УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная работа №1

Метод Гаусса-Зейделя

Преподаватель: Перл Ольга Вячеславовна

Выполнили: Кульбако Артемий Юрьевич P3212

Санкт-Петербург
2020

Описание метода

Метод Гаусса-Зейделя один из самых распространённых метод решения СЛАУ вида

$$Ax = B,$$

относится к категории итерационных методов. Их преимущество состоит уменьшении погрешности при вычислении вектора неизвестных до заданного значения. Недостатком является достаточно жёсткое условие сходимости итераций:

$$|a_{ii}| \geq \sum_{j \neq i} |aij|, i = 1..n$$

(при этом хотя бы для одного уравнения неравенство должно выполнятся строго). После проверки на сходимость нужно задать начальные приближение вектора x (обычно задают равными 0), и менять их на каждой итерации согласно формуле:

$$x_i^{(k)} = \frac{1}{a_{ii}} (b_k - a_{ij} x_{i+1}^{(k-1)} - a_{ij} x_{i+2}^{(k-1)}$$

Особенностью метода является использование предыдущих приближений $(k-1)$ при вычислении следующих (k) приближений. Итерации прекращаются, когда погрешность вычислений приблизится к заданной точности

$$|x^{k+1} - x^k| \leq \varepsilon.$$

Вывод

Метод Зейделя, являясь итерационным методом, более эффективен при решении СЛАУ большой размерности. Они (итерационные методы) не требуют хранения всей матрицы в ОЗУ, в отличие от прямых методов, которые более эффективны при решении небольших СЛАУ, так как позволяют найти решение за конечное число операций. Большую роль в скорости выполнения метода Гаусса-Зейдаля играют диагональные элементы матрицы A и начальные приближения – чем они ближе к настоящим значениям $x^{\rightarrow}$, тем быстрее этот вектор будет найден.

Полученные навыки программирования методов решения СЛАУ могут быть полезны, если захочу развиваться с сфере программирования графических библиотек, где подобные операции являются фундаментальными.

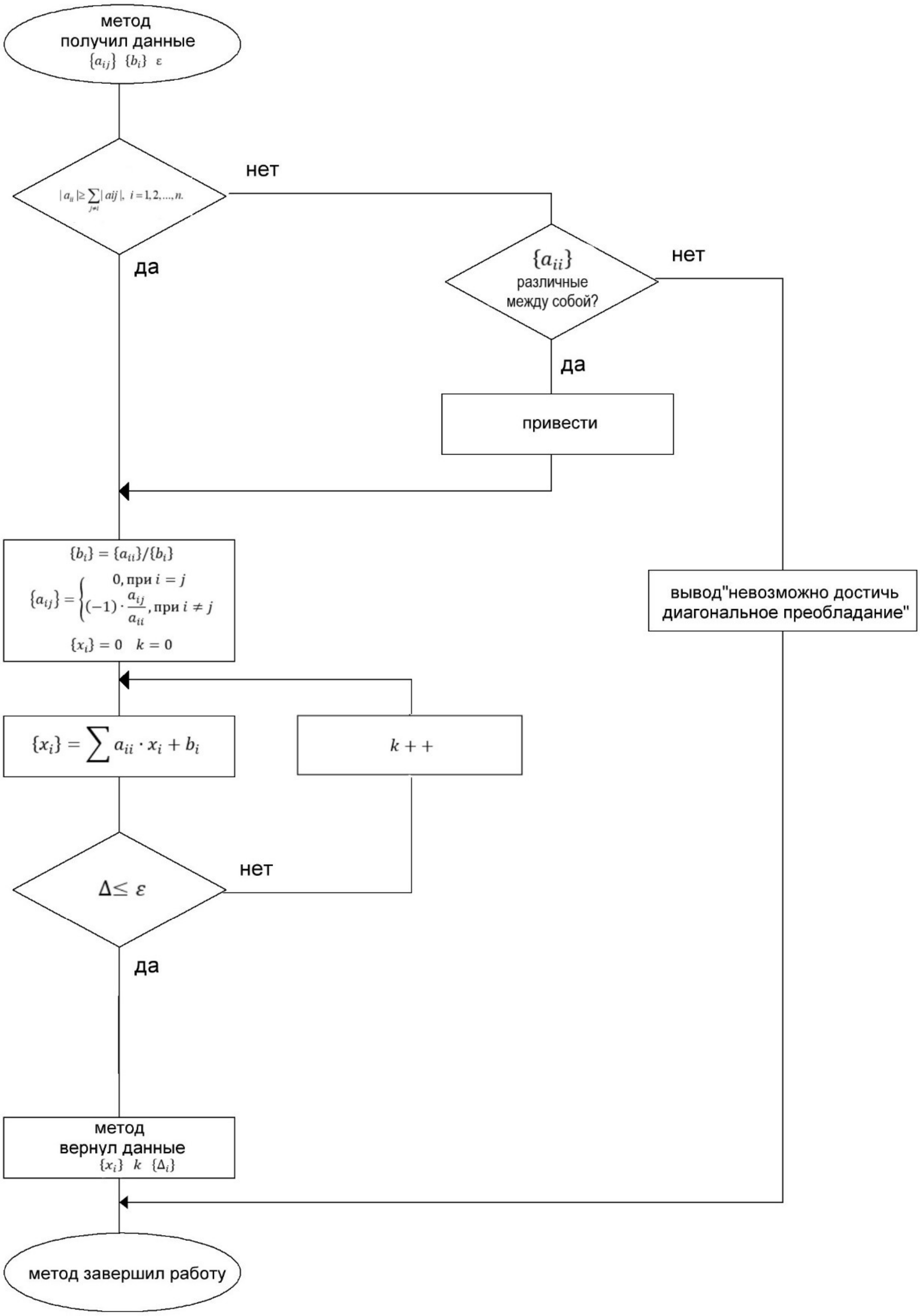
Код численного метода представлен ниже. Полный код можно найти по ссылке: https://github.com/testpassword/Computational-mathematics/tree/master/lab1-26.02.20

Примеры

```
Генерирование матрицы размерностью 6...

Структура данных матрицы создана.

[244904.0, 211.0, 730.0, 157.0, 351.0, 226.0]  *  x  = 883.0

[977.0, 2027430.0, 710.0, 956.0, 773.0, 996.0]  *  x  = 120.0

[455.0, 499.0, 1083696.0, 12.0, 165.0, 109.0]  *  x  = 653.0

[74.0, 346.0, 57.0, 258995.0, 30.0, 547.0]  *  x  = 355.0

[721.0, 369.0, 727.0, 394.0, 1130766.0, 453.0]  *  x  = 253.0

[194.0, 535.0, 813.0, 102.0, 37.0, 1870284.0]  *  x  = 539.0


Введите точность ε [0.000001 ; 1]

0.0001

Вектор неизвестных = [0.0036021940937404897, 5.6370841111458075E-5,
6.009514785920086E-4, 0.0013688133794582925, 2.2044843343636245E-4,
2.8746151217514694E-4]

Погрешности = ± [3.3003040643643164E-6, 1.079937447281146E-6, 7.577638586151337E-8,
6.302657535740396E-7, 1.1243555979781176E-7, 7.207883653001859E-10]

Количество итераций = 2
```

```
Чтение из файла V:\itmo\2 course\computational mathematics\lab1-
26.02.20\docs\ex5.txt...

Структура данных матрицы создана.

[1.0, 3.0, 5.0]  *  x  = 6.0

[4.0, 1.0, 2.0]  *  x  = 8.0

[3.0, 7.0, 3.0]  *  x  = 2.0


Введите точность ε [0.000001 ; 1]

0.000001

Вектор неизвестных = [1.5000000414646992, -0.9999998756078686, 1.4999999170717813]

Погрешности = ± [1.2439868379843233E-7, 3.731750830571201E-7, 2.487847867715942E-7]

Количество итераций = 12
```

Блок-схема


метод
получил данные
$\{a_{ij}\}$ $\{b_i\}$ ε
$|a_{ii}| \geq \sum_{j \neq i} |aij|, \ i = 1, 2, \ldots, n.$
нет
да
$\{a_{ii}\}$
различные
между собой?
нет
да
привести
$\{b_i\} = \{a_{ii}\}/\{b_i\}$
$\{a_{ij}\} = \begin{cases} 0, \text{при } i = j \\ (-1) \cdot \frac{a_{ij}}{a_{ii}}, \text{при } i \neq j \end{cases}$
$\{x_i\} = 0 \quad k = 0$
вывод"невозможно достичь
диагональное преобладание"
$\{x_i\} = \sum a_{ii} \cdot x_i + b_i$
$k++$
$\Delta \leq \varepsilon$
нет
да
метод
вернул данные
$\{x_i\}$ k $\{\Delta_i\}$
метод завершил работу

```kotlin
import kotlin.math.abs

/**
 * Функция-расширения, возвращающяя, диагональные элементы любой матрицы.
 * @return массив, состоящий из диагональных элементов матрицы.
 */
fun Array<DoubleArray>.getDiagonalElements() = mapIndexed { i, el -> el[i] }

/**
 * Класс, содержащий результат решения СЛАУ, согласно шаблону Data Transfer Object.
 * @property xVector вектор незивестных.
 * @property infelicity столбец погрешностей вычислений.
 * @property count количество итераций, за которое метод нашёл решение.
 */
data class GaussSeidelAnswer(val xVector: DoubleArray, val infelicity: DoubleArray, val counter: Int)

/**
 * Находит столбец-вектор неизвестных для СЛАУ.
 * @author Артемий Кульбако.
 */
class LinearSystemSolver {

    companion object {
        /**
         * Решает СЛАУ методом Гаусса-Зейдаля.
         * @param linearSystem СЛАУ.
         * @param precision точность вычислений.
         * @param modify true - если разрешено изменять матрицу, false - если запрещено (работа с клоном).
         * @return GaussSeidelAnswer.
         * @throws Exception если невозможно привести матрицу к диагональному преобразованию.
         */
        fun solveByGaussSeidel(linearSystem: LinearSystem, precision: Double, modify: Boolean): GaussSeidelAnswer {
            val linSys = if (modify) clone(linearSystem) else linearSystem
            linSys.let {
                if (!toDiagonalPrevalence(it)) throw Exception("Невозможно достичь дзен диагональное преобладание. Итерации расходятся.")
                transform(it)
                return iterate(it, precision)
            }
        }

        private fun clone(linSys: LinearSystem) =
            LinearSystem(linSys.equations.map { it.clone() }.toTypedArray(), linSys.resVector.clone())

        private fun toDiagonalPrevalence(linSys: LinearSystem): Boolean {

            fun isDiagonalPrevalence(matrix: Array<DoubleArray>): Boolean {
                var condition1 = 0 //все эл. главной диагонали должно быть >= сумме модулей коэф. остальных ур-я
                var condition2 = false //хотя бы 1 из элементов должен быть > сумме модулей коэф. своего ур-я
                matrix.forEachIndexed { i, el ->
                    val diagEl = abs(el[i])
                    val seriesSum = el.sumByDouble { abs(it) } - diagEl
                    if (diagEl >= seriesSum) condition1++
                    if (diagEl > seriesSum) condition2 = true
                }
                return (condition1 == matrix.size) && condition2
            }

            val A = linSys.equations
            val r = A.indices
            if (!isDiagonalPrevalence(A)) {
                val maxValuesIndices = A.map { it.indexOf(it.maxBy { number -> abs(number) }!!) }
                for (i in r)
                    for (j in r)
                        if (i != j)
                            if (maxValuesIndices[i] == maxValuesIndices[j]) return false
                A.sortBy { it.indexOf(it.maxBy { number -> abs(number) }!!) }
                val B = linSys.resVector.clone()
                linSys.resVector.forEachIndexed { i, _ -> linSys.resVector[maxValuesIndices[i]] = B[i] }
            }
            return true
        }

        private fun transform(linSys: LinearSystem) {
            linSys.resVector = linSys.resVector.zip(linSys.equations.getDiagonalElements()) { a, b -> a / b }.toDoubleArray()
```

```
            linSys.equations = linSys.equations.mapIndexed { i, doubles ->
                doubles.mapIndexed { j, d ->
                    if (i == j) 0.0 else (-1) * d / doubles[i]
                }.toDoubleArray()
            }.toTypedArray()
        }

        private fun iterate(linSys: LinearSystem, precision: Double): GaussSeidelAnswer {

            fun isAccuracyReached(newX: DoubleArray, oldX: DoubleArray, precision: Double) =
                (newX.zip(oldX) { a, b -> abs(a - b) }.toDoubleArray().max()!! < precision)

            var iterCounter = 0
            val newApproximations = DoubleArray(linSys.size) { 0.0 } //x(0)
            var oldApproximations: DoubleArray
            do {
                oldApproximations = newApproximations.clone()
                newApproximations.forEachIndexed { i, _ ->
                    var sum = 0.0
                    newApproximations.forEachIndexed { j, d -> sum += linSys.equations[i][j] * d }
                    newApproximations[i] = linSys.resVector[i] + sum
                }
                iterCounter++
            } while (!isAccuracyReached(newApproximations, oldApproximations, precision))
            val infelicity = newApproximations.zip(oldApproximations) { a, b -> abs(a - b) }.toDoubleArray()
            return GaussSeidelAnswer(newApproximations, infelicity, iterCounter)
        }
    }

}
```