Санкт-Петербургский политехнический университет
Высшая школа прикладной математики
и вычислительной физики, ФизМех

Направление подготовки
«Прикладная математика и информатика»

Отчет по лабораторной работе №4
«Решение алгебраической проблемы собственных значений итерационными методами»

Дисциплина: «Численные методы»

Выполнил студент гр. 5030102/00003 Анищенко М.Д.

Преподаватель: Курц В.В.

Санкт-Петербург 2021

Оглавление

Глава 1

Формулировка задачи и её формализация

Собственные векторы и значения — основные характеристики матрицы. Поэтому их нахождение является важной задачей вычислительной математики. Методы нахождения можно разделить на те, которые решают частичную АПСЗ (находят max/min собственное число или максимальное близкое к заданному значению), и те, которые решают полную задачу (находят все собственные числа).

1.1 Формализация задачи

Пусть дана матрица A размера $n \times n$. Соответственно она имеет n собственных чисел λ. Требуется найти эти собственные числа с помощью итерационного метода Якоби.

1.2 Постановка задачи

1. Исследовать условия применимости метода и алгоритм его работы.
2. Найти все собственные числа заданным методом.
3. Исследовать влияние отделимости СЧ на скорость сходимости итерационного процесса.
4. Исследовать устойчивость АПСЗ.

Глава 2

Алгоритм метода и условия его применимости

Метод вращений Якоби применим к симметричным матрицам ($A = A^\top$).

2.1 Алгоритм метода

Итерационная формула для метода Якоби: $A_2 = T_{ij}^\top A_1 T_{ij}$, где T — матрица вращений. Напоминание о том, что такое матрица вращения:

$$T_{ij}(\phi) = \begin{array}{c} \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} & & i & & j \end{array} \\ \begin{pmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & c & \cdots & -s & \vdots \\ \vdots & \vdots & 1 & \vdots & \vdots \\ \vdots & s & \cdots & c & \vdots \\ 0 & 0 & \cdots & \cdots & 1 \end{pmatrix} \end{array},$$

где $c = \cos(\phi)$ и $s = \sin(\phi)$.
T_{ij} поворачивает вектор x в плоскости Ox_ix_j на угол ϕ.
$T_{ij}^{-1} = T_{ij}^\top$.
При этом действие $A_2 = T_{ij}^\top A_1 T_{ij}$ можно выполнить по следуюему алгоритму:

$$B = AT_{ij}$$

$$b_{kl} = \begin{cases} a_k l & \forall k, l \neq i, j \\ a_{ki}c + a_{kj}s & \forall k, l = i \\ -a_{ki}s + a_{kj}c & \forall k, l = j \end{cases}$$

$$\tilde{A} = T_{ij}^\top B$$

$$\tilde{a}_{kl} = \begin{cases} b_k l & \forall l, k \neq i.j \\ b_{il}c + b_{jl}s & \forall l, k = i \\ -b_{il}s + b_{jl}c & \forall l, k = j \end{cases}$$

Введём следующие обозначения:
Норма Фробениуса: $||A||_F = \sqrt{\sum\limits_{i,j} a_{ij}^2}$.

$||A||_F^2 = \sum\limits_{i=1}^{n} a_{ii}^2 + \sum\limits_{i \neq j} a_{ij}^2 = S_1(A) + S_2(A)$.

Угол поворота ϕ для матрицы T должен удовлетворять условию: $\tan(2\phi) = \frac{2a_{ij}^{(1)}}{a_{ii}^{(1)} - a_{jj}^{(1)}}$.

Условие остановки: $S_2(A_k) < \varepsilon$

2.1.1 Алгоритм с выбором оптимального элемента

$A_2 = T_{ij}^\top A_1 T_{ij}$
$r_k = \sum\limits_{j=1, j\neq k}^{n} (a_{kj}^{(1)})^2 \Rightarrow S_2(A_1) = \sum\limits_k r_k$.
$i_0 : r_{i_0} = \max\limits_k r_k$
$j_0 : |a_{i_0 j_0}^{(1)}| = \max\limits_{i, j \neq i_0} |a_{i_0 j}^{(1)}|$.

Глава 3

Анализ задачи

АПСЗ наиболее устойчива, если $A = A^\top$. Зададим матрицу как $A = QDQ^\top$, где Q — ортогональная матрица, D — диагональная. Также такой способ поможет контролировать отделимость собственных чисел.

Глава 4

Тестовый пример

Возьмём для примера матрицу 3×3:

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix} Q = \begin{pmatrix} -0.7092 & -0.1260 & -0.6936 \\ -0.3030 & -0.8339 & 0.4612 \\ -0.6365 & 0.5373 & 0.5533 \end{pmatrix} A_1 = \begin{pmatrix} 1.9781 & -0.5348 & -0.8352 \\ -0.5348 & 2.1209 & 0.0623 \\ -0.8352 & 0.0623 & 1.9009 \end{pmatrix}$$

Будем действовать согласно алгоритму, устанавливая погрешность $\varepsilon = 0.01$. Нумерация начинается с нуля.

1. $$i_0 = 0, j_0 = 2$$
$$\tan(2\phi) = -21.636455; \phi = -0.762305.$$
$$A_2 = \begin{pmatrix} 2.775661 & -0.429850 & 0 \\ -0.429850 & 2.120933 & -0.324269 \\ 0 & -0.324269 & 1.103405 \end{pmatrix}$$

2. $$i_0 = 1, j_0 = 0$$
$$\tan(2\phi) = 1.313064; \phi = 0.459963.$$
$$A_3 = \begin{pmatrix} 2.988610 & 0 & 0.143948 \\ 0 & 1.907984 & -0.290567 \\ 0.143948 & -0.290567 & 1.103405 \end{pmatrix}$$

3. $$i_0 = 2, j_0 = 1$$
$$\tan(2\phi) = 0.722284; \phi = 0.312763.$$
$$A_4 = \begin{pmatrix} 2.988610 & -0.044291 & 0.136965 \\ -0.044291 & 2.001947 & 0 \\ 0.136965 & 0 & 1.009443 \end{pmatrix}$$

4.

$$i_0 = 0, j_0 = 2$$

$$\tan(2\phi) = 0.138406; \phi = 0.068766.$$

$$A_5 = \begin{pmatrix} 2.998044 & -0.044186 & 0 \\ -0.044186 & 2.001947 & 0.003043 \\ 0 & 0.003043 & 1.000009 \end{pmatrix}$$

Отсюда $\lambda_1 = 2.998044, \lambda_2 = 2.001947, \lambda_3 = 1.000009$. Заданные нами собственные числа равны $\lambda_1^* = 3, \lambda_2^* = 2, \lambda_3^* = 1$. Как мы можем наблюдать, собственные значения, полученные с помощью метода Якоби, заметно приблизились к точным за 4 итерации.

Глава 5

Контрольные тесты

Создадим две матрицы с хорошей и плохой отделимостями, равными 0.1 и 0.99 соответственно, и посмотрим, как отделимость влияет на количество итераций, необходимое для нахождения собственных чисел с заданной точностью. Точность мы будем менять от 10^{-15} до 10^{-3}.

Глава 6

Модульная структура программы

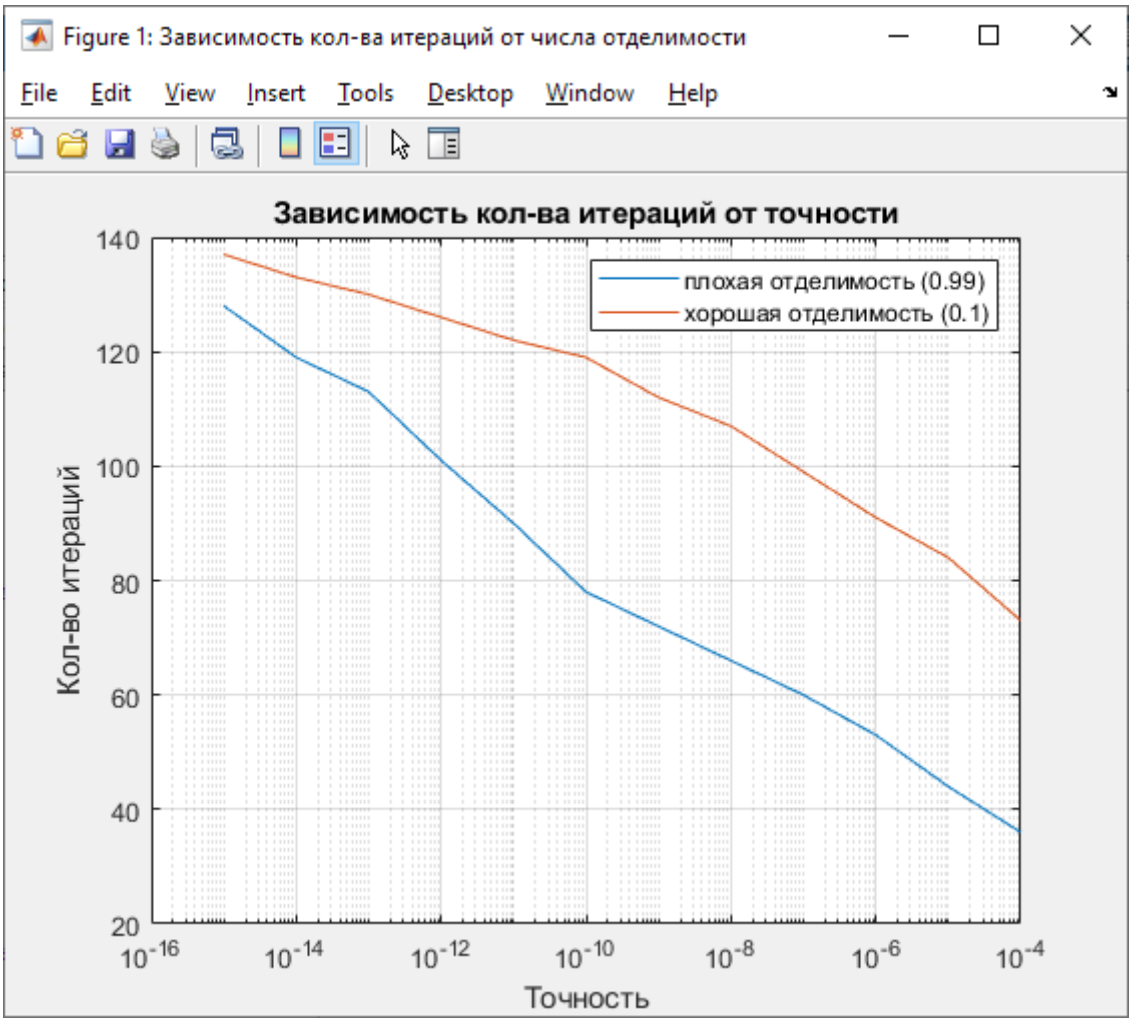
- double** CreateMatrix(int n, int m) — функция выделения памяти под матрицу размера $n \times m$.
- void DestroyMatrix(double** A, int n) — функция удаления матрицы и очистки памяти.
- double** ReadMatrix(FILE* matrixfile, int n) — функция считывания матрицы из файла.
- void ChoosingTheOptimalElement(double** matrix, int n, int* i0, int* j0) — функция выбора оптимального элемента $a_{i_0 j_0}$.
- double getPhi(double** matrix, int i, int j) — функция нахождения угла поворота ϕ.
- double** TAT(double** matrix, int n, double phi, int i, int j) — функция матричного умножения $T^\top AT$.
- double StopCondition(double** matrix, int n) — функция проверки выполнения уловия остановки.

- void GetSeparability(double** matrix, FILE* separabilityFile, int n, int iters) — функция записи в файл количества итераций, соответствующего конкретному показателю отделимости.
- void GetEigenValues(double** matrix, int n, FILE* EigenValues) — функция записи в файл собственных значений, найденных с помощью выбранного метода.
- double** Jacobi(double** matrix, int n, int* iters) — функция, осуществляющая выполнение поиска собственных значений матрицы с помощью метода Якоби.

Глава 7

Численне результаты

Зададим 2 разных показателя отделимости (хорошая отделимость = 0.1 и плохая = 0.99) и будем считать количество итераций при разной точности.



Из графика мы видим, что количество итераций для хорошей отделимости больше, чем для плохой, то есть, чем ближе собственные значения матрицы друг к другу, тем меньше вычислений нам потребуется.

Глава 8

Общие выводы

В данной работе был реализован метод Якоби. Анализируя наши исследования, заметим, что данный метод для симметричных матриц достаточно быстро находит собственные значения матрицы. Также мы выяснили, что при хорошей отделимости метод сходится медленнее, чем при плохой, когда собственные числа матрицы находятся близко друг к другу.