
        for i in range(ii + 1, n):
            if abs(matrix_a[i][ii]) > abs(matrix_a[ind_row_x_ii][ii]):  # будем искать строку, в к-рой x_ii max по abs
                ind_row_x_ii = i
        # меняем строки с индексами ind_row_x_ii и ii, чтобы такая x_ii стояла в строке с индексом ii
        matrix_a[ii], matrix_a[ind_row_x_ii] = matrix_a[ind_row_x_ii], matrix_a[ii]
        elem_to_zero = matrix_a[ii][ii]  # обозначим такую x_ii за elem_to_zero
        if elem_to_zero == 0:  # если обнулять оказалось нечего, то пропускаем эту x_ii
            continue
        for i in range(ii + 1, n):  # для обнуления эл-та elem_to_zero в последующих строках нужно к этим строкам
            mult = -matrix_a[i][ii] / elem_to_zero  # прибавлять строку, в к-рой есть elem_to_zero, умноженную на mult
            for j in range(n + 1):
                matrix_a[i][j] += matrix_a[ii][j] * mult
    # определяем совместность или несовместность СЛАУ
    flag_matrix_a = "совместная"
    for i in range(n):
        if all(matrix_a[i][j] == 0 for j in range(n)) and matrix_a[i][n] != 0:
            flag_matrix_a = "несовместная"
            break
    if flag_matrix_a == "несовместная":
        print("СЛАУ не имеет решений")  # но такая ситуация невозможна для задачи, так как для
        # собственных значений всегда есть собственный вектор
        input()  # чтобы консоль не закрылась
```

```
      exit()  # завершение всей программы
   # если система совместна - её определённость или неопределённость
   flag_matrix_a = "определённая"
   for i in range(n):
      if all(matrix_a[i][j] == 0 for j in range(n)) and matrix_a[i][n] == 0:
         flag_matrix_a = "неопределённая"
         break
   vector_column_x = [None] * n
   if flag_matrix_a == "определённая":  # => СЛАУ имеет 1 реш. - начинаем обратный ход:
вычисляем значения неизвестных
      for i in range(n - 1, -1, -1):
         mult = matrix_a[i][i]
         s = 0
         for j in range(i + 1, n):
            s += matrix_a[i][j] * vector_column_x[j]
         vector_column_x[i] = (matrix_a[i][n] - s) / mult
   else:  # flag_matrix_a == "неопределённая" => СЛАУ имеет бесконечно много решений -
обратный ход реализован сложнее
      # тогда в матрице matrix_a некоторые неизвестные могут определяться однозначно, а
некоторые - быть любыми
      for ii in range(n):  # нужно n итераций, чтобы определить n неизвестных
         # ищем строку, в которой можно определить min неизвестных (но не 0 неизвестных)
         cnt_x_can_define_min, ind_i = 10000000000000000000000, None
         for i in range(n):
            cnt_x_can_define = sum([1 for j in range(n) if (matrix_a[i][j] != 0 and vector_column_x[j] ==
None)])
            if cnt_x_can_define != 0 and cnt_x_can_define < cnt_x_can_define_min:
               cnt_x_can_define_min = cnt_x_can_define
               ind_i = i
         if cnt_x_can_define_min == 1:  # если нашли строку, где нужно определить одно x
            s, ind_j = 0, None  # тгд оно определяется однозначно
            for j in range(n):
               if matrix_a[ind_i][j] != 0:
                  if vector_column_x[j] == None:
                     ind_j = j
                     mult = matrix_a[ind_i][j]
                  else:
                     s += matrix_a[ind_i][j] * vector_column_x[j]
            vector_column_x[ind_j] = (matrix_a[ind_i][n] - s) / mult
         else:  # иначе придадим одному неизвестному x произвольное значение, пусть это будет 1
для красоты
            for j in range(n):
               if matrix_a[ind_i][j] != 0 and vector_column_x[j] == None:
                  vector_column_x[j] = 1  # тгд в след итерации цикла for ii возможно другие
неизвестные снова
                  break  # будут определяться однозначно
   return vector_column_x


def main():
   """
   Главная функция: задаёт матрицу и вводит точность от пользователя, затем для этих данных
решается задача нахождения
   собственных значений и соответствующих им собственных векторов.
```

```
    """
    # задаём в коде программы матрицу matrix_a, для которой будет решаться задача
    matrix_a = [
        [2, 1, 3, 1, 1],
        [1, 2, 1, 1, 1],
        [3, 1, 2, 1, 1],
        [1, 1, 1, 2, 1],
        [1, 1, 1, 1, 2]
    ]
    # гарантируется, что матрица matrix_a удовлетворяет всем условиям применения методов
    n = len(matrix_a)  # размер квадратной матрицы
    # ввод точности от пользователя
    print("Введите точность eps от 10^-14 до 10^-1", "в формате [0.0...0[ненулевое число]],",
          "например: eps = 0.001", sep="\n", end="\n\n")
    print("Для точности eps < 10^-14", "решение задачи невозможно в силу",
          "ограниченных вычислительных возможностей Python.", sep="\n", end="\n\n")
    while True:
        try:
            eps_str = input("eps = ")
            eps = float(eps_str)
        except ValueError:
            print("\nВы ввели не число или использовали запятую.", "Введите ещё раз:", sep="\n",
end="\n\n")
        else:
            if not (10**(-14) <= eps <= 10**(-1)):
                print("\nВы ввели число вне диапазона [10^-14; 10^-1].", "Введите ещё раз:", sep="\n",
end="\n\n")
            elif eps_str.count('0') != (len(eps_str) - 2) or eps_str[-1] == "0":
                print("\nВы ввели число не в формате [0.0...0[ненулевое число]].",
                      "Введите ещё раз:", sep="\n", end="\n\n")
            else:
                break
    eps_signs = len(eps_str) - 2  # столько знаков после точки надо будет оставлять при выводе
ответов
    print()
    # нахождение собственных значений и засечение времени работы алгоритмов
    t1 = time.time()
    cnt_iters_qr_gram_schmidt_process, eigenvalues_qr_gram_schmidt_process =
qr_algorithm(matrix_a, eps,
                                                    qr_decomposition_gram_schmidt_process)
    t2 = time.time()
    cnt_iters_qr_givens_turn, eigenvalues_qr_givens_turn = qr_algorithm(matrix_a, eps,
qr_decomposition_givens_turn)
    t3 = time.time()
    cnt_iters_jacobi_rotation, eigenvalues_jacobi_rotation = jacobi_rotation(matrix_a, eps)
    t4 = time.time()
    # вывод собственных значений
    print("Собственные значения,", "найденные QR-алгоритмом для QR-разложения", "по процессу
Грама-Шмидта:", sep="\n")
    for i in range(n):
        print(f"L{i + 1} = {eigenvalues_qr_gram_schmidt_process[i]:.{eps_signs}f}")
    print()
    print("Собственные значения,", "найденные QR-алгоритмом для QR-разложения", "по повороту
Гивенса:", sep="\n")
```

```
    for i in range(n):
        print(f"L{i + 1} = {eigenvalues_qr_givens_turn[i]:.{eps_signs}f}")
    print()
    print("Собственные значения,", "найденные методом вращений Якоби:", sep="\n")
    for i in range(n):
        print(f"L{i + 1} = {eigenvalues_jacobi_rotation[i]:.{eps_signs}f}")
    print()
    # нахождение собственных векторов
    print("Собственные векторы:\n")  # возьмём собственные значения eigenvalues_jacobi_rotation
    for i in range(n):  # находим собственный вектор для каждого собственного значения
        e_val = eigenvalues_jacobi_rotation[i]
        matrix_a_slau = []  # составляем матрицу для СЛАУ (A - L * E) * x = 0
        for ii in range(n):
            matrix_a_slau.append([0] * n)
            for jj in range(n):
                matrix_a_slau[ii][jj] = matrix_a[ii][jj]
                if ii == jj:
                    matrix_a_slau[ii][jj] -= round(e_val)
        matrix_b_slau = [0] * n
        eigenvectors = gauss_method(matrix_a_slau, matrix_b_slau)
        print(f"для собственного значения L{i + 1} = {e_val:.{eps_signs}f}:")
        for e_vector in eigenvectors:
            print(f"{e_vector:.{eps_signs}f}")
        print()
    # вывод сравнительной таблицы - по времени и кол-ву итераций
    print("Сравнительная таблица:")
    comparison_table = prettytable.PrettyTable()
    comparison_table.field_names = ["Критерий сравнения", "QR - Грамм-Шмидт", "QR - Гивенс", "М.
вр. Якоби"]
    comparison_table.add_row(["Время в секундах", f"{t2-t1:.10f}", f"{t3-t2:.10f}", f"{t4-t3:.10f}"])
    comparison_table.add_row(["Кол-во итераций", cnt_iters_qr_gram_schmidt_process,
cnt_iters_qr_givens_turn,
                              cnt_iters_jacobi_rotation])
    print(comparison_table)


main()
input()  # чтобы консоль не закрылась
```