



Міністерство освіти та науки України
Національний технічний університет України
“Київський політехнічний інститут”
Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

ЛАБОРАТОРНА РОБОТА № 5
З дисципліни «Алгоритми та методи обчислень»

На тему «Розв’язання систем лінійних алгебраїчних рівнянь»

Виконав:
студент 2 курсу ФІОТ
групи ІВ-71
Мазан Я. В.
Залікова – 7109

Перевірив:
ст.вик. Порев В. М.

Мета:

Вивчити алгоритми методів розв'язання систем лінійних алгебраїчних рівнянь на ЕОМ.

Завдання:

Відповідно до варіанту завдання скласти схему алгоритму розв'язання систем лінійних алгебраїчних рівнянь зазначеним у варіанті методом. Відповідно до блок-схеми скласти програму розв'язання систем лінійних алгебраїчних рівнянь алгоритмічною мовою, узгодженою з викладачем. Розв'язати СЛАР на ЕОМ відповідно до варіанту.

Індивідуальне завдання:

| | | | | | |
|-------------|------|------|------|------|--|
| 9 | 2,36 | 2,37 | 2,13 | 1,48 | |
| Метод Якобі | 2,51 | 2,40 | 2,10 | 1,92 | |
| | 2,59 | 2,41 | 2,06 | 2,16 | |

Код програми:

```
package com.labworks.lab5
import java.util.*
class EquationsSystem(var A: MutableList<Array<Double>>, var B: Array<Double>) {
    @Deprecated(message = "Works incorrectly")
    fun makeConvergent(a: MutableList<Array<Double>>, b: Array<Double>) : List<Any> {
        var newA = a
        var newB = b
        var iterations_num = 0
        val joint_matrix = Array(a.size, {i -> a[i].plus(b[i])})
        val rowsCollection = joint_matrix.toMutableList()
        val permutations = PermutationIterator(rowsCollection) // Collections2.permutations(rowsCollection)
        while (permutations.hasNext()) {
            val el = permutations.next()
            val iter_el = el.map({elem -> elem.dropLast(1).toTypedArray()}).toTypedArray()
            val iter_free_members = Array(el.size, {i -> el[i].last()})
            if (solutionConverges(iter_el.toMutableList())) {
                //split free_vals and coefficients in equations
                val coeffs = iter_el
                val free_values = Array(a.size, {i -> el[i].last()})
                return listOf(coeffs, free_values)
            }
        }
        while (!solutionConverges(newA)) {
            println(iterations_num++)
            println(EquationsSystem(newA, newB))
            val newEquation = randomAction(a, b)
            newA = newEquation[0] as MutableList<Array<Double>>
            newB = newEquation[1] as Array<Double>
        }
        println(iterations_num)
        return listOf(newA, newB)
    }
}
fun normForm(a: MutableList<Array<Double>>, b: Array<Double>) : List<Any> {
    val n = a.size
```

```

    val normA = MutableList(n, {k -> Array(n, {i -> if (k != i) -a[k][i]/a[k][k] else 0.0})})
    val normB = Array(n, {k -> b[k]/a[k][k]})
    return listOf(normA, normB)
}

fun solutionConverges(coeffs_matrix: MutableList<Array<Double>>): Boolean {
    val abs_matrix = Array(coeffs_matrix.size, {i -> Array(coeffs_matrix.size, {j -> Math.abs(coeffs_matrix[i][j])})})
    val convergency: Array<Boolean> = Array(abs_matrix.size, {index -> abs_matrix[index].sum() - abs_matrix[index][index] <= abs_matrix[index][index]})
    return convergency.fold(true) {conv, el -> conv && el}
}

fun random(): EquationsSystem {
    val newEqu = randomAction(A, B)
    A = newEqu[0] as MutableList<Array<Double>>
    B = newEqu[1] as Array<Double>
    return this
}

/**
 * Conduct a random operation on our matrix attempting to make it convergable some time
 * random actions:
 * 1. Swap rows in equation
 * 2. Add some row, multiplied on random number, to other row
 */
private fun randomAction(a: MutableList<Array<Double>>, b: Array<Double>): List<Any> {
    val randomiser = Random()
    val action = randomiser.nextInt(4)
    when(action) {
        0,1,2 -> return swapRandomRows(a, b)
        3 -> return addRandomRows(a, b)
        else -> return addRandomRows(a, b)
    }
}

private fun swapRandomRows(a: MutableList<Array<Double>>, b: Array<Double>): List<Any> {
    val randomiser = Random()
    var row1: Int
    var row2: Int
    do {
        row1 = randomiser.nextInt(a.size)
        row2 = randomiser.nextInt(a.size)
    } while (row1 == row2)
    val temp = a[row1]
    val newA = a
    val newB = b
    newA[row1] = newA[row2]
    newA[row2] = temp
    val temp2 = b[row1]
    newB[row1] = newB[row2]
    newB[row2] = temp2
    return listOf(newA, newB)
}

private fun addRandomRows(a: MutableList<Array<Double>>, b: Array<Double>): List<Any> {
    val randomiser = Random()
    val newA = a
    val newB = b
    var row1: Int
    var row2: Int
    do {
        row1 = randomiser.nextInt(a.size)
        row2 = randomiser.nextInt(a.size)
    } while (row1 == row2)
    val r = (randomiser.nextDouble() - 0.5) * 4
    val randomMultiplier = if (r != 0.0) r else 1.0
    val multipliedRow = Array(a.size, {i -> newA[row1][i] * randomMultiplier})
    newA[row2] = Array(a.size, {i -> newA[row2][i] + multipliedRow[i]})
    val freeMembMultiplied = newB[row1] * randomMultiplier
    newB[row2] += freeMembMultiplied
    return listOf(newA, newB)
}

private fun nextIter(prevX: Array<Double>): Array<Double> {
    return Array(prevX.size, {i -> List(prevX.size, {j -> if (i != j) (A[i][j] * prevX[j] - B[i]) / -A[i][i] else 0.0}).sum()})
}

fun solve(epsilon: Double): Array<Double> {
    var iter1 = Array(A.size, {i -> 0.0})
    var iter2 = nextIter(iter1)
    val difference = {x0: Array<Double>, x1: Array<Double>, precision: Double -> Array(x0.size, {i -> Math.abs(x0[i] - x1[i])}).max()!! < epsilon}
    var iterNum = 0
    while (!difference(iter1, iter2, epsilon)) {
        iterNum++
        iter1 = iter2
        iter2 = nextIter(iter1)
    }
}

```

```

    }
    println(iterNum)
    return iter2.map({el -> el/(A.size-1)}).toArray()
}
fun solveUnconvergent(epsilon: Double) : Array<Double> {
    var iter1 = Array(A.size, {i -> 0.0})
    var iter2 = nextIter(iter1)
    val difference = {x0: Array<Double>, x1: Array<Double>, precision: Double -> Array(x0.size, {i -> Math.abs(x0[i]- x1[i]))}.max()!! < epsilon}
    var iterNum = 0
    while (!difference(iter1, iter2, epsilon) && iterNum < 10000) {
        iterNum++
        iter1 = iter2
        iter2 = nextIter(iter1)
    }
    println(iterNum)
    return iter2.map({el -> el/(A.size-1)}).toArray()
}
@Override
override fun toString() : String {
    var res = ""
    for (i in 0..A.size-1) {
        var row = ""
        for (j in 0..A[0].size-1) {
            row += A[i][j].toString() + "; "
        }
        row += " = ${B[i]}"
        res += row + "\n"
    }
    return res
}
}

```

package com.labworks.lab5

```

class MakeConvergent(system: EquationsSystem) {
    val correctEquation = makeConvergent(system)
    /**
     * @return convergent variant of input matrix
     */
    fun makeConvergent(equation: EquationsSystem) : EquationsSystem {
        val coeffs_matrix = equation.A
        val free_vals = equation.B
        val joint_matrix = Array(coeffs_matrix.size, {i -> coeffs_matrix[i].plus(free_vals[i])})
        val rowsCollection = joint_matrix.toMutableList()
        val permutations = PermutationIterator(rowsCollection) // Collections2.permutations(rowsCollection)
        while (permutations.hasNext()) {
            val el = permutations.next()
            val elFreeVals = Array(el.size, {i -> el[i].last()})
            val elCoeffs = el.map({elem -> elem.dropLast(1).toArray()}).toMutableList()
            if (equation.solutionConverges(elCoeffs)) {
                //split free_vals and coefficients in equations
                val coeffs = elCoeffs
                val free_values = Array(coeffs_matrix.size, {i -> el[i].last()})
                return EquationsSystem(coeffs, free_values)
            }
        }
        val a = equation.A
        val b = equation.B
        var iterations_num = 0
        // while (!equation.solutionConverges(a, b)) {
        //     println(iterations_num++)
        //     println(EquationsSystem(newA, newB))
        //     val newEquation = randomAction(a, b)
        //     newA = newEquation[0] as MutableList<Array<Double>>
        //     newB = newEquation[1] as Array<Double>
        // }
        println(iterations_num)
        return listOf(newA, newB)
        throw ArithmeticException("Matrix can't be made converged")
    }
}

```

Результати виконання програми:

23:53  38

АМО лабораторна 5

Введіть коефіцієнти та вільні члени СЛАР:

$2.2 x_1 + 2 x_2 + 15 x_3 = 2.6$

$2.6 x_1 + 16.6 x_2 + 4 x_3 = 4.4$

$22 x_1 + 2 x_2 + -1 x_3 = 5$

Введіть точність ϵ

0.001

РОЗВ'ЯЗАТИ СЛАР

$x_1 = 0,304 \pm \epsilon;$
 $x_2 = -0,829 \pm \epsilon;$
 $x_3 = 0,036 \pm \epsilon$

Висновок:

Під час виконання даної лабораторної роботи я навчився програмувати розв'язання СЛАР з використанням методу Якобі, який забезпечує обчислення невідомих коефіцієнтів з певною точністю. Лабораторна робота виконана на Android із використанням Android Studio. Під час виконання роботи проблем не виникло.