|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатики и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

**Лабораторная работа № 1**

«Решение СЛАУ с трехдиагональной матрицей методом прогонки»

по курсу «Численные методы»

Выполнила:

студент группы ИУ9-61Б

Яровикова Анастасия

Проверила:

Домрачева А. Б.

Москва, 2023

1. **Цель**

Целью данной работы является изучение накопления погрешности в решении системы линейных алгебраических уравнений (СЛАУ) с трехдиагональной матрицей методом прогонки.

1. **Постановка задачи**

**Дано:** , где , – трехдиагональная матрица.

**Найти:** Решение СЛАУ (вектор ) с помощью метода прогонки при исходных .

Для достижения цели работы были поставлены следующие задачи:

* ознакомление с теорией метода прогонки;
* реализация алгоритма поиска решения СЛАУ методом прогонки на языке программирования Go;
* нахождение погрешности решения и оценка точности результата.

1. **Основные теоретические сведения**

Метод прогонки является одним из способов решения систем линейных алгебраических уравнений вида , где - трехдиагональная матрица. Данный метод представляет собой вариацию метода последовательного исключения неизвестных системы.

**Описание алгоритма:**

Пусть – массив элементов под главной диагональю, – массив элементов главной диагонали, – массив элементов над главной диагональю.

Данная матрица задает следующую СЛАУ:

Из данной системы получаем:

Произведя замену , получаем .

Подставляем полученные значения во второе уравнение системы:

Получаем:

Произведя замену , получаем .

Продолжая аналогичные рассуждения, получаем формулы для нахождения :

где

вычисляется следующим образом:

Вычисление называется прямым ходом метода прогонки, а вычисление – обратным ходом метода прогонки.

**Достаточные условия метода прогонки:**

**Оценка погрешности для решения СЛАУ при отсутствии точного решения:**

Необходимо найти решение СЛАУ методом прогонкой – вектор . Для оценки погрешности вычисления вычисляем:

где – искомый вектор ошибок.

Тогда

Точное решение .

1. **Реализация**

Листинг 1. Метод прогонки для решения СЛАУ с трехдиагональной матрицей

package main

import (

"bufio"

"fmt"

"log"

"math"

"os"

"strconv"

"strings"

"gonum.org/v1/gonum/mat"

)

var N int

func parseArrs(line string, N int) ([]float64, error) {

arr := make([]float64, 0, N)

//string -> []string

strs := strings.Split(line, " ")

for \_, s := range strs {

num, err := strconv.ParseFloat(s, 64)

if err != nil {

return nil, err

} else {

arr = append(arr, num)

}

}

return arr, nil

}

func solution(a, b, c, d []float64) ([]float64) {

x := make([]float64 , N)

//forward

var alpha, beta []float64

alpha = append(alpha, - c[0] / b[0])

beta = append(beta, d[0] / b[0])

var y float64

for i := 1; i < N; i++ {

if i != N-1 {

y = a[i-1] \* alpha[i-1] + b[i]

alpha = append(alpha, -c[i] / y)

beta = append(beta, (d[i]-a[i-1] \* beta[i-1]) / y)

} else {

y = a[N-2] \* alpha[N-2] + b[N-1]

beta = append(beta, (d[N-1] - a[N-2] \* beta[N-2]) / y)

}

}

//backwards

for i := N-1; i >= 0; i-- {

if i == N-1 {

x[N-1] = beta[N-1]

} else {

x[i] = alpha[i] \* x[i+1] + beta[i]

}

}

return x

}

func makeMatrix(c, b, a []float64) [][]float64 {

m := make([][]float64, N)

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

m[i] = make([]float64, N)

}

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

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

if i == j {

m[i][j] = b[i]

if i != N-1 {

m[i][i+1] = c[i]

m[i+1][i] = a[i]

}

}

}

}

return m

}

func mulMatVec(matrix [][]float64, x []float64) []float64 {

d := make([]float64, N)

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

var s float64 = 0

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

s += matrix[i][j] \* x[j]

}

d[i] = s

}

return d

}

func main() {

// tets<i>.txt = dimension; matrix: main diagonal, above diagonal , under diagonal; vector D

file, err := os.Open("tests/test1.txt")

if err != nil {

log.Fatal(err.Error())

}

defer file.Close()

var arrs []string

scanner := bufio.NewScanner(file)

for scanner.Scan() {

N, \_ = strconv.Atoi(scanner.Text())

break

}

for scanner.Scan() {

arrs = append(arrs, scanner.Text())

}

if err := scanner.Err(); err != nil {

log.Fatal(err)

}

b, err := parseArrs(arrs[0], N)

if err != nil {

log.Fatal(err.Error())

}

c, err := parseArrs(arrs[1], N-1)

if err != nil {

log.Fatal(err.Error())

}

a, err := parseArrs(arrs[2], N-1)

if err != nil {

log.Fatal(err.Error())

}

d, err := parseArrs(arrs[3], N)

if err != nil {

log.Fatal(err.Error())

}

// a priori х = 1 1 1 1

xf := []float64{1, 1, 1, 1}

// a posteriori:

x := solution(a, b, c, d)

m := makeMatrix(c, b, a)

matrix := make([]float64, 0)

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

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

matrix = append(matrix, m[i][j])

}

}

mm := mat.NewDense(N, N, matrix)

f := mat.Formatted(mm, mat.Prefix(" "), mat.Squeeze())

fmt.Printf("A = %.16f\n\n", f)

fmt.Print("X: ")

for \_, n := range xf {

fmt.Print(fmt.Sprintf("%.16f", n), " ")

}

fmt.Println()

fmt.Print("d: ")

for \_, n := range d {

fmt.Print(fmt.Sprintf("%.16f", n), " ")

}

fmt.Println()

fmt.Print("\nX\*: ")

for \_, n := range x {

fmt.Print(fmt.Sprintf("%.16f", n), " ")

}

fmt.Println()

d\_new := mulMatVec(m, x)

fmt.Print("d\*: ")

for \_, n := range d\_new {

fmt.Print(fmt.Sprintf("%.16f", n), " ")

}

fmt.Println()

// the diference between old vector d and new vector d

var dif float64

r := make([]float64, 0)

fmt.Print("\nvector r = |d - d\*|: ")

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

dif = math.Abs(d[i] - d\_new[i])

r = append(r, dif)

fmt.Print(fmt.Sprintf("%.16f", dif), " ")

}

fmt.Println()

// the difference between a priori X and a posteriori X:

// e = A^(-1) \* r

var inv mat.Dense

err = inv.Inverse(mm)

if err != nil {

log.Fatalf("matrix is not invertible: %v", err)

}

f = mat.Formatted(&inv, mat.Prefix(" "), mat.Squeeze())

fmt.Printf("\nA^(-1) = %.16f\n\n", f)

inverted := inv.RawMatrix().Data

inv\_m := make([][]float64, N)

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

inv\_m[i] = make([]float64, N)

}

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

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

inv\_m[i][j] = inverted[i \* N + j]

}

}

e := mulMatVec(inv\_m, r)

fmt.Print("vector e1 = |x - x\*|: ")

for i, n := range xf {

fmt.Print(fmt.Sprintf("%.16f", math.Abs(n - x[i])), " ")

}

fmt.Println()

fmt.Print("vector e2 = A^(-1) \* r: ")

for \_, n := range e {

fmt.Print(fmt.Sprintf("%.16f", n), " ")

}

fmt.Println()

}

1. **Тестирование**

Для тестирования полученной программы в качестве трехдиагональной матрицы была выбрана следующая матрица:

В качестве вектора :

Таким образом, СЛАУ имеет вид:

В результате работы программы (см. листинг 1) получены значения:

Как видно выше, вектор погрешности не является нулевым, что связано с использованием типа данных float64 с точностью в 16 знаков после запятой.

1. **Вывод**

В ходе выполнения лабораторной работы был изучен метод решения СЛАУ с трехдиагональной матрицей – метод прогонки. Алгоритм был реализован на языке программирования Go.

Для метода прогонки можно отметить эффективность, обусловленную хранением лишь части данных матрицы (ненулевые диагональные элементы). У данного метода отсутствует методологическая погрешность, однако имеет место вычислительная. В представленной реализации присутствует существенная погрешность, т.е. полученное решение значительно отличается от априорного решения данной системы – единичного вектора. Это связано с особенностью использования чисел с плавающей точкой, а также усечением разрядной сетки результатов вычисления, что и приводит к вычислительной погрешности.