Министерство образования и науки Российской Федерации

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 № 4

"Частичная проблема собственных чисел."

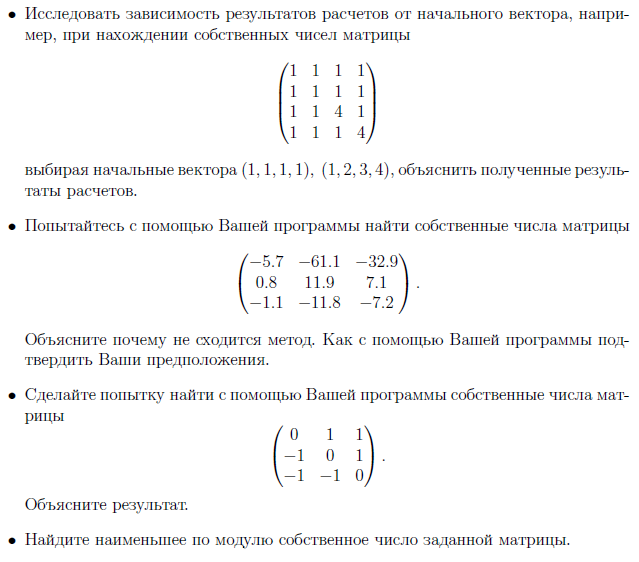
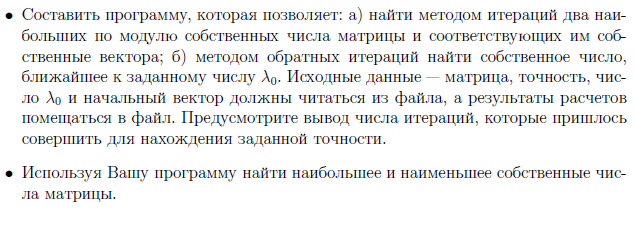
по дисциплине "Вычислительная математика"

Студент группы ПИ-72 Н. А. Черемнов

Преподаватель С. А. Кантор

Барнаул 2020

Задание:



Алгоритм:

1. Вычисление λ1:

1) Получить xn+1 = Ax0, λn = (xn, xn+1) / (xn, xn) – первая итерация.

2) xn+1=Axn, λn+1 = (xn, xn+1) / (xn, xn) – вторая итерация

3) Далее выполнять приближения xn, λn пока |λn - λn+1| >= ε либо не превышено число допустимых итераций. Причем, если число допустимых итераций не задано, то при увеличении расстояния между λn ­и λn+1 завершить процедуру с ошибкой сходимости. Так же через каждое заданное число n итераций выполнять нормирование вектора xn.

4) Получить собственный вектор e1 = xn / ||xn||, соответствующий найденному собственному числу λ.

2. Вычисление λ2:

1) Получить AT

2) Найти λ1 для AT и соответствующий собственный вектор g1

3) Получить y0 = (x0, g1) / (e1, g1)

4) Выполнить алгоритм п 1. для x0 = y0 , причем вместо нормирования xn применять приведение вида xn = xn – (xn, g1) / (e1, g1) \* e1.

5) Получить собственный вектор e2 = xn / ||xn||

3. Вычисление λ ближайшего к λ0:

1) Взять xn = x0, λn = λ0

2) Далее выполнять итерационно:

2.1) Решить систему уравнений (исп. программа лр 1. решения методом Гаусса): 

2.2) Получить:



Процесс выполняется пока |λn - λn+1| >= ε, при увеличении расстояний между приближениями λ (если не задана критическая величина итераций) либо при достижении числа итераций критического значения процесс завершается с ошибкой сходимости.

4. Вычислении λmin:

1) Получить матрицу B = A + cE, где с – достаточно большое по модулю отрицательное число (от выбора c зависит то, минимальное или максимальное λ будет получено, c задано оператором).

2) Выполнить алгоритм п. 1. для матрицы B.

3) Искомое λ = λB – c, собственный вектор B равен вектору λ.

Тесты:

Исходные данные в файле имеют вид:

n e λ0­

a11 … a1n

…

an1 … ann

x1 … xn

Где n – размерность матрицы, e – точность, λ0 – заданное значения для поиска ближайшего λ, A – матрица, x – начальный вектор приближения.

1) Тестовый пример.

Исходные данные:

3 0.001 -1

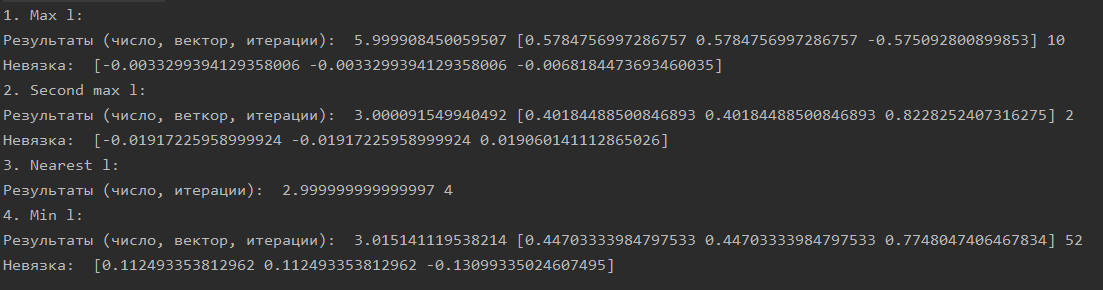
4 1 -1

1 4 -1

-1 -1 4

1 1 1

Результат:



Комментарий: результаты соответствуют действительности, приемлемая величина невязки.

2) п.3 задания

Исходные данные:

4 0.001 -0.1

1 1 1 1

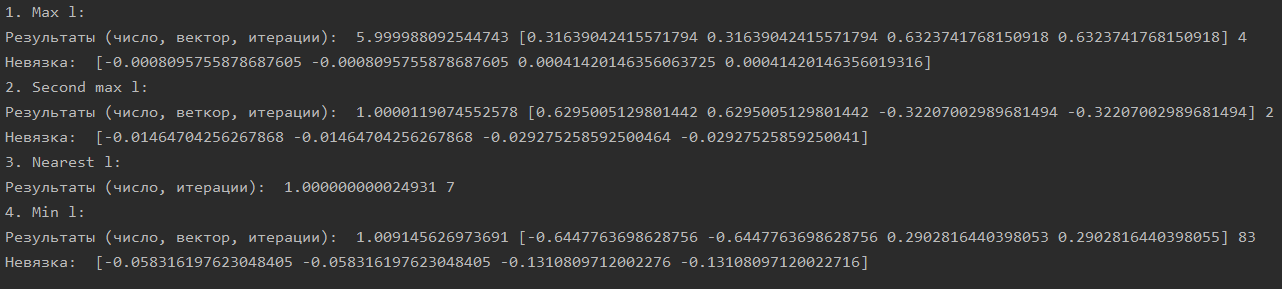
1 1 1 1

1 1 4 1

1 1 1 4

1 1 1 1

Результат:



Исходные данные:

4 0.001 -0.1

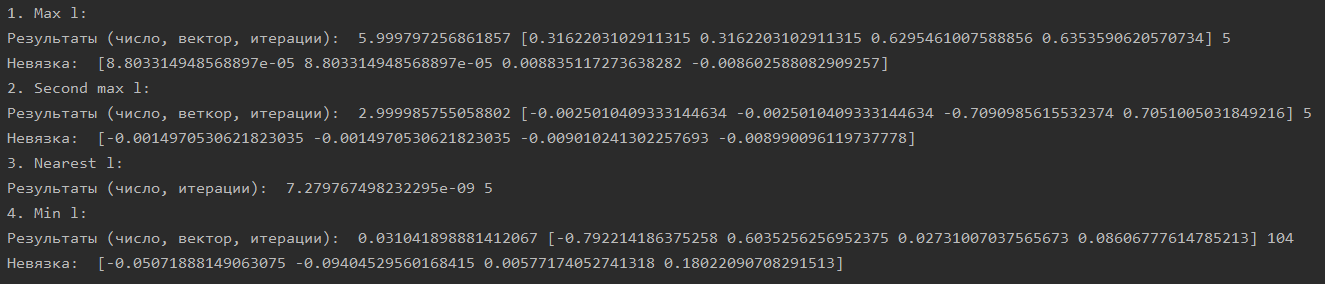
1 1 1 1

1 1 1 1

1 1 4 1

1 1 1 4

1 2 3 4



Комментарий: видно, что в данном примере от выбора начального вектора зависит корректность результата поиска λ2, т.е. скорее всего в процессе вычисления при x0 = (1,1,1,1) слагаемое в разложении по базису, соответствующее вектору e2 обращается в 0, поэтому для получения корректного результата необходимо попробовать другое начальное приближение, в данном случае 1,2,3,4.

3) п 4. задания:

Исходные данные:

3 0.001 4

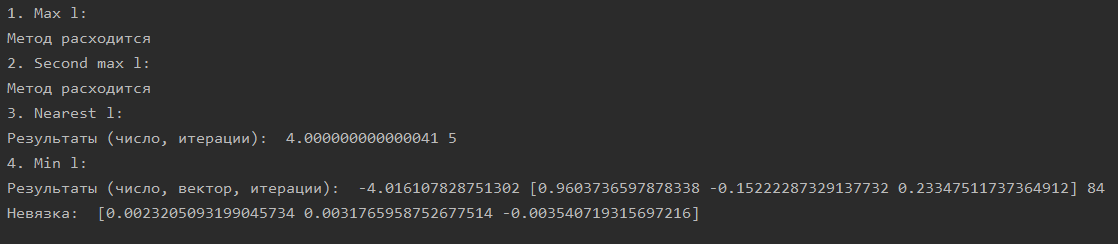
-5.7 -61.1 -32.9

0.8 11.9 7.1

-1.1 -11.8 -7.2

1 1 1

Результат:



Комментарий: видно, что п.1 и п.2 алгоритма расходятся, причиной тому могут служить нарушение условия , т.к. видно, что п.3 и п.4 отработали корректно.

4) п.5 задания:

Исходные данные:

3 0.0001 5

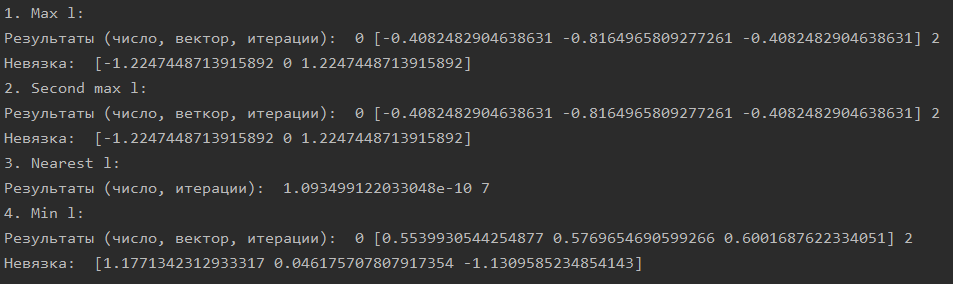
0 1 1

-1 0 1

-1 -1 0

1 1 1

Результат:



Комментарий: во всех методах была получена очень большая невязка, при этом фактически число итераций близко к 0 (т.к. во всех случаях итерационный процесс начинается после двух обязательных итераций, кроме поиска ближайшего собственного числа). Отсюда следует соображение проверить обусловленность данной матрицы. Для данной матрицы данное число бесконечно велико (detA = 0 => обратная матрица не существует (бесконечное множество решений)), я считаю, причина такого результата в плохой обусловленности исходной матрицы.

Текст программы (яп Go):

package main

import (

"errors"

"fmt"

"math"

"myReader"

"os"

)

func main(){

n, A, e, l0, x0, err := myReader.ReadM4("5.txt")

if err != nil{

panic(err)

}

fmt.Println("1. Max l:")

l1, ev1, count1, err := eignvalue1(n, A, e, x0, 1, 999999)

if err != nil {

fmt.Println(err)

} else {

fmt.Println("Результаты (число, вектор, итерации): ", l1, ev1, count1)

fmt.Println("Невязка: ", r(n, A, ev1, l1))

}

fmt.Println("2. Second max l:")

l2, ev2, count2, err := eignvalue2(n, A, e, x0, ev1, 1, 1, 10000)

if err != nil {

fmt.Println(err)

} else {

fmt.Println("Результаты (число, веткор, итерации): ", l2, ev2, count2)

fmt.Println("Невязка: ", r(n, A, ev2, l2))

}

fmt.Println("3. Nearest l:")

nl, count3, err := nearestEignvalue(n, A, e, x0, l0, 10000)

if err != nil {

fmt.Println(err)

} else {

fmt.Println("Результаты (число, итерации): ", nl, count3)

}

fmt.Println("4. Min l:")

minl, eMin, countMin, err := minEignvalue(n, A, e, x0, 10, -100, 10000)

if err != nil {

fmt.Println(err)

} else {

fmt.Println("Результаты (число, вектор, итерации): ", minl, eMin, countMin)

fmt.Println("Невязка: ", r(n, A, eMin, minl))

}

}

//вспомогательная процедура невязки

func r(n int, A [][]float64, x []float64, l float64)(res []float64){

A\_x := Ax(n, A, x)

for i := 0; i < n; i++{

res = append(res, A\_x[i] - l \* x[i])

}

return res

}

//попробуем организовать поиск минимального собственного числа

//вход: размерность матрицы, матрица, точность, начальный вектор, период нормализации, c, критическое число итераций

//выход:

func minEignvalue(n int, A [][]float64, e float64, x0 []float64, np int, c float64, crucialCount int) (l float64, ev []float64, count int, err error){

B := make([][]float64, n)

E := func(i int, j int) float64 {if i == j {return 1} else {return 0} }

for i := 0; i < n; i++{ //получить матрицу B

B[i] = make([]float64, n)

for j := 0; j < n; j++{

B[i][j] = A[i][j] + c \* E(i, j)

}

}

lb, eb, countB, err := eignvalue1(n, B, e, x0, 1, crucialCount)

if err != nil {

return -1, nil, -1, err

}

l = lb - c

return l, eb, countB, nil

}

//процедура нахождения собственного числа, ближайшего к данному

//вход: размерность матрицы, матрица, точность, начальное приближение, l0, crucialCount

//выход:

func nearestEignvalue(n int, A [][]float64, e float64, x0 []float64, l0 float64, crucialCount int)(l float64, count int, err error){

isFirst := true;

var ln1 float64

ln := l0

xn := make([]float64, n)

copy(xn, x0) //скопировать начальное приближение

var xn1 []float64

count = 0

delta := e + 1

var oldDelta float64

for isFirst || math.Abs(ln - ln1) >= e{

if (delta >= oldDelta && crucialCount == 0 && oldDelta != e + 1) || (count > crucialCount) {

return -1, -1, errors.New("Метод расходится")

}

if isFirst {

isFirst = false

} else {

ln = ln1

}

subA := make([][]float64, n)

for i := 0; i < n; i++{

subA[i] = make([]float64, n) //получить матрицу системы

for j := 0; j < n; j++ {

if i == j {

subA[i][j] = A[i][j] - ln

} else {

subA[i][j] = A[i][j]

}

}

subA[i] = append(subA[i], xn[i]) //добавить столбец b для получения системы

}

xn1, \_, \_, \_, \_, err = systemProcessing(subA, n, true) //попытаться решить систкму

if err != nil {

return -1, -1, err

}

ln1 = ln + scalar(n, xn, xn) / scalar(n, xn1, xn)

norm := math.Sqrt(scalar(n, xn1, xn1))

for i := 0; i < n; i++{ //обновить предшествующий вектор с нормированием

xn[i] = xn1[i] / norm

}

count++

oldDelta = delta

delta = math.Abs(ln - ln1)

}

return ln1, count, nil

}

//процедура нахождения второго по модулю собственного числа матрицы

//вход: размерность матрицы, матрица, точность, x0, собственный вектор max L, период обновления, периодн нормирования, crucial count

//выход:

func eignvalue2(n int, A [][]float64, e float64, x0 []float64, e1 []float64, up int, np int, crucialCount int)(l float64, ev []float64, count int, err error){

At := make([][]float64, n)

for i := 0; i < n; i++{

At[i] = make([]float64, n)

for j := 0; j < n; j++{

At[i][j] = A[j][i] //получить транспонированную матрицу

}

}

\_, g1, \_, err := eignvalue1(n, At, e, x0, np, crucialCount)

if err != nil{

return -1, nil, -1, err

}

y0 := make([]float64, n) //начальное приближение для запуска алгоритма

scalarConst := scalar(n, x0, g1) / scalar(n, e1, g1)

for i := 0; i < n; i++{

y0[i] = x0[i] - scalarConst \* e1[i]

}

//повторить аналогичный алгоритм

xn := make([]float64, n)

copy(xn, y0) //скопировать начальное приближение

var xn1 []float64

var ln, ln1 float64 //приближения собственных чисел

//провести первую итерацию

xn1 = Ax(n, A, xn)

ln = scalar(n, xn, xn1) / scalar(n, xn, xn)

//провести вторую итерацию

xn = xn1

xn1 = Ax(n, A, xn)

ln1 = scalar(n, xn, xn1) / scalar(n, xn, xn)

count = 2 //счетчик итерации

xn = xn1 //обновить предыдущий вектор

delta := math.Abs(ln1 - ln)

oldDelta := delta + 1

for delta >= e { //основной цикл итерации

if (delta >= oldDelta && crucialCount == 0) || (count > crucialCount) {

return -1, nil, -1, errors.New("Метод расходится")

}

xn1 = Ax(n, A, xn) //получить следующий вектор

ln = ln1 //сохранить старое l

ln1 = scalar(n, xn, xn1) / scalar(n, xn, xn) //получить новое l

count++ //увлечить число итераций

if count%up == 0 { //если достигли периода обновления

scalarConst = scalar(n, xn1, g1) / scalar(n, e1, g1)

for i := 0; i < n; i++ {

xn1[i] -= scalarConst \* e1[i] //обновить вектор

}

}

xn = xn1 //обновить предыдущий вектор

oldDelta = delta

delta = math.Abs(ln1 - ln)

}

norm := math.Sqrt(scalar(n, xn, xn))

for i := 0; i < n; i++{

ev = append(ev, xn[i] / norm) //нормализировать вектор

}

return ln1, ev, count, nil

}

//процедура нахождения маскимального по модулю собственного числа матрицы

//вход: размерность матрицы, матрица, точность, x0, период нормирования, критическое число итераций

//выход:

func eignvalue1(n int, A [][]float64, e float64, x0 []float64, np int, crucialCount int) (l float64, ev []float64, count int, err error){

xn := make([]float64, n)

copy(xn, x0) //скопировать начальное приближение

var xn1 []float64

var ln, ln1 float64 //приближения собственных чисел

//провести первую итерацию

xn1 = Ax(n, A, xn)

ln = scalar(n, xn, xn1) / scalar(n, xn, xn)

//провести вторую итерацию

xn = xn1

xn1 = Ax(n, A, xn)

ln1 = scalar(n, xn, xn1) / scalar(n, xn, xn)

count = 2 //счетчик итерации

xn = xn1 //обновить предыдущий вектор

delta := math.Abs(ln1 - ln)

oldDelta := delta + 1

for delta >= e{ //основной цикл итерации

if (delta >= oldDelta && crucialCount == 0) || (crucialCount != 0 && count > crucialCount){

return -1, nil, -1, errors.New("Метод расходится")

}

xn1 = Ax(n, A, xn) //получить следующий вектор

ln = ln1 //сохранить старое l

ln1 = scalar(n, xn, xn1) / scalar(n, xn, xn) //получить новое l

count++ //увлечить число итераций

if count%np == 0 { //если достигли периода нормализации

norm := math.Sqrt(scalar(n, xn1, xn1))

for i := 0; i < n; i++ {

xn1[i] /= norm //нормализировать вектор

}

}

xn = xn1 //обновить предыдущий вектор

oldDelta = delta

delta = math.Abs(ln1 - ln)

}

norm := math.Sqrt(scalar(n, xn, xn))

for i := 0; i < n; i++{

ev = append(ev, xn[i] / norm) //нормализировать вектор

}

return ln1, ev, count, nil

}

//вспомогательная процедура скалярного произведения

func scalar(n int, x1 []float64, x2 []float64) float64{

var res float64 = 0

for i := 0; i < n; i++{

res += x1[i] \* x2[i]

}

return res

}

//всопомогательная процедура оператора

func Ax(n int, A [][]float64, x[]float64)(res []float64){

for i := 0; i < n; i++{

var sum float64 = 0

for j := 0; j < n; j++{

sum += A[i][j] \* x[j]

}

res = append(res, sum)

}

return res

}

//ниже копипаст из lr1 для решения системы методом гаусса

//

//

//

//

//

//процедура обработки системы: получает решение системы, велечину невязки, определитель матрицы системы и обратную матрицу системы

//входные параметры: исходная матрица уравнения Ab, кол-во неизвестных, признак использования главного элемента в алгоритме приведения к треугольному виду

//результат: решение системы, вектор невязки для решения системы, определитель, обратная матрица, невязка для обратной матрицы, информация об ошибке

func systemProcessing(A [][]float64, n int, withMajor bool) ([]float64, []float64, float64, [][]float64, [][]float64, error){

//скопировать матрицу A и вектор b для вычисления невязки обратной матрицы ниже

AbE := A //сформируем матрицу вида A|b|E - расширенную марице уравнения

A = make([][]float64, n)

for i := 0; i < n; i++{

A[i] = make([]float64, n)

copy(A[i], AbE[i][:n])

}

b := make([]float64, n)

for i := 0; i < n; i++{

b[i] = AbE[i][n]

}

for i := 0; i < n; i++{ //цикл по строкам

for j := n + 1; j <= n \* 2; j++{ //цикл по столбцам

if i == j - (n + 1) { //если элемент на главной диагонали подматрицы E

AbE[i] = append(AbE[i], 1) //то записать 1

} else {

AbE[i] = append(AbE[i], 0) //иначе 0

}

}

}

AbE, swapStringNumb, err := reductionToTriangular(AbE, n, withMajor) // привести всё это дело к треугольному виду

if err != nil { //проверка на возникновение ошибок

return nil, nil, 0, nil, nil, err

}

solution, err := solve(AbE, n, n)//далее получить решение для системы уравнений

if err != nil {

return nil, nil, 0, nil, nil, err

}

//определить вектор невязки для полученного решения

solutionDelta := make([]float64, n) //вектор невязки

for i:= 0; i < n; i++{

var sum float64 = 0

for j := 0; j < n; j++ {

sum += A[i][j] \* solution[j]

}

solutionDelta[i] = b[i] - sum

}

//далее получить решения являющиеся столбцами обратной матрицы и сформировать обратную матрицу

A\_ := make([][]float64, n) //обратная матрица

for i := n + 1; i < n + 1 + n; i++ { // цикл по всем столбцам подматрицы E

column, err := solve(AbE, n, i) //решить систему и получить очередной столбец обратной матрицы

if err != nil {

return nil, nil, 0, nil, nil, err

}

for j := 0; j < n; j++{

A\_[j] = append(A\_[j], column[j]) //заполнить столбец обратной матрицы

}

}

//определить невязку для обратной матрицы

R := make([][]float64, n) //произведение прямой и обратной матрицы A, затем вектор невязки

for i := 0; i < n; i++{ //цикл по строкам

for j := 0; j < n; j++ { //цикл по столбцам

R[i] = append(R[i], 0) //добавить элемент AA\_[i][j]

for k := 0; k < n; k++ { //цикл умножения iой строки A на jый столбец A\_

R[i][j] += A[i][k] \* A\_[k][j]

}

}

}

//преобразовать AA\_ к вектору невязки (AA\_ = E - AA\_)

for i := 0; i < n; i++ {

for j := 0; j < n; j++{

if i == j { //если элемент на главной диагонали

R[i][j] = 1 - R[i][j]

} else {

R[i][j] = 0 - R[i][j]

}

}

}

detA := det(AbE, n, swapStringNumb) // вычислить определитель матрицы A

return solution, solutionDelta, detA, A\_, R, nil

}

//процедура получения корней системы линейных уравнений по треугольной матрице системы

//входные параметры: расширенная матрица системы, кол-во неизвестных, индекс столбца вектора правой части системы

func solve(AbE [][]float64, n int, bIndex int) ([]float64, error) {

res := make([]float64, n) // результат

for i := n - 1; i >= 0; i-- { // цикл подъема по матрице вверх для нахождения корней

var sum float64 = 0.0; // получить отнимаемую сумму

for j := i + 1; j < n; j++ {

sum += AbE[i][j] \* res[j]

}

res[i] = (AbE[i][bIndex] - sum) / AbE[i][i]

if math.IsNaN(res[i]) || math.IsInf(res[i], 1) || math.IsInf(res[i], -1) { // проверка на выход за пределы множества машинных чисел

return nil, errors.New("Одно из чисел за пределами множества машинных чисел, получить решение невозможно. Попробуйте использовать алгоритм с выбором главного элемента.")

}

}

return res, nil

}

//процедура нахождения определителя исходной матрицы A по приведенной треугольной матрице Ab

//входные параметры: приведенная к треугольному виду матрица Ab, число неизвестных, число перестановок строк в процессе приведения

//результат: определитель матрицы

func det(Ab [][]float64, n int, swapStringNumb int) float64 {

var det float64 = 1

for i := 0; i < n; i++ {

det \*= Ab[i][i]

}

if swapStringNumb % 2 == 1{ //если необходимо изменить знак

det \*= -1

}

return det

}

//процедура приведения к треугольному виду матрицы Ab

//треугольный вид имеет подматрица A

//входные параметры: матрица уравнения Ab, кол-во неизвестных n, признак использования главного элемента в алгоритме

//результат: матрица, приведённая к указанному виду, число перестановок строк в процессе алгоритма, сообщение об ошибках

func reductionToTriangular (Ab [][]float64, n int, withMajor bool) ([][]float64, int, error){

swapStringNumb := 0 // число перестановок строк в процессе алгоритма

for i := 0; i < n; i++ { // цикл по столбцам матрицы A

if withMajor { // если необходимо ставить на диагональ главный элемент

major := i // индекс первичного значения

for j := i + 1; j < n; j++ {

if (math.Abs(Ab[major][i]) == 0.0) || ((math.Abs(Ab[j][i]) < math.Abs(Ab[major][i])) && math.Abs(Ab[j][i]) != 0.0) { // сравнить по модулю

major = j

}

}

if math.Abs(Ab[major][i]) == 0.0 { // если матрица вырожденая

return nil, swapStringNumb, errors.New("Вырожденая матрица")

}

if i != major { //если выполняется перестановка

Ab = swapString(Ab, i, major) // установить главный элемент если требуется

swapStringNumb++ //увеличить число перестановок

}

}

for j := i + 1; j < n; j++ { // цикл по строкам для столбца i

Cji := Ab[j][i] / Ab[i][i] // строка i умножается на Cji и вычитается из строки j

if math.IsNaN(Cji) || math.IsInf(Cji, 1) || math.IsInf(Cji, -1) { // проверка на выход за пределы множества машинных чисел

return nil, swapStringNumb, errors.New("Одно из чисел за пределами множества машинных чисел, получить решение невозможно. Попробуйте использовать алгоритм с выбором главного элемента.")

}

for k := i; k <= n \* 2; k++ { // цикл вычитания строки i из строки j

Ab[j][k] -= Ab[i][k] \* Cji // вычесть очередной элемент строки

}

}

}

return Ab, swapStringNumb, nil

}

//вспомогательная процедура перестановки строк i и j в матрице m

func swapString(m [][]float64, i int, j int) [][]float64{

buf := m[i]

m[i] = m[j]

m[j] = buf

return m

}

//процедура чтения исодных данных:

//считывает расширенную матрицу уравнения из заданного файла

func readSourceData(fileName string) ([][]float64, int, error) {

file, err := os.Open(fileName) // открыть файл с данными

defer file.Close() // закрыть по завершению

if err != nil { // если ошибка открытия, то завершения с ошибкой

return nil, -1, err

}

var n int // кол-во неизвестных

\_, err = fmt.Fscanf(file, "%d\n", &n) //считать кол-во неихвестныъ

if err != nil { // если ошибка, то завершение

return nil, -1, err

}

var Ab [][]float64 = make([][]float64, n) //матрица n x (n + 1) - расширенная вида A|b

//цикл чтения матрицы из файла

for i := 0; i < n; i++{

Ab[i] = make([]float64, n + 1) // выделить память под строку матрицы

for j := 0; j <= n; j++ { // считтаь элементы строки

\_, err = fmt.Fscanf(file, "%g", &Ab[i][j])

if err != nil {

return nil, -1, err

}

}

fmt.Fscanf(file, "\n") //пропустить newline

}

return Ab, n, nil // вернуть результат

}