**Лабораторна робота №4**

Розв’язання систем лінійних алгебраїчних рівнянь (СЛАР)

за точними та ітераційними методами

Мета: Навчитися розв’язувати системи лінійних алгебраїчних рівнянь з застосуванням ЕОМ за ітераційними методами; набуття теоретичних знань та практичних навичок розв’язку систем лінійних алгебраїчних рівнянь з застосуванням ЕОМ за точними методами.

Завдання:

Скласти програму ЕОМ для розв’язання СЛАР (з точністю 10-5), відповідно варіанту, за методами:

- Метода Гауса (з вибором головного елементу)

- простої ітерації (Якобі);

- Гауса- Зейделя;

Короткі теоретичні відомості

Система лінійних алгебраїчних рівнянь (СЛАР) розмірністю n має вигляд:

 (1)

або в матричній формі: **AX=B**,

де **А**- матриця коефіцієнтів системи; **В**- вектор вільних членів; **X**- вектор невідомих.

Методи розв'язку СЛАР діляться на дві групи - прямі та ітераційні. Прямі методи використовують скінченні співвідношення для визначення невідомих. Ці методи дають розв'язок за чітко визначену кількість арифметичних операцій. До них належать відомий зі школи метод виключення невідомих Гаусса, застосування формул Крамера, різноманітні методи обернення матриці коефіцієнтів системи. Ці методи порівняно прості і найбільш універсальні, але не враховують структуру матриці А, що призводить при значних *п* до перевитрат комп'ютерних ресурсів (пам'яті, часу). Крім того, суттєвим недоліком прямих методів є накопичення похибок в процесі розв'язку задачі, тому що обчислення на будь-якому етапі використовують результати попередніх операцій. Це особливо небезпечно для великих систем, коли значно збільшується загальна кількість операцій, а також для слабо обумовлених матриць, які дуже чутливі до похибок. Тому прямі методи, як правило, застосовують для порівняно невеликих систем з щільно заповненою матрицею А, у якої не близький до нуля визначник

**Метод Гауса з вибором головного елемента**

Ідея цього методу виникла у зв’язку з тим, що коефіцієнти СЛАР є параметрами реальних інженерних систем та в більшості є наближеними значеннями, тому що отримані звичайно в результаті вимірювання або як статистичні дані. Для таких систем рівнянь при обчисленні масштабного множника можлива ситуація при визначені m1_t1_lecture2_image175, що ділення наближеного числа m1_t1_lecture2_image177 на достатньо мале число m1_t1_lecture2_image175 веде до різкого збільшення похибки методу. Тому для того, щоб не збільшувати похибку результату необхідно виконувати такі дії:

1) в системі (1) необхідно знайти з *k-*го стовпця найбільший за абсолютним значенням коефіцієнт *ak j*;

2) переставити *k-*те рівняння з рівнянням у якому знаходиться цій максимальний коефіцієнт;

Ітераційні методи розв’язання СЛАР

Розроблені ітераційні методи розв’язання СЛАР, які дозволяють послідовно уточнювати розв’язок, починаючи з деякого наближення. Для цих методів також існують межі точності, яку можна досягнути при їх застосуванні, але вони можуть виявитися роботоспроможними , коли “точні” вже не працюють. Крім того, вони, у певних випадках, вимагають меншої пам’яті ЕОМ.

# Метод простої ітерації

*.* Для реалізації такого методу систему  зводять до вигляду:



або в матричній формі

;

тобто , де 

Задавши тепер початковий вектор невідомих Х(0), можна одержати ітераційний процес:; і т. д. доти, поки модуль різниці норм векторів X(I) та Х(I+1) не стане меншим заданої точністі:

.

Такий процес буде збіжним лише тоді, коли

 або ,

тобто якщо будь-яка норма матриці α буде менша за одиницю. Ця умова по відношенню до матриці А набуває такого змісту: процес прямої ітерації буде збіжним, якщо модулі діагональних елементів матриці А будуть більші за суму модулей її сторонніх елементів:



**МЕТОД ГАУСА-ЗЕЙДЕЛЯ**

***Метод Гауса-Зейделя*** *-* модифікація методу простої ітерації, суть якого полягає в тому, що при визначенні поточного наближення невідомого Хi враховуються визначені на цій же ітерації наближення невідомих Х1,Х2,...Хi-1.

**Хід роботи**

1. Текст програми

public class Main {

private double eps = 0.0000001;

private GaussianEliminationMethod gaussMethod = new GaussianEliminationMethod();

private IterativeMethod iterativeMethod = new IterativeMethod(eps);

private GetResidual getResidualCommand = new GetResidual();

private Matrix A;

private Vector B;

private int size;

public static void main(String[] args) {

Main main = new Main();

main.run();

}

private void run() {

init();

runCalc();

}

private void runCalc() {

runGauss();

try {

runIterative();

} catch (RuntimeException e) {

System.out.println(e.getMessage());

}

try {

runGaussSeidel();

} catch (RuntimeException e) {

System.out.println(e.getMessage());

}

}

private void runGauss() {

hr();

System.out.println("Gaussian Elimination Method:");

Vector X = gaussMethod.execute(new Matrix(A), new Vector(B));

printResults(A, X, B);

}

private void runIterative() {

hr();

System.out.println("Iterative Method:");

Vector X = iterativeMethod.executeIterative(new Matrix(A), new Vector(B));

printResults(A, X, B);

System.out.println("Number of Iteration: " + iterativeMethod.getIterCount());

}

private void runGaussSeidel() {

hr();

System.out.println("Gauss-Seidel Method:");

Vector X = iterativeMethod.executeGaussSeidel(new Matrix(A), new Vector(B));

printResults(A, X, B);

System.out.println("Number of Iteration: " + iterativeMethod.getIterCount());

}

private void printResults(Matrix a, Vector x, Vector b) {

var residual = getResidualCommand.execute(a, x, b);

System.out.println(x);

System.out.println("Residual:");

System.out.println(residual);

}

protected void init() {

size = promptInt("Enter the number of equations:");

A = new Matrix(size, size);

B = new Vector(size);

enterEquation();

}

private void enterEquation() {

System.out.println("Enter the equation (each line is a line of A and a B's item at the end)");

Scanner scanner = new Scanner(System.in);

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

A.set(i, j, scanner.nextDouble());

}

B.set(i, scanner.nextDouble());

}

}

}

public class Matrix {

protected int n;

protected int m;

protected double[][] matrix;

public Matrix() {

}

public Matrix(int n, int m) {

setN(n);

setM(m);

init();

}

public Matrix(Matrix matrix) {

setN(matrix.getN());

setM(matrix.getM());

init();

for (int i = 0; i < getN(); i++) {

for (int j = 0; j < getM(); j++) {

set(i, j, matrix.get(i, j));

}

}

}

public double get(int i, int j) {

if (i >= n || i < 0) return 0;

if (j >= m || j < 0) return 0;

return matrix[i][j];

}

public void set(int i, int j, double val) {

if (i >= n || i < 0) return;

if (j >= m || j < 0) return;

matrix[i][j] = val;

}

public void swapLines(int lineA, int lineB) {

var \_aLine = matrix[lineA];

matrix[lineA] = matrix[lineB];

matrix[lineB] = \_aLine;

}

public double[][] getInternal() {

return matrix;

}

protected void init() {

matrix = new double[n][m];

}

public int getN() {

return n;

}

protected void setN(int n) {

this.n = n;

}

public int getM() {

return m;

}

protected void setM(int m) {

this.m = m;

}

@Override

public String toString() {

StringBuilder res = new StringBuilder();

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) {

res.append(matrix[i][j]).append(" ");

}

res.append("\n");

}

return res.toString();

}

static public Vector mulMatToVec(Matrix mat, Vector vec) {

Vector res = new Vector(vec.getN());

for (int i = 0; i < vec.getSize(); i++) {

double iRes = 0;

for (int j = 0; j < mat.getM(); j++) {

iRes += vec.get(j) \* mat.get(i, j);

}

res.set(i, iRes);

}

return res;

}

}

public class Vector extends Matrix {

private int size;

public Vector(int size) {

super(size, 1);

this.size = size;

}

public Vector(Vector matrix) {

setN(matrix.getN());

setM(matrix.getM());

size = matrix.getN();

init();

for (int i = 0; i < size; i++) {

set(i, 0, matrix.get(i));

}

}

public double get(int i) {

return super.get(i, 0);

}

public void set(int i, double val) {

super.set(i, 0, val);

}

public int getSize() {

return size;

}

}

public class GetResidual {

public Vector execute(Matrix A, Vector X, Vector B) {

Vector XA = Matrix.mulMatToVec(A, X);

Vector residual = new Vector(B.getSize());

for (int i = 0; i < B.getSize(); i++) {

residual.set(i, abs(XA.get(i) - B.get(i)));

}

return residual;

}

}

public class GaussianEliminationMethod {

private Matrix A;

private Vector B;

private Vector X;

private int n;

public Vector execute(Matrix A, Vector B) {

this.A = A;

this.B = B;

this.n = A.getN();

forward();

back();

return X;

}

private void forward() {

// Step

for (int i = 0; i < n - 1; i++) {

selectMainElement(i);

double main = A.get(i, i);

// Line

for (int j = i + 1; j < n; j++) {

double k = A.get(j, i) / main;

// Element of A

for (int t = i; t < n; t++) {

A.set(j, t, A.get(j, t) - A.get(i, t) \* k);

}

// Element of B

B.set(j, B.get(j, 0) - B.get(i) \* k);

}

}

}

private void back() {

Vector X = new Vector(n);

for (int i = n - 1; i >= 0; i--) {

double base = 0;

for (int j = i + 1; j < n; j++) {

base += A.get(i, j) \* X.get(j);

}

double x = (B.get(i) - base) / A.get(i, i);

X.set(i, x);

}

this.X = X;

}

private void selectMainElement(int step) {

double maxV = abs(A.get(step, step));

int maxIndex = step;

for (int i = step + 1; i < n; i++) {

double cur = abs(A.get(i, step));

if (cur > maxV) {

maxV = cur;

maxIndex = i;

}

}

A.swapLines(step, maxIndex);

B.swapLines(step, maxIndex);

}

}

public class IterativeMethod {

private int size;

private double eps = 0.00000001;

private int iterCount = 0;

private int maxIterations = 10000;

private GetResidual getResidual = new GetResidual();

public IterativeMethod() {

}

public IterativeMethod(double eps) {

setEps(eps);

}

public Vector executeIterative(Matrix A, Vector B) {

size = B.getSize();

iterCount = 0;

Vector b = prepareB(A, B);

Vector x = new Vector(b);

Matrix a = prepareA(A, B);

while (!isDone(A, x, B)) {

x = nextIterativeStep(a, x, b);

iterCount++;

if (iterCount > maxIterations) {

throwsIterationsOut();

}

}

return x;

}

public Vector executeGaussSeidel(Matrix A, Vector B) {

size = B.getSize();

iterCount = 0;

Vector b = prepareB(A, B);

Vector x = new Vector(b.getSize());

Matrix a = prepareA(A, B);

while (!isDone(A, x, B)) {

x = nextGaussSeidelStep(a, x, b);

iterCount++;

if (iterCount > maxIterations) {

throwsIterationsOut();

}

}

return x;

}

private Vector nextGaussSeidelStep(Matrix a, Vector x, Vector b) {

Vector res = new Vector(x);

for (int i = 0; i < size; i++) {

double newVal = 0;

for (int j = 0; j < size; j++) {

newVal += res.get(j) \* a.get(i, j);

}

res.set(i, newVal + b.get(i));

}

return res;

}

private Vector nextIterativeStep(Matrix a, Vector x, Vector b) {

Vector res = Matrix.mulMatToVec(a, x);

for (int i = 0; i < size; i++) {

res.set(i, res.get(i) + b.get(i));

}

return res;

}

private boolean isDone(Matrix A, Vector X, Vector B) {

var residual = getResidual.execute(A, X, B);

double max = abs(residual.get(0));

for (int i = 1; i < size; i++) {

double cur = abs(residual.get(i));

if (cur > max) {

max = cur;

}

}

return max < eps;

}

private Matrix prepareA(Matrix A, Vector B) {

Matrix a = new Matrix(A);

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

double newVal = 0;

if (i != j) {

newVal = A.get(i, j) / A.get(i, i);

newVal = -newVal;

}

a.set(i, j, newVal);

}

}

return a;

}

private Vector prepareB(Matrix A, Vector B) {

Vector b = new Vector(size);

for (int i = 0; i < size; i++) {

b.set(i, B.get(i) / A.get(i, i));

}

return b;

}

public void setEps(double eps) {

this.eps = eps;

}

public int getIterCount() {

return iterCount;

}

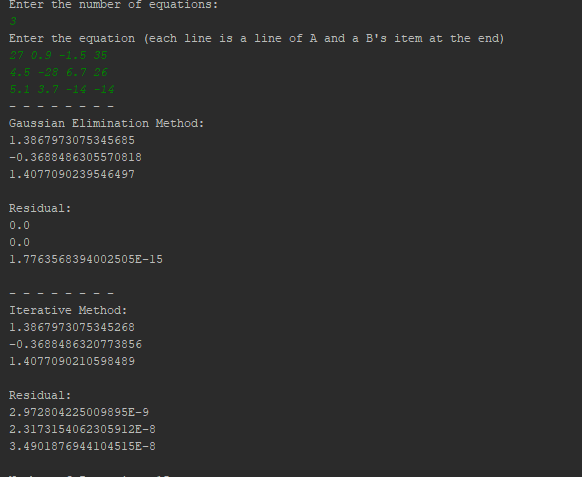
private void throwsIterationsOut() {

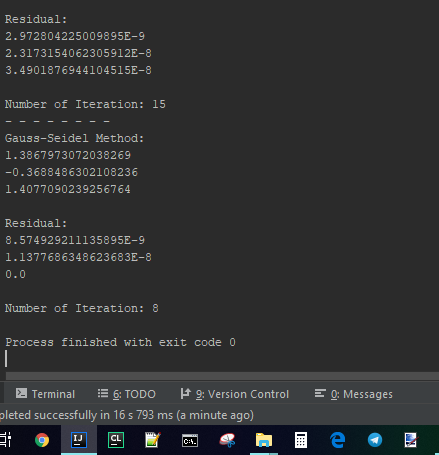
throw new RuntimeException("Invalid system (too many iterations)");

}

}

1. Результат виконання





**Висновок:** на цій лабораторній роботі навчився розв’язувати системи лінійних алгебраїчних рівнянь з застосуванням ЕОМ за ітераційними методами; набуття теоретичних знань та практичних навичок розв’язку систем лінійних алгебраїчних рівнянь з застосуванням ЕОМ за точними методами.

Я з’ясував, що найбільш ефективним методом (серед реалізованих) є метод Гауса-Зейделя. Йому потрібно приблизно у два разі менше ітерацій чим для методу ітерацій. А метод гауса хоча і точний, але має усі недоліки точних рішень під час рішень СЛАР – швидкий ріст часу виконання та зниження точності, при збільшені системи.