**ЗАВДАННЯ № .**

**Знаходження оберненої матриці методом Гаусса**

Знаходження оберненої матриці методом Гаусса - метод, який використовується для вирішення квадратних систем лінійних алгебраїчних рівнянь, знаходження зворотної матриці, знаходження координат вектора в заданому базисі або відшукання рангу матриці

Алгоритм :

1. Вибирають перший зліва стовпець, в якому є хоч одне відмінне від нуля значення.

2. Якщо саме верхнє число в цьому стовпці нуль, то змінюють всю перший рядок матриці з іншого рядком матриці, де в цій колонці немає нуля.

3. Всі елементи першого рядка ділять на верхній елемент вибраного стовпця.

4. З решти рядків віднімають перший рядок, помножену на перший елемент відповідного рядка, з метою отримати першим елементом кожного рядка (крім першої) нуль.

5. Далі проводять таку ж процедуру з матрицею, що виходить з вихідної матриці після викреслювання першого рядка і першого стовпця.

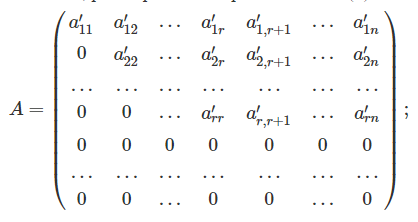
6. Після повторення цієї процедури раз отримують верхню трикутну матрицю

7. віднімається з передостанньої рядки останній рядок, помножену на відповідний коефіцієнт, з тим, щоб в передостанньому рядку залишилася тільки 1 на головній діагоналі.

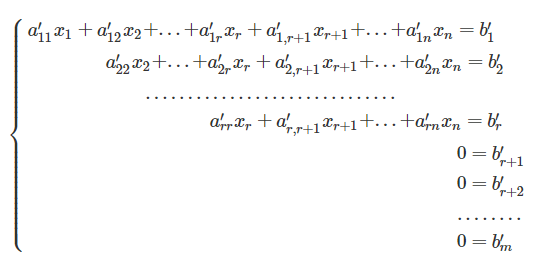
8. Повторюють попередній крок для наступних рядків. В результаті отримують одиничну матрицю і рішення на місці вільного вектора (з ним необхідно проводити всі ті ж перетворення).

**Прямий крок метода Гаусса:**

За допомогою елементарних перетворень над рядками і перестановкою стовпців, розширена матриця системи може бути приведена до вигляду



Матриця є розширеною матрицею системи яка еквівалентна вихідної системі.



КОД ПРОГРАМИ :

**public** **class** EquationsSet {

**public** **double**[][] Coefficients;

**public** **double**[] Solve() {

**this**.Simplify();

**this**.MakeNullsInUpperLines();

**double**[][] t = Coefficients;

**double**[] **Result** = **new** **double**[t.length];

**for** (**int** i = 0; i < **Result**.length; i++) {

**Result**[i] = t[i][t[i].length - 1];

        }

**return** **Result**;

    }

**public** **void** print() {

**for** (**int** i = 0; i < Coefficients.length; i++) {

**for** (**int** j = 0; j < Coefficients.length; j++) {

**System**.out.printf("%-7f ", Coefficients[i][j]);

            }

**System**.out.println(" = "

                    + Coefficients[i][Coefficients[i].length - 1]);

        }

 }

**public** **static** **void** main(**String**[] args) {

        EquationsSet S = **new** EquationsSet();

        {

**double**[][] t = {{4.2, 0.32, 0.11, 0.13, },

                            {0.17, 0.25, 0.48, 0.52},

                            {0.12, 0.08, 0.572, 0.61, },

                            {0.54, 0.13, 0.81, 0.17}};

            S.Coefficients = t;

        }

        S.print();

**System**.out.println();

**double**[] t = S.Solve();

**System**.out.println("Solution:");

**for** (**double** e : t) {

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

        }

    }

**private** **double**[][] Simplify() {

**int**[] notNull = FindFirstNotNull();

        Make1LineNotStartingWithNull(notNull[0], notNull[1]);

        DivideOn(0, Coefficients[notNull[0]][notNull[1]]);

        MakeNulls();

        EquationsSet es = **new** EquationsSet();

        es.Coefficients = GetTruncatedArray();

**if** (es.Coefficients.length > 1) {

            es.Simplify();

        } **else** {

            es.Coefficients[0][1] /= es.Coefficients[0][0];

            es.Coefficients[0][0] = 1;

        }

**this**.RIP(es);

**return** **this**.Coefficients;

    }

**private** **int**[] FindFirstNotNull() {

**int**[] **Result** = **new** **int**[2];

**Result**[0] = 0;

**Result**[1] = 0;

        find\_column: **for** (**Result**[0] = 0; **Result**[0] < Coefficients.length; **Result**[0]++) {

**for** (**Result**[1] = 0; **Result**[1] < Coefficients.length - 1; **Result**[1]++) {

**if** ((Coefficients[**Result**[1]][**Result**[0]]) != 0)

**break** find\_column;

            }

        }

**return** **Result**;

    }

**private** **void** Make1LineNotStartingWithNull(**int** i, **int** h) {

**if** (Coefficients[0][0] == 0) {

**int** j;

**for** (j = 1; j < Coefficients.length; j++) {

**if** ((Coefficients[j][0]) != 0)

**break**;

**else** **if** (j == Coefficients.length - 1) {

                            .println("Помилка: У першій колонці нема ненульових елементів...");

**System**.out.println("Коректна робота не гарантується");

                                  }

            }

**double**[] t = **new** **double**[Coefficients[0].length];

            t = Coefficients[0];

            Coefficients[0] = Coefficients[j];

            Coefficients[j] = t;

        }

    }

**private** **void** DivideOn(**int** lineNumber, **double** p) {

**for** (**int** j = 0; j < Coefficients[0].length; j++) {

            Coefficients[lineNumber][j] /= p;

        }

    }

**private** **void** MakeNulls() {

**for** (**int** k = 1; k < Coefficients.length; k++) {

**double** t = Coefficients[k][0];

**for** (**int** l = 0; l < Coefficients[0].length; l++)

                          Coefficients[k][l] -= Coefficients[0][l] \* t;

            }

        }

    }

**private** **double**[][] GetTruncatedArray() {

**double**[][] result = **new** **double**[Coefficients.length - 1][Coefficients[0].length - 1];

**for** (**int** k = 1; k < Coefficients.length; k++) {

**for** (**int** l = 1; l < Coefficients[0].length; l++) {

                result[k - 1][l - 1] = Coefficients[k][l];

            }

        }

**return** result;

    }

**private** **void** RIP(EquationsSet o) {

**for** (**int** i = 0; i < o.Coefficients.length; i++) {

**for** (**int** j = 0; j < o.Coefficients[i].length; j++) {

**int** line = **this**.Coefficients.length - 1 - i;

**int** column = **this**.Coefficients[line].length - 1 - j;

**int** parLine = o.Coefficients.length - i - 1;

**int** parCol = o.Coefficients[parLine].length - j - 1;

**this**.Coefficients[line][column] = o.Coefficients[parLine][parCol];              }            }        }

**private** **void** MakeNullsInUpperLines() {

**int** j = 0;

**for** (**int** i = Coefficients.length - 1; i >= 0; i--)

**for** (j = i - 1; j >= 0; j--) {

**int** t = Coefficients[i].length - 1;                                    Coefficients[j][t] -= Coefficients[j][i] \* Coefficients[i][t];                    Coefficients[j][i] = 0;                }            }       }    }

РІШЕННЯ СИСТЕМИ ЗА ДОПОМОГОЮ ПРОГРАМИ

Дані

{{4.2, 0.32, 0.11, 0.13, },

{0.17, 0.25, 0.48, 0.52},

{0.12, 0.08, 0.572, 0.61, },

{0.54, 0.13, 0.81, 0.17}};

Результат :

1 0 0 0 0.254 -0.384 0.291 -0.066

0 1 0 0 -0.221 5.202 -4.701 1.126

0 0 1 0 -0.172 -0.6 0.131 1.497

-0 -0 -0 1 0.182 0.102 2.044 -1.902

ПРОРАХУНОК МЕТОДА ПРОСТИХ ІТЕРАЦІЙ

Вихідна система

Ділимо рядок матриці 1 на 4.2

Множимо рядок 1 на -0.17 і складаємо з рядком 2

Множимо рядок 1 на -0.12 і складаємо з рядком 3

Множимо рядок 1 на -0.54 і складаємо з рядком 4

Ділимо її рядок 2 на 0.237

Множимо рядок 2 на -0.071 та складаємо з рядком 3

Множимо рядок 2 на -0.089 та складаємо з рядком 4

Ділимо її рядок 3 на 0.575

Множимо рядок 3 на -0.618 та складаємо з рядком 4

Ділимо її рядок 4 на -0.526

Множимо рядок 4 на -0.787 та складаємо з рядком 3

Множимо рядок 4 на -2.171 та складаємо з рядком 2

Множимо рядок 4 на -0.031 та складаємо з рядком

Множимо рядок 3 на -2.006 та складаємо з рядком 2

Множимо рядок 3 на -0.026 та складаємо з рядком 1

Множимо рядок 2 на -0.076 та складаємо з рядком 1

Перевірка

Перевірка успішна. результат

Література:

1. Самарский А.А., Гулин А.В. Численные методы: Учеб. Пособие для вузов М.: Наука. Гл. ред. физ-мат. лит., 1989. – 432 с.
2. https://ru.wikipedia.org/wiki/Метод\_Гаусса\_—\_Жордана