Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
Московский государственный технический университет имени Н.Э. Баумана

Лабораторная работа №1
по курсу «Численные методы»
«Решение СЛАУ. Метод прогонки»

Выполнил:
студент группы ИУ9-62Б
Егоров Алексей

Проверила:
Домрачева А.Б.

Москва, 2022

1. Цель

Анализ метода прогонки и решение СЛАУ с трёхдиагональной матрицей с помощью данного метода. Сравнение решения СЛАУ, полученного методом прогонки, с решением СЛАУ, полученного с помощью метода Гаусса.

2. Постановка задачи

Дано: $A\overline{x} = \overline{d}$, где $A \in \mathbb{R}^{n\times n}$ и $\overline{x}, \overline{d} \in \mathbb{R}^n$, A - трехдиагональная матрица

Найти: Решение СЛАУ с помощью метода прогонки, т.е $\overline{x}$ - ? при известных $A, \overline{d}$

Тестовый пример:

В качестве трехдиагональной матрицы A возьмем:

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

А в качестве вектора $\overline{d}$:

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

При этом уравнение примет вид:

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

3. Теоретические сведения

3.1 Метод прогонки

Метод прогонки используется для решения систем линейных уравнений вида $A\overline{x} = \overline{d}$, где A - трёхдиагональная матрица. Представляет собой вариант метода последовательного исключения неизвестных.

Описание алгоритма:

Пусть массив a - элементы матрицы А под диагональю, b - на диагонали, c – над диагональю.

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

Соответствующая СЛАУ:

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

x_i вычисляется следующим образом:

$$x_i = \alpha_i x_{i+1} + \beta_i, i = \overline{n-1, 1}$$

$$x_n = \frac{d_n - a_{n-1}b_{n-1}}{a_{n-1}\alpha_{n-1} + b_n}$$

α_i, β_i вычисляются следующим образом:

$$\alpha_i = -\frac{c_i}{a_{i-1}\alpha_{i-1} + b_i}, i = \overline{2, n-1}$$

$$\beta_i = \frac{d_i - a_{i-1}\beta_{i-1}}{a_{i-1}\alpha_{i-1} + b_i}, i = \overline{2, n}$$

$$\alpha_1 = \frac{c_1}{b_1}, \beta_1 = \frac{d_1}{b_1}$$

Вычисление α_i, β_i называется прямым ходом метода прогонки. Вычисление x_i - обратным ходом метода прогонки.

Достаточные условия метода:

1. $|b_i| \geq |a_{i-1}| + |c_i|$, $i = \overline{2, n-1}$ - без него возможно решение, но может и не быть.

2. $|d_i| > |c_i|$, $i = \overline{2, n-1}$

3.2 Метод Гаусса

Метод Гаусса заключается в последовательном исключении переменных: с помощью элементарных преобразований система уравнений приводится к равносильной системе треугольного вида, из которой последовательно, начиная с последних (по номеру), находятся все переменные системы.

Описание алгоритма:

Пусть дана система:

$$\begin{cases} a_{1,1}x_1 + ... + a_{1,n}x_n = d_1 \\ ... \\ a_{i,1}x_1 + ... + a_{i,n}x_n = d_i \\ ... \\ a_{n,1}x_1 + ... + a_{n,n}x_n = d_n \end{cases}$$

Приведем матрицу коэффициентов к верхнетреугольному виду: На первом шаге вычитаем из i-ой строки, где $i = 2...n$, первую строку, домноженную на $\frac{a_{i,1}}{a_{1,1}}$. Далее аналогичным образом вычитаем вторую строку, в конечном итоге получаем систему вида:

$$\begin{cases} a''_{1,1}x_1 + ... + a''_{1,n}x_n = d''_1 \\ ... \\ a''_{i,i}x_1 + ... + a''_{i,n}x_n = d''_i \\ ... \\ a''_{n,n}x_n = d''_n \end{cases}$$

Далее начинается обратной ход метода Гаусса. Вычитаем из i-ой строки, где $i = 1...n-1$, последнюю, домноженную на $\frac{a''_{i,n}}{a''_{n,n}}$.

Продолжая этот процесс придём к диагональной матрице:

$$\begin{cases} a'''_{1,1}x_1 = d'''_1 \\ ... \\ a'''_{i,i}x_i = d'''_i \\ ... \\ a'''_{n,n}x_n = d'''_n \end{cases}$$

Разделив i-ую строку на $a'''_{i,i}$ получаем решение.

Оценка погрешности для решения СЛАУ при отсутствии точного решения:

Найти вектор-решение $\overline{x}^*$ с помощью метода прогонки и вектор $\overline{x}$ с помощью метода Гаусса.

При вычислении $\overline{x}^*$ и $\overline{x}$ с плавающей точкой возникает погрешность:

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

4. Практическая реализация:

Листинг 1. Метод прогонки и метод Гаусса для решения СЛАУ с трёхдиагональной матрицей

```
package main

import (
        "bufio"
        "errors"
        "fmt"
        "log"
        "math"
        "os"
        "strconv"
        "strings"
)
```

```go
const SIZE = 4

func readCoeff(line string, name string, size int) ([]float64, error) {
	res := make([]float64, 0, size)

	strNums := strings.Split(line, " ")
	if size != len(strNums) {
		return nil, errors.New("wrong size of " + name)
	}

	for _, sNum := range strNums {
		num, err := strconv.ParseFloat(sNum, 64)
		if err != nil {
			return nil, err
		}
		res = append(res, num)
	}

	return res, nil
}

func direct(bs, as, cs, ds []float64) (alpha, beta []float64) {
	alpha = append(alpha, -cs[0]/bs[0])
	beta = append(beta, ds[0]/bs[0])
	var div float64
	for i := 1; i < SIZE-1; i++ {
		div = as[i-1]*alpha[i-1] + bs[i]
		alpha = append(alpha, -cs[i]/div)
		beta = append(beta, (ds[i]-as[i-1]*beta[i-1])/div)
	}
```

```
	div = as[SIZE-2]*alpha[SIZE-2] + bs[SIZE-1]
	beta = append(beta, (ds[SIZE-1]-as[SIZE-2]*beta[SIZE-2])/div)
	return alpha, beta
}

func reverse(alpha, beta []float64) (xs []float64) {
	xs = make([]float64, SIZE)
	xs[SIZE-1] = beta[SIZE-1]
	for i := SIZE - 2; i >= 0; i-- {
		xs[i] = alpha[i]*xs[i+1] + beta[i]
	}
	return xs
}

func gauss(matrix [][]float64, ds []float64) (xs []float64) {
	xs = make([]float64, SIZE)
	for i := 0; i < SIZE; i++ {
		for j := i + 1; j < SIZE; j++ {
			var k float64 = matrix[j][i] / matrix[i][i]
			for t := i; t < SIZE; t++ {
				matrix[j][t] -= k * matrix[i][t]
			}
			ds[j] -= k * ds[i]
		}
	}
	for i := SIZE - 1; i >= 0; i-- {
		var k float64 = 0
		for j := i + 1; j < SIZE; j++ {
			k += matrix[i][j] * xs[j]
		}
		xs[i] = (ds[i] - k) / matrix[i][i]
```

```
	}
	return xs
}

func multiply(matrix [][]float64, xs []float64) (ds []float64) {
	ds = make([]float64, SIZE)
	for i := 0; i < SIZE; i++ {
		var sum float64 = 0
		for j := 0; j < SIZE; j++ {
			sum += matrix[i][j] * xs[j]
		}
		ds[i] = sum
	}
	return ds
}

func buildMatrix(bs, as, cs []float64) [][]float64 {
	matrix := make([][]float64, SIZE)
	for i := 0; i < SIZE; i++ {
		matrix[i] = make([]float64, SIZE)
	}
	for i := 0; i < SIZE; i++ {
		for j := 0; j < SIZE; j++ {
			if i == j {
				matrix[i][j] = bs[i]
				if i != SIZE-1 {
					matrix[i][i+1] = as[i]
					matrix[i+1][i] = cs[i]
				}
			}
		}
```

```go
	}
	return matrix
}

func main() {
	file, err := os.Open("input.txt")
	if err != nil {
		log.Fatal(err.Error())
	}
	defer file.Close()

	var lines []string
	scanner := bufio.NewScanner(file)
	for scanner.Scan() {
		lines = append(lines, scanner.Text())
	}
	if scanner.Err() != nil {
		log.Fatal(scanner.Err().Error())
	}

	bs, err := readCoeff(lines[0], "bs", SIZE)
	if err != nil {
		log.Fatal(err.Error())
	}
	as, err := readCoeff(lines[1], "as", SIZE-1)
	if err != nil {
		log.Fatal(err.Error())
	}
	cs, err := readCoeff(lines[2], "cs", SIZE-1)
	if err != nil {
		log.Fatal(err.Error())
```

```
	}
	ds, err := readCoeff(lines[3], "ds", SIZE)
	if err != nil {
		log.Fatal(err.Error())
	}

	firstRes := reverse(direct(bs, as, cs, ds))
	fmt.Print("First: ")
	for _, res := range firstRes {
		fmt.Print(fmt.Sprintf("%.16f", res), " ")
	}

	matrix := buildMatrix(bs, as, cs)
	secondRes := gauss(matrix, ds)
	fmt.Print("\nSecond: ")
	for _, res := range secondRes {
		fmt.Print(fmt.Sprintf("%.16f", res), " ")
	}
	fmt.Print("\nDiff: ")
	for i := 0; i < SIZE; i++ {
		fmt.Print(fmt.Sprintf("%.16f", math.Abs(firstRes[i]-secondRes[
	}
	fmt.Println()
	matrix = buildMatrix(bs, as, cs)
	newDs := multiply(matrix, firstRes)
	fmt.Print("NewDs: ")
	for _, res := range newDs {
		fmt.Print(fmt.Sprintf("%.16f", res), " ")
	}
	fmt.Println()
}
```

5. Результат:

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

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

В качестве вектора $\overline{d}$:

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

В результате работы программы (Листинг 1) получаем значения:

$$\overline{\varepsilon} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \overline{x} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad \overline{x}^* = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Как видно выше, не всегда вектор $\overline{\varepsilon}$ может содержать погрешность (нулевой вектор ошибок) в связи с использованием типов данных с двойной точностью. Протестируем программу на измененном векторе $\overline{d}$:

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

В результате работы программы (Листинг 1) получаем значения:

$$\overline{\varepsilon} = \begin{pmatrix} 0.0000000000000001 \\ 0.0000000000000002 \\ 0.0000000000000000 \\ 0.0000000000000000 \end{pmatrix}, \quad \overline{x} = \begin{pmatrix} 0.9952153110047848 \\ 1.0191387559808611 \\ 0.9282296650717703 \\ 1.2679425837320575 \end{pmatrix}, \quad \overline{x}^{*} = \begin{pmatrix} 0.9952153110047847 \\ 1.0191387559808613 \\ 0.9282296650717703 \\ 1.2679425837320575 \end{pmatrix}$$

6. Вывод:

В ходе выполнения лабораторной работы был изучен метод решения СЛАУ с трёхдиагональной матрицей: метод прогонки. Так же были реализованы методы прогонки и Гаусса на языке программирования Go.

Для метода прогонки можно отметить то, что отсутствует методологическая (логическая) погрешность, но присутствует вычислительная погрешность в связи с использованием чисел с плавающей запятой, ведущая к высокому накоплению вычислительной ошибки.