МІНІСТЕРСВО ОСВІТИ І НАУКИ УКРАІНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

ЛАБОРАТОРНА РОБОТА №2

з навчальної дисципліни

«*Чисельні методи*»

на тему:

*«РОЗВ’ЯЗУВАННЯ СИСТЕМ ЛІНІЙНИХ АРИФМЕТИЧНИХ РІВНЯНЬ»*

*варіант 8*

Виконав:

студент ІІ курсу, групи К-27

спеціальності «Комп’ютерні науки. Інформатика»

*Некряч Владислав Вадимович*

*Київ, 2021 рік*

ЗМІСТ

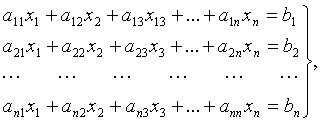
[1. Метод Гаусса 3](#_Toc67953006)

[2. Метод квадратного кореня 10](#_Toc67953007)

1. Метод Гаусса

**Теоретичні відомості**

Для збіжності ітераційного процесу суттєве значення

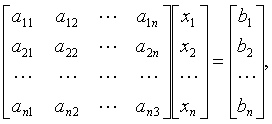
Система n лінійних рівнянь з n невідомими у загальному вигляді може бути записана:

(2.1)

де - невідомі величини, які необхідно визначити при розв’язанні системи;

- коефіцієнти при невідомих;

- вільні члени рівнянь системи.

У матричній формі ця система рівнянь має вигляд:

(2.2)

або ,

де А - квадратна матриця коефіцієнтів системи розмірністю ,

- вектор невідомих;

- вектор вільних членів.

Розв’язати систему (2.1) - означає обчислити такі значення елементів вектора невідомих , за яких кожне рівняння системи перетворюється на тотожність.

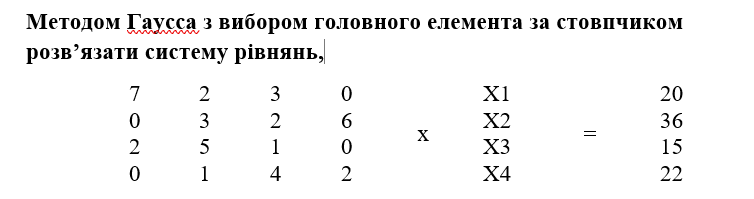
Для цього можна застосувати як прямі, так і ітераційні методи. Поширеним прямим способом розв’язання систем лінійних рівнянь є алгоритм послідовного виключення невідомих, що має назву метод Гауса.

Існують різні алгоритми його реалізації. Один із них - метод Гауса із зворотнім ходом для розв’язання СЛАР розглядається у цій роботі.

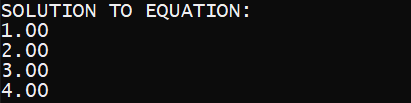
Метод Гауса із зворотнім ходом передбачає виконання двох етапів: прямий і зворотній хід методу. Прямий хід - послідовність однотипних кроків виключення невідомих із системи рівнянь. В результаті його виконання вихідна система (2.1) або (2.2) з квадратною матрицею коефіцієнтів перетворюється на еквівалентну систему рівнянь з верхньою трикутною матрицею коефіцієнтів. На зворотному ході обчислюються значення невідомих, починаючи з останнього (від до ).

Для перевірки правильності розв’язання системи рівнянь, необхідно обчислені значення невідомих підставити у вихідну систему (2.1). Усі рівняння повинні перетворитись при цьому на тотожності.

**Постанова задачі**



**Розв’язок**

****

**Фрагмент коду(Solver.tsx)**

import pStepService = require('./pStep')

import mStepService = require('./mStep')

type Array4 = [number,number,number,number] | []

type Array4x4 = [Array4, Array4, Array4, Array4]

const printMatrix4x4 = (matrix: Array4x4) =>

{

  for (let i = 0; i < 4; i++) {

    for (let j = 0; j < 4; j++) {

      process.stdout.write((matrix[i][j].toFixed(2)).padEnd(10))

    }

    console.log(  )

  }

}

const printVector = (vector: Array4) =>

{

  for (let i = 0; i < 4; i++) {

    console.log(vector[i].toFixed(2) + " ")

  }

}

const multiplyMatrixByMatrix = (firstMatrix: Array4x4, secondMatrix: Array4x4) =>

{

  let resultMatrix: Array4x4 = [[], [], [], []]

  for (let i = 0; i < 4; i++) {

    resultMatrix[i] = [0, 0, 0, 0];

  }

  for (let i = 0; i < 4; i++) {

    for (let j = 0; j < 4; j++) {

      //console.log(`indices for result matrix: ${i}, ${j}`)

      //console.log('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

      for (let k = 0; k < 4; k++) {

        resultMatrix[i][j] += firstMatrix[i][k] \* secondMatrix[k][j]

        //console.log(`first matrix element: ${firstMatrix[i][k]}, second matrix element: ${secondMatrix[k][j]}`)

        /\*printMatrix(resultMatrix)\*/

      }

    }

  }

  return resultMatrix

}

const multiplyMatrixByVector = (matrix: Array4x4, vector: Array4) =>

{

  const newVector: Array4 = new Array(4).fill(0) as Array4

  for (let i = 0; i < 4; i++) {

    for (let j = 0; j < 4; j++) {

      newVector[i] += matrix[i][j] \* vector[j]

    }

  }

  return newVector

}

const backwardsSolving = (matrix: Array4x4, vector: Array4) =>

{

  const solution: Array4 = [0, 0, 0, 0]

  for (let i = 3; i >= 0; i--) {

    let subtractionPart = 0

    for (let j = i; j < 4; j++) {

      subtractionPart += matrix[i][j] \* solution[j]

    }

    solution[i] = vector[i] - subtractionPart

  }

  return solution

}

const solver = (currentMatrix: Array4x4, constantsVector: Array4) => {

  console.log ('matrix before gauss method:')

  printMatrix4x4(currentMatrix)

  console.log ('vector before gauss method:')

  printVector(constantsVector)

  for (let i = 0; i < 4; i++) {

    let [newMatrixP, newConstantsVectorP] : (Array4x4 | Array4)[] = pStepService.pStep(currentMatrix, constantsVector, i)

    console.log (`matrix after row swap #${i + 1}:`)

    printMatrix4x4(newMatrixP as Array4x4)

    console.log (`vector after row swap #${i + 1}:`)

    printVector(newConstantsVectorP as Array4)

    let [newMatrixM, newConstantsVectorM] : (Array4x4 | Array4)[] = mStepService.mStep(newMatrixP as  Array4x4, newConstantsVectorP as Array4, i)

    console.log (`matrix after normalizing #${i + 1}:`)

    printMatrix4x4(newMatrixM as Array4x4)

    console.log (`vector after normalizing #${i + 1}:`)

    printVector(newConstantsVectorM as Array4)

    currentMatrix = newMatrixM as Array4x4

    constantsVector = newConstantsVectorM as Array4

  }

  let solution : Array4 = backwardsSolving(currentMatrix, constantsVector)

  console.log('SOLUTION TO EQUATION:')

  printVector(solution)

}

export {

  multiplyMatrixByMatrix,

  multiplyMatrixByVector,

  printMatrix4x4,

  printVector,

  solver

}

**PStep.tsx**

import matrixService = require('./matrixService')

type Array4 = [number,number,number,number] | []

type Array4x4 = [Array4, Array4, Array4, Array4]

const pStep = (currentMatrix: Array4x4, constantsVector: Array4, index: number) => {

  let maxRow: number = -1

  maxRow = findMax(currentMatrix, index).ind

  //console.log('maxRow index:', maxRow)

  console.log('keys', matrixService.multiplyMatrixByMatrix)

  const pMatrix : Array4x4 = createPMatrix(currentMatrix, maxRow, index)

  //matrixService.printMatrix4x4(pMatrix)

  const updatedMatrix : Array4x4 = matrixService.multiplyMatrixByMatrix(pMatrix, currentMatrix)

  const updatedVector : Array4 = matrixService.multiplyMatrixByVector(pMatrix, constantsVector)

  return [updatedMatrix, updatedVector]

}

const findMax = (currentMatrix: Array4x4, columnIndex: number) => {

  const column : Array4 = [0,0,0,0]

  for (let i = 0; i < 4; i++) {

    column[i] = currentMatrix[i][columnIndex]

  }

  return column.reduce((currMaxPair: {val: number, ind: number}, currEl: number, currInd: number) => {

    return (currMaxPair.val < currEl)

    ? {val: currEl, ind: currInd}

    : currMaxPair

  }, {val: Number.MIN\_VALUE, ind: -1})

}

const createPMatrix = (currentMatrix: Array4x4, rowIndexOfMaxEl: number, rowIndexOfCurRow: number) => {

  const pMatrix: Array4x4 = [[],[],[],[]]

  for(let i = 0;  i < 4; i++) {

      pMatrix[i] = [0, 0, 0, 0];

  }

  for (let i = 0; i < 4; i++) {

    if (i === rowIndexOfCurRow) {

      pMatrix[i][rowIndexOfMaxEl] = 1

    }

    else if (i === rowIndexOfMaxEl) {

      pMatrix[i][rowIndexOfCurRow] = 1

    }

    else pMatrix[i][i] = 1

  }

  return pMatrix

}

export { pStep }

**MStep.tsx**

import matrixService = require('./matrixService')

type Array4 = [number,number,number,number] | []

type Array4x4 = [Array4, Array4, Array4, Array4]

const mStep = (currentMatrix: Array4x4, currentVector: Array4, rowIndex: number) : [Array4x4, Array4] =>  {

  const mMatrix : Array4x4 = createMMatrix(currentMatrix, rowIndex)

 // matrixService.printMatrix4x4(mMatrix)

  const newMatrix = matrixService.multiplyMatrixByMatrix(mMatrix, currentMatrix)

  const newVector = matrixService.multiplyMatrixByVector(mMatrix, currentVector)

  return [newMatrix, newVector]

}

const createMMatrix = (currentMatrix: Array4x4, rowIndex: number) => {

  const mMatrix : Array4x4 = [[],[],[],[]]

  for (let i = 0; i < 4; i++) {

    mMatrix[i] = [0,0,0,0]

  }

  for (let i = 0; i < rowIndex; i++){

    mMatrix[i][i] = 1

  }

  for (let i = rowIndex; i < 4; i++) {

    if (i === rowIndex) {

      mMatrix[i][i] = 1 / currentMatrix[i][i]

    }

    else {

      mMatrix[i][i] = 1

      mMatrix[i][rowIndex] = -(currentMatrix[i][rowIndex]/currentMatrix[rowIndex][rowIndex])

    }

  }

  return mMatrix

}

export {mStep}

1. Метод квадратного кореня

**Теоретичні відомості**

Його застосовують для розв’язання СЛАР із неособливою симетричною матрицею . Цей метод базується на тому, що симетричну матрицю можна подати у вигляді

де права трикутна матриця

а діагональним матриця з елементами .

Елементи матриці та обчислюють за формулами

Тоді розв’язання СЛАР зводиться до розв’язання наступних двох СЛАР

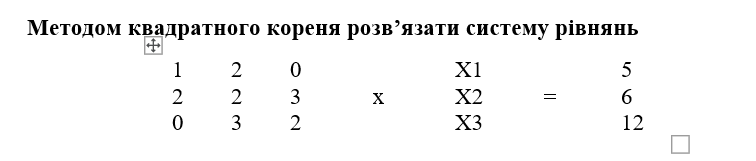
Матриця ліва трикутна. Це дає змогу знайти розв’язок системи , виконавши зворотній хід методу Гаусса зверху вниз, а після цього розв’язати систему , виконавши, як у методі Гаусса, зворотній хід знизу вгору (тобто починаючи з останнього рівняння).

Отже, алгоритм методу квадратного кореня можна подати так:

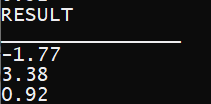
1. перевірити, чи симетрична матриця
2. обчислити елементи матриць і за формулою
3. розв’язати СЛАР і знайти вектор
4. розв’язати СЛАР і знайти шуканий розв’язок системи

Загальна кількість арифметичних операцій, які потрібно виконати для реалізації методу квадратного кореня, має порядок .

**Постанова задачі**



**Розв’язок (округлено до 2 знаків після коми для читабельності)**

****

Фрагмент коду (Solver.tsx)

type Array3 = [number, number, number] | []

type Array3x3 = [Array3, Array3, Array3]

const multiplyMatrices = (firstMatrix: Array3x3, secondMatrix: Array3x3) => {

  let resultMatrix: Array3x3 = [[], [], []]

  for (let i = 0; i < 3; i++) {

    resultMatrix[i] = [0, 0, 0];

  }

  for (let i = 0; i < 3; i++) {

    for (let j = 0; j < 3; j++) {

      //console.log(`indices for result matrix: ${i}, ${j}`)

      //console.log('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

      for (let k = 0; k < 3; k++) {

        resultMatrix[i][j] += firstMatrix[i][k] \* secondMatrix[k][j]

        //console.log(`first matrix element: ${firstMatrix[i][k]}, second matrix element: ${secondMatrix[k][j]}`)

        /\*printMatrix(resultMatrix)\*/

      }

    }

  }

  return resultMatrix

}

const printMatrix3x3 = (matrix: Array3x3) =>

{

  console.log('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

  for (let i = 0; i < 3; i++) {

    for (let j = 0; j < 3; j++) {

      process.stdout.write((matrix[i][j].toFixed(2)).padEnd(10))

    }

    console.log(  )

  }

}

const printVector = (vector: Array3) =>

{

  console.log('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

  for (let i = 0; i < 3; i++) {

    console.log(vector[i].toFixed(2) + " ")

  }

}

const backwardsSolvingTop = (matrix: Array3x3, vector: Array3) =>

{

  const solution: Array3 = [0, 0, 0]

  printMatrix3x3(matrix)

  printVector(vector)

  for (let i = 0; i < 3; i++) {

    const rowCoefficient = matrix[i][i]

    for (let j = 0; j < 3; j++) {

      matrix[i][j] /= rowCoefficient

    }

    vector[i] /= rowCoefficient

  }

  printMatrix3x3(matrix)

  printVector(vector)

  for (let i = 0; i < 3; i++) {

    let subtractionPart = 0

    for (let j = 0; j < i; j++) {

      subtractionPart += matrix[i][j] \* solution[j]

    }

    solution[i] = vector[i] - subtractionPart

  }

  return solution

}

const backwardsSolvingBottom = (matrix: Array3x3, vector: Array3) =>

{

  const solution: Array3 = [0, 0, 0]

  printMatrix3x3(matrix)

  printVector(vector)

  for (let i = 0; i < 3; i++) {

    const rowCoefficient = matrix[i][i]

    for (let j = 0; j < 3; j++) {

      matrix[i][j] /= rowCoefficient

    }

    vector[i] /= rowCoefficient

  }

  printMatrix3x3(matrix)

  printVector(vector)

  for (let i = 2; i >= 0; i--) {

    let subtractionPart = 0

    for (let j = i; j < 3; j++) {

      subtractionPart += matrix[i][j] \* solution[j]

    }

    solution[i] = vector[i] - subtractionPart

  }

  return solution

}

const createSDMatrices = (currentMatrix: Array3x3) => {

  const dMatrix : Array3x3 = [[],[],[]]

  const sMatrix: Array3x3 = [[],[],[]]

  for (let i = 0; i < 3; i++) {

    dMatrix[i] = [0,0,0]

    sMatrix[i] = [0,0,0]

  }

  // fill in diagonal elements

  for (let i = 0; i <= 2; i++) {

    let subtractionPart = 0

    for (let p = 0; p <= (i - 1); p++) {

      subtractionPart += sMatrix[p][i] \* sMatrix[p][i] \* dMatrix [p][p]

    }

    dMatrix[i][i] = Math.sign(currentMatrix[i][i] - subtractionPart)

    sMatrix[i][i] = Math.sqrt(Math.abs(currentMatrix[i][i] - subtractionPart))

    for (let j = i + 1; j <= 2; j++) {

      if (i != 2)

      {

        subtractionPart = 0

        for (let p = 0; p <= (i - 1); p++) {

          subtractionPart += sMatrix[p][i] \* dMatrix[p][p] \* sMatrix[p][j]

        }

        sMatrix[i][j] = (currentMatrix[i][j] - subtractionPart)/(dMatrix[i][i] \* sMatrix[i][i])

      }

    }

  }

  return [dMatrix, sMatrix]

}

const transposeMatrix = (matrix: Array3x3) => {

  const transposedMatrix : Array3x3 = [[],[],[]]

  for (let i = 0; i < 3; i++) {

    transposedMatrix[i] = [0,0,0]

  }

  for (let i = 0; i < 3; i++) {

    for (let j = 0; j < 3; j++) {

      transposedMatrix[j][i] = matrix[i][j]

    }

  }

  return transposedMatrix

}

const solver = (currentMatrix: Array3x3, constantsVector: Array3) => {

  const [dMatrix, sMatrix] : Array3x3[] = createSDMatrices(currentMatrix)

  console.log('printing matrix D and matrix S:')

  printMatrix3x3(dMatrix)

  printMatrix3x3(sMatrix)

  const sTMatrix : Array3x3 = transposeMatrix(sMatrix)

  console.log('printing transposed matrix S:')

  printMatrix3x3(sTMatrix)

  console.log('printing multiplication of sT and D:')

  const sT\_DMatrix = multiplyMatrices(sTMatrix, dMatrix)

  printMatrix3x3(sT\_DMatrix)

  // normalize matrix -> solve first equation -> use the same method for second one

  console.log('printing top-bottom gauss solving of first equation:')

  const result : Array3 = backwardsSolvingTop(sT\_DMatrix, constantsVector)

  console.log('printing vector y after first equation:')

  printVector(result)

  console.log('printing bottom-top gauss solving of second equation:')

  const finalResult : Array3 = backwardsSolvingBottom(sMatrix, result)

  console.log('RESULT')

  printVector(finalResult)

}

export {solver}