Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Алтайский государственный технический университет

им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой \_\_\_\_\_

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Отчет

по лабораторной работе № 5

"Полная проблема собственных чисел."

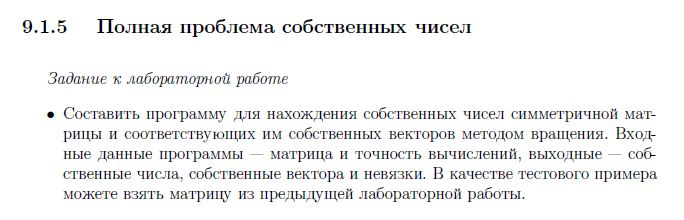
по дисциплине "Вычислительная математика"

Студент группы ПИ-72 Н. А. Черемнов

Преподаватель С. А. Кантор

Барнаул 2020

Задание:



Алгоритм:

Основная логика алгоритма:

1. Получать индекс k, l элемента, который будем исключать. Пока такой элемент есть:

1.1. Получить матрицу поворота Ukl

1.2. Выполнить D = DU (изначально D = E, по завершению получим ПUi0

1.3. Выполнить итерацию Ai+1 = UTAiU

2. Восстановить результаты:

2.1. Получить собственные числа, которые соответствуют диагональным элементам матрицы An.

2.2. Получить соответствующие собственные векторы: если Anii = λi, то ему соответствует собственный вектор xn = (0, 0, … , λi, …, 0) – где i-ая координата = λi,, а остальные координаты = 0. Тогда собственный вектор x соответствующий i-му собственному числу исходной матрицы равен i-му столбцу матрицы D. Для удобства представления и вычисления невязки, каждый столбец переписывается в соответствующий вектор.

Алгоритм выбора k, l:

1. Изначально посчитать суммы квадратов недиагональных элементов каждой строки.

2. При выборе очередного k, l взять для рассмотрения строку с максимальной суммой.

3. В выбранной строке взять максимальный по модулю недиагональный элемент.

4. После очередной итерации алгоритма обновить суммы для k-ой и l-ой строк.

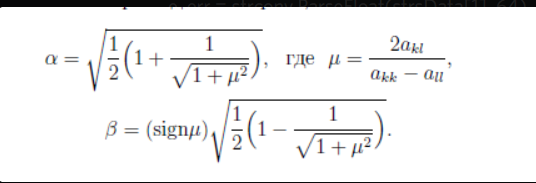
Алгоритм получения матрицы U:

1. Получить значения α и β.

2. Заполнить как единичную матрицу, за исключением того, что Ukk = Ull = α, Ukl = -β, Ulk = β.

3. Затем матрицу можно обновлять зануляя недиагональные ненулевые элементы индексов k и l, и приравнивая к 1 диагональный элементы с данными индексами, и затем выполнить для новых k и l Ukk = Ull = α, Ukl = -β, Ulk = β.

α и β получаем по формулам:



D = DU: при умножении изменяются только столбцы k, l, поэтому они пересчитываются, и записываются в матрицу D. остальные элементы остаются неизменными.

Поворот матрицы:

1. При A = AU изменяются только элементы столбцов k и l, поэтому они пересчитываются.

2. Аналогично при A = UtA изменяются только строки k, l, поэтому пересчитываются только они, таким образом, чтоб использовать матрицу U в качестве U­T.

Тесты:

Исходные данные в файле имеют вид:

n e

a11 … a1n

an1 … ann

x1 … xn

Где n – размерность матрицы, e – точность, A – матрица.

1) Тестовый пример.

Исходные данные:

4 0.001

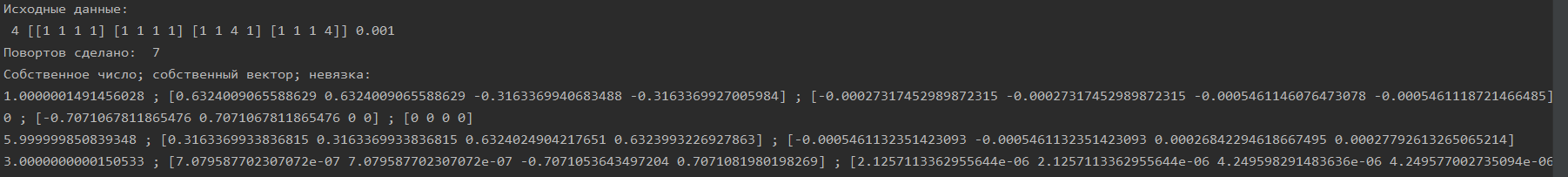
1 1 1 1

1 1 1 1

1 1 4 1

1 1 1 4

Результат:



равносильно:

Исходные данные:

4 [[1 1 1 1] [1 1 1 1] [1 1 4 1] [1 1 1 4]] 0.001

Повортов сделано: 7

Собственное число; собственный вектор; невязка:

1.0000001491456028 ; [0.6324009065588629 0.6324009065588629 -0.3163369940683488 -0.3163369927005984] ; [-0.00027317452989872315 -0.00027317452989872315 -0.0005461146076473078 -0.0005461118721466485]

0 ; [-0.7071067811865476 0.7071067811865476 0 0] ; [0 0 0 0]

5.999999850839348 ; [0.3163369933836815 0.3163369933836815 0.6324024904217651 0.6323993226927863] ; [-0.0005461132351423093 -0.0005461132351423093 0.00026842294618667495 0.00027792613265065214]

3.0000000000150533 ; [7.079587702307072e-07 7.079587702307072e-07 -0.7071053643497204 0.7071081980198269] ; [2.1257113362955644e-06 2.1257113362955644e-06 4.249598291483636e-06 4.249577002735094e-06]

Комментарий: найдены собственные числа 1, 0, 6, 3 и соответствующие им собственные векторы. Невязки для собственных векторов являются приемлемыми.

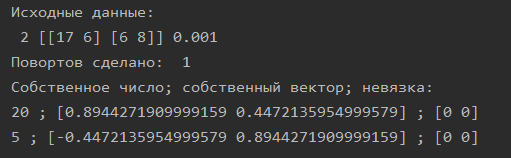
2) Исходные данные:

2 0.001

17 6

6 8

Результат:



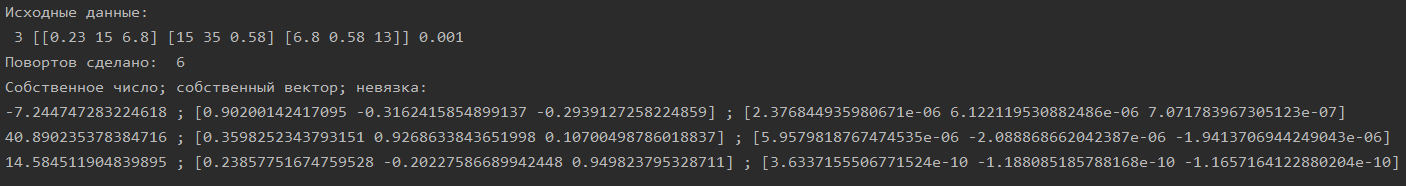
3) Исходные данные:

3 0.001

0.23 15 6.8

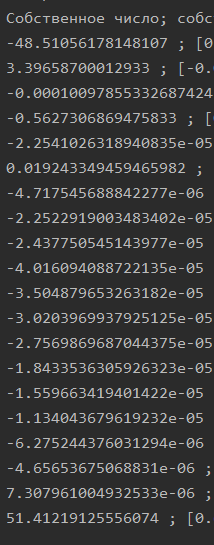
15 35 0.58

6.8 0.58 13



4) Большая симметричная матрица:

Hij = Hji = (i + j) / (i \* j + 1) при n = 20.



Поместил только область с найденными собственными числами, видно, что значительно больше итераций, однако не так много, как я предполагал. Собственные вектора и невязки не вывел, т.к. они длинные, но с виду невязки приемлемы.

Текст программы:

package main  
  
import (  
 "fmt"  
 "math"  
 "myReader"  
)  
  
func main() {  
 n, A, e, err := myReader.ReadM5("big.txt")  
 if err != nil {  
 panic(err)  
 }  
 oldA := make([][]float64, n)  
 for i := 0; i < n; i++{  
 oldA[i] = make([]float64, n)  
 copy(oldA[i], A[i])  
 }  
 fmt.Println("Исходные данные:\n", n, A, e)  
 eigenValues, eigenVectors, rotates := rotate(n, A, e, 1000000)  
 fmt.Println("Повортов сделано: ", rotates)  
 fmt.Println("Собственное число; собственный вектор; невязка:")  
 for i, elem := range eigenVectors {  
 fmt.Println(eigenValues[i], ";", elem, ";", r(n, oldA, eigenValues[i], elem))  
 }  
}  
  
//процедура вычисления невязки  
func r(n int, A [][]float64, l float64, x []float64) []float64 {  
 res := make([]float64, n)  
 for i := 0; i < n; i++ {  
 for j := 0; j < n; j++ {  
 res[i] += A[i][j] \* x[j]  
 }  
 }  
 for i := 0; i < n; i++{  
 res[i] -= l \* x[i]  
 }  
 return res  
}  
  
//процедура метода вращения  
//вход: размерность матрицы, матрица, точность  
//выход:  
func rotate(n int, A [][]float64, e float64, criticalIteration int) ([]float64, [][]float64, int){  
 r := make([]float64, n) //суммы недиагональных элементов каждой строки  
 for i := 0; i < n; i++ { //заполнить суммы  
 for j := 0; j < n; j++ {  
 if i != j {  
 r[i] += A[i][j] \* A[i][j]  
 }  
 }  
 }  
 D := make([][] float64, n) //создать изначальную матрицу D  
 for i := 0; i < n; i++ {  
 D[i] = make([]float64, n)  
 D[i][i] = 1  
 }  
 //основной итерационный цикл  
 rotatesCount := 0  
 for k, l := getkl(n, A, r, e); k != -1 && l != -1 && rotatesCount < criticalIteration; k, l = getkl(n, A, r, e) {  
 U := getU(n, A, k, l) //получить очередную матрицу поворота  
 D = du(n, D, U, k, l) //обновить матрицу D  
 A = rotation(n, A, k, l) //повернуть матрицу  
 r = rUpdate(n, A, r, k, l) //обновить суммы  
 rotatesCount++  
 }  
 eigenValues, eigenVectors := recoverResults(n, A, D)  
 return eigenValues, eigenVectors, rotatesCount  
}  
  
func recoverResults(n int, A [][]float64, D [][]float64)([]float64, [][]float64){  
 eigenValues := make([]float64, n)  
 for i := 0; i < n; i++{  
 eigenValues[i] = A[i][i] //проинициализировать собственные числа  
 }  
 eigenVectors := make([][]float64, n)  
 for i := 0; i < n; i++{  
 eigenVectors[i] = make([]float64, n) //очередной собственный вектор для iго собственного числа  
 for j := 0; j < n; j++{   
 eigenVectors[i][j] = D[j][i]  
 }  
 }  
 return eigenValues, eigenVectors  
}  
  
//процедура корректирования сумм r  
//вход: размерность матрицы, очередная матрица A, k, l, вектор сумм  
//выход: обновленный вектор сумм  
func rUpdate(n int, A [][]float64, r []float64, k int, l int) []float64{  
 r[k] = 0  
 r[l] = 0  
 for j := 0; j < n; j++ {  
 if j != k {  
 r[k] += A[k][j] \* A[k][j]  
 }  
 }  
 for j := 0; j < n; j++ {  
 if j != l {  
 r[l] += A[l][j] \* A[l][j]  
 }  
 }  
 return r  
}  
  
//C=AU  
func rotation(n int, A [][]float64, k int, l int)(C [][]float64){  
 a, b := getAlphBeta(A, k, l)  
 kCol := make([]float64, n)  
 lCol := make([]float64, n)  
 //сделали A = AU  
 for i := 0; i < n; i++ {  
 kCol[i] = A[i][k] \* a + A[i][l] \* b  
 lCol[i] = A[i][l] \* a - A[i][k] \* b  
 }  
 for i := 0; i < n; i++{  
 A[i][k] = kCol[i]  
 A[i][l] = lCol[i]  
 }  
 //сделаем A = UtA  
 for i := 0; i < n; i++ {  
 kCol[i] = A[k][i] \* a + A[l][i] \* b  
 lCol[i] = A[l][i] \* a - A[k][i] \* b  
 }  
 for i := 0; i < n; i++{  
 A[k][i] = kCol[i]  
 A[l][i] = lCol[i]  
 }  
 return A  
}  
  
//процедра D = DU  
//вход: n, матрицы D и U, числа k и l  
//выход:  
func du(n int, D [][]float64, U [][]float64, k int, l int) [][]float64 {  
 kCol := make([]float64, n)  
 lCol := make([]float64, n)  
 for i := 0; i < n; i++ {  
 for j := 0; j < n; j++{  
 kCol[i] += D[i][j] \* U[j][k]  
 lCol[i] += D[i][j] \* U[j][l]  
 }  
 }  
 for i := 0; i < n; i++ { //изменить матрицу D  
 D[i][k] = kCol[i]  
 D[i][l] = lCol[i]  
 }  
 return D  
}  
  
//формирования матрицы U  
//вход:  
//выход: матрица U  
func getU(n int, A [][]float64, k int, l int) [][]float64 {  
 U := make([][]float64, n)  
 a, b := getAlphBeta(A, k, l)  
 for i := 0; i < n; i++{ //заполним матрицу u  
 U[i] = make([]float64, n)  
 for j := 0; j < n; j++{  
 if i == j { //если диагональный элемент  
 if i == k || i == l {  
 U[i][j] = a  
 } else {  
 U[i][j] = 1  
 }  
 } else { //иначе недиагональный элемент  
 if i == k && j == l {  
 U[i][j] = -b  
 } else if i == l && j == k {  
 U[i][j] = b  
 } else {  
 U[i][j] = 0  
 }  
 }  
 }  
 }  
 return U  
}  
  
//процедура получения a, b  
func getAlphBeta(A [][]float64, k int, l int)(ap float64, bt float64){  
 if A[k][k] == A[l][l]{  
 return math.Sqrt(0.5), math.Sqrt(0.5)  
 }  
  
 ny := 2 \* A[k][l] / (A[k][k] - A[l][l])  
 ap = math.Sqrt(0.5 \* (1 + 1/ (math.Sqrt(1 + ny\*ny))))  
 bt = math.Sqrt(0.5 \* (1 - 1/(math.Sqrt(1 + ny\*ny))))  
 if ny < 0{  
 bt \*= -1  
 }  
 return ap, bt  
}  
  
//более топорный способ получения kl  
func fooGetkl(n int, A [][]float64, e float64) (int, int) {  
 k := -1  
 l := -1  
 for i := 0; i < n; i++{  
 for j := 0; j < n; j++ {  
 if i != j && (k == -1 || A[i][j] \* A[i][j] > A[k][l] \* A[k][l]){  
 k = i  
 l = j  
 }  
 }  
 }  
 if math.Abs(A[k][l]) < e {  
 return -1, -1  
 } else {  
 return k, l  
 }  
}  
  
//процедура выбора элемента akl  
//вход: размерность матрицы, матрица, массив сумм r, тончость e  
//выход: k, l  
func getkl(n int, A [][]float64, r []float64, e float64) (int, int){  
 k := -1  
 for i := 0; i < n; i++ {  
 if k == -1 || r[i] > r[k] {  
 k = i  
 }  
 }  
 l := -1  
 for j := 0; j < n; j++ { //найти минимальное слагаемое  
 if j != k && (l == -1 || A[k][j] \* A[k][j] > A[k][l] \* A[k][l]) { //если есть более большое слагаемое  
 l = j  
 }  
 }  
 if math.Abs(A[k][l]) < e { //если достигнута нужная точность  
 return -1, -1  
 } else {  
 return k, l  
 }  
}

Ответ на незакрытый вопрос из предыдущей лр:

Прежде всего хочу извиниться за то, что при вашей просьбе прислать отчет следующей лр побыстрее, я делаю это через неприемлемый срок. Поэтому постараюсь подробно описать суть вопроса и дать ответ.

Вопрос: если я правильно понял, то вопрос касается следующего примера (п 4. задания лр 4):

3 0.001 4

-5.7 -61.1 -32.9

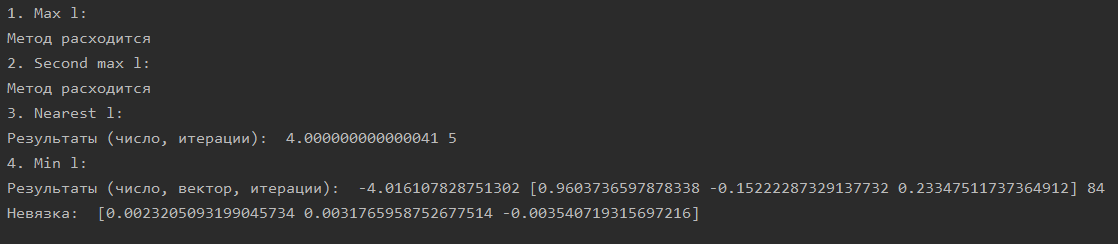
0.8 11.9 7.1

-1.1 -11.8 -7.2

1 1 1

И заключается в том, как с помощью программы подтвердить предположение о том, почему метод не сходится. Я предположил, что п.1 (max) и п.2 (второй max) алгоритма расходятся, причиной тому могут служить нарушение условия  , т.к. видно, что п.3 и п.4 отработали корректно.

Программой были получены следующие результаты:



Видно, что минимальное собственное число по модулю примерно равно числу, ближайшему к 5. При увеличении l0 (число, ближайшее собственное к которому мы ищем), ближайшим все так же остается 4. Т.е. мы видим ситуацию, когда |l1| = |l2|, что подтверждает мое предположение выше.