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.

2. Постановка задачи

Дано: $A\bar{x} = \bar{d}$, где $A \in \mathbb{R}^{n \times n}$, $\bar{x} = \bar{d} \in \mathbb{R}^n$, A – трехдиагональная матрица.

Найти: Решение СЛАУ (вектор $\bar{x}$) с помощью метода прогонки при исходных $A\ и\ \bar{d}$.

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.

3. Основные теоретические сведения

Метод прогонки является одним из способов решения систем линейных алгебраических уравнений вида $A\bar{x} = \bar{d}$, где A - трехдиагональная матрица. Данный метод представляет собой вариацию метода последовательного исключения неизвестных системы.

Описание алгоритма:

Пусть a – массив элементов под главной диагональю, b – массив элементов главной диагонали, c – массив элементов над главной диагональю.

$$\begin{pmatrix} b_1 & c_1 & 0 & \cdots & \cdots & 0 \\ a_1 & b_2 & c_2 & \cdots & \cdots & 0 \\ 0 & a_2 & b_3 & c_3 & \cdots & 0 \\ \vdots & \cdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & a_{n-2} & b_{n-1} & \vdots \\ 0 & \cdots & \cdots & \cdots & a_{n-1} & b_n \end{pmatrix}$$

Данная матрица задает следующую СЛАУ:

$$\begin{cases} b_1x_1 + c_1x_2 = d_1 \\ a_1x_1 + b_2x_2 + c_2x_3 = d_2 \\ \dots \\ a_{n-1}x_{n-1} + b_nx_n = d_n \end{cases}$$

Из данной системы получаем:

$$x_1 = \frac{d_1 - c_1x_2}{b_1} = -\frac{c_1}{b_1}x_2 + \frac{d_1}{b_1}, \quad b_1 \neq 0$$

Произведя замену $\alpha_1 = -\frac{c_1}{b_1}, \beta_1 = \frac{d_1}{b_1}$, получаем $x_1 = \alpha_1x_2 + \beta_1$.

Подставляем полученные значения во второе уравнение системы:

$$a_1(\alpha_1x_2 + \beta_1) + b_2x_2 + c_2x_3 = d_2$$

Получаем:

$$x_2 = -\frac{c_2}{a_1\alpha_1 + b_2}x_3 + \frac{d_2 - a_1\beta_1}{a_1\alpha_1 + b_2}$$

Произведя замену $\alpha_2 = -\frac{c_2}{a_1\alpha_1+b_2}, \beta_2 = \frac{d_2-a_1\beta_1}{a_1\alpha_1+b_2}$, получаем $x_2 = \alpha_2x_3 + \beta_2$.

Продолжая аналогичные рассуждения, получаем формулы для нахождения α_i и β_i:

$$x_i = -\frac{c_i}{a_{i-1}\alpha_{i-1} + b_i}x_{i+1} + \frac{d_i - a_{i-1}\beta_{i-1}}{a_{i-1}\alpha_{i-1} + b_i}$$

$$\alpha_i = -\frac{c_i}{a_{i-1}\alpha_{i-1} + b_i}, \beta_2 = \frac{d_i - a_{i-1}\beta_{i-1}}{a_{i-1}\alpha_{i-1} + b_i}$$

где $i = \overline{2, n-1}$.

x_i вычисляется следующим образом:

$$x_i = \alpha_ix_{i+1} + \beta_i, \qquad i = \overline{n-1,1}$$

$$x_n = \frac{d_n - a_{n-1}\beta_{n-1}}{a_{n-1}\alpha_{n-1} + b_n} = \beta_n.$$

Вычисление α_i и β_i называется прямым ходом метода прогонки, а вычисление x_i – обратным ходом метода прогонки.

Достаточные условия метода прогонки:

1. $|b_i| \geq |a_{i-1}| + |c_i|, \quad i = \overline{2, n}$
2. $\left|\frac{a_{i-1}}{b_i}\right| \leq 1, \left|\frac{c_i}{b_i}\right| \leq 1$

Оценка погрешности для решения СЛАУ при отсутствии точного решения:

Необходимо найти решение СЛАУ методом прогонкой – вектор $\overline{x^*}$. Для оценки погрешности вычисления вычисляем:

$$A\overline{x^*} = \overline{d^*}$$

$$A(\bar{x} - \overline{x^*}) = (\bar{d} - \overline{d^*})$$

$$\bar{r} = (\bar{d} - \overline{d^*})$$

$$\bar{e} = (\bar{x} - \overline{x^*})$$

$$A\bar{e} = \bar{r},$$

где $\bar{e}$ – искомый вектор ошибок.

Тогда $\boxed{\bar{e} = A^{-1}\bar{r}}$

Точное решение $\bar{x} = \overline{x^*} - \bar{e}$.

4. 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 x = 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()
```

```
}
```

5. Тестирование

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

$$A = \begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

В качестве вектора $\bar{d}$:

$$\bar{d} = \begin{pmatrix} 5 \\ 6 \\ 6 \\ 5 \end{pmatrix}$$

Таким образом, СЛАУ имеет вид:

$$A\bar{x} = \bar{d} = \begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 5 \\ 6 \\ 6 \\ 5 \end{pmatrix}$$

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

$$\overline{x^*} = \begin{pmatrix} 0.9952153110047848 \\ 1.0191387559808611 \\ 0.9282296650717703 \\ 1.2679425837320575 \end{pmatrix}, \bar{e} = A^{-1}\bar{r} = \begin{pmatrix} -0.0000000000000001 \\ 0.0000000000000003 \\ -0.0000000000000001 \\ 0.0000000000000000 \end{pmatrix}$$

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

6. 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.