Белорусский государственный университет

Факультет прикладной математики и информатики

Лабораторная работа №1

Решение СЛАУ методом Гаусса
(выбор главного элемента по матрице)

Выполнил:

Студент 2 курса 7 группы ПМ ФПМИ

Шевцов Евгений

Преподаватель:

Будник Анатолий Михайлович

Минск – 2021

Описание метода нахождения решения СЛАУ методом Гаусса с выбором главного элемента по матрице

1. Прямой ход.

Будем рассматривать СЛАУ, матрица системы которой квадратная и невырожденная.

Система линейных алгебраических уравнений выглядит следующим образом:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ a_{n2}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$, отметим её как систему (1).

Пусть без ограничения общности элемент a_{11} является максимальным по модулю ненулевым элементом в матрице (в противном случае, если максимальным по модулю ненулевым является элемент a_{ij}, то меняем 1-ю и i-ю строки и 1-й и j-й столбец местами, запоминая количество перестановок, а так же перенумерацию переменных).

Далее исключаем x_1 из всех уравнений системы, кроме первого, добавляя к i-му уравнению первое, домноженное на $\frac{-a_{i1}}{a_{11}}\ \forall i = [2; n]$. В итоге из системы (1) получаем равносильную ей систему: $$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{22,1}x_2 + \cdots + a_{2n,1}x_n = b_{2,1} \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ a_{n2,1}x_2 + \cdots + a_{nn,1}x_n = b_{n,1} \end{cases}$$ (2), где в индексе после запятой указан номер шага.

По аналогичному рассуждению, исключая неизвестные из уравнений системы (2) получим равносильную ей систему следующего вида: $$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{22,1}x_2 + \cdots + a_{2n,1}x_n = b_{2,1} \\ a_{33,2}x_3 + \cdots + a_{3n,2}x_n = b_{3,2} \\ a_{44,3}x_4 + \cdots + a_{4n,3}x_n = b_{4,3} \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ a_{nn,n-1}x_n = b_{n,n-1} \end{cases}$$ (3).

Распишем общие формулы для $a_{ij,k}$ и $b_{i,k}$ на k-м шаге:

$a_{ij,k} = a_{ij,k-1} - \frac{a_{ik,k-1}}{a_{kk,k-1}} a_{kj,k-1}$; $b_{i,k} = b_{i,k-1} - \frac{a_{ik,k-1}}{a_{kk,k-1}} b_{k,k-1}$, где k = $\overline{2, n}$, а i, j = $\overline{k+1, n}$.

Таким образом, прямым ходом Гаусса мы привели искомую систему (1) к системе (3), матрица которой верхне-треугольная, и можем приступить к нахождению неизвестных переменных обратным ходом Гаусса.

2. Обратный ход.

Рассмотрим систему (3). Из последнего уравнения заметим, что $x_n = \frac{b_{n,n-1}}{a_{nn,n-1}}$, из предпоследнего $x_{n-1} = \frac{b_{n-1,n-2} - a_{n-1\,n,n-2}x_n}{a_{n-1\,n-1,n-2}}$ и т.д. В итоге получаем формулу для нахождения k-го неизвестного: $x_k = \frac{b_{k,k-1} - \sum_{j=k+1}^{n} a_{kj,k-1}x_j}{a_{kk,k-1}}$, где k = $\overline{1, n}$, j = $\overline{k+1, n}$.

Не забывая о перенумерации переменных, расставим их по своим местам и получим итоговый вектор значений x_i – решение СЛАУ.

3. Нахождение обратной матрицы

Для нахождения обратной матрицы достаточно решить методом Гаусса матричное уравнение вида AX = E т.к. $X = A^{-1}$, т.е. систему с расширенной матрицей системы вида:

$$\begin{pmatrix} a_{11} & \dots & a_{1n} & | & 1 & 0 & \dots & 0 \\ a_{12} & \dots. & a_{2n} & | & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & | & \dots & \dots & \dots & \dots \\ a_{1n} & \dots & a_{nn} & | & 0 & \dots & 0 & 1 \end{pmatrix}$$

При этом матрицей системы будет являться сама матрица A, а матрицей неоднородности – единичная матрица E.

4. Нахождение определителя матрицы

Рассмотрим матрицу системы (3): она является верхне-треугольной, а её определитель отличается от определителя исходной матрицы системы на (-1) в степени количества перестановок строк и столбцов, которые мы меняли для нахождения главного элемента, т.е. может отличаться только знаком. А определитель матрицы системы (3) равен произведению диагональных элементов этой матрицы. Следовательно, получаем формулу для нахождения определителя матрицы:

$$|A| = (-1)^{(\text{количество перестановок})} * a_{11} * a_{22,1} * a_{33,2} * \dots * a_{nn,n-1}$$

5. Вычисление невязки

Для проверки точности вычислений находят величину невязки. В случае с решением СЛАУ методом Гаусса, нашей невязкой будет являться вектор, а значения этого вектора будут очень малы, т.к. данный метод является точным, а погрешность может быть только лишь в хранении типов данных. В случае нахождения обратной матрицы наша невязка будет являться матрицей.

Формула вычисления невязки:

$\Gamma = AX - B$ (невязка для решения СЛАУ); $M = AA^{-1} - E$ (невязка для обратной матрицы)

6. Вычисление числа обусловленности

Для вычисления числа обусловленности найдём октаэдрические нормы матриц A и A^{-1} по формуле$||A|| = \sum_{i,j=1}^{n} |a_{ij}|$.
Число обусловленности найдём по формуле: $\vartheta(A) = ||A|| * ||A^{-1}||$.

Листинг

```
#include <iostream>
#include <iomanip>
#include <vector>

int searchMaxI(std::vector<std::vector<double>> M, int startI, int startJ) {
    int maxI = startI;
    int maxJ = startJ;
    for (int i = startI; i < M.size(); ++i) {
        for (int j = startJ; j < M.size(); ++j) {
            if (abs(M[i][j]) > abs(M[maxI][maxJ])) {
                maxI = i;
                maxJ = j;
            }
        }
    }

    return maxI;
}

int searchMaxJ(std::vector<std::vector<double>> M, int startI, int startJ) {
    int maxI = startI;
    int maxJ = startJ;
    for (int i = startI; i < M.size(); ++i) {
        for (int j = startJ; j < M.size(); ++j) {
            if (abs(M[i][j]) > abs(M[maxI][maxJ])) {
                maxI = i;
                maxJ = j;
            }
        }
    }

    return maxJ;
}

int main() {
    //Исходные данные
    std::vector<std::vector<double>> systemM =
      { {0.7941, 0.0000, -0.2067, 0.1454, 0.2423},
        {-0.0485, 0.5168, 0.0000, -0.0985, 0.0323},
        {0.0162, -0.1454, 0.9367, 0.0178, 0.0565},
        {0.0485, 0.0000, -0.1179, 0.9367, 0.0000},
        {0.0323, -0.0485, 0.2342, -0.0194, 0.6783} };

    std::vector<double> columnN = { 1.5569, 2.0656, -2.9054, -8.0282, 3.4819 };

    std::vector<std::vector<double>> extendedSystemMat =
    { {0.7941, 0.0000, -0.2067, 0.1454, 0.2423, 1.5569},
      {-0.0485, 0.5168, 0.0000, -0.0985, 0.0323, 2.0656},
      {0.0162, -0.1454, 0.9367, 0.0178, 0.0565, -2.9054},
      {0.0485, 0.0000, -0.1179, 0.9367, 0.0000, -8.0282},
      {0.0323, -0.0485, 0.2342, -0.0194, 0.6783, 3.4819} };

    //Для обратной матрицы
    std::vector<std::vector<double>> inverseMatrix =
    { {1, 0, 0, 0, 0},
      {0, 1, 0, 0, 0},
      {0, 0, 1, 0, 0},
      {0, 0, 0, 1, 0},
      {0, 0, 0, 0, 1} };

    std::vector<int> permutations = { 0, 1, 2, 3, 4 };
    std::vector<int> inverseRowPermutations;
    std::vector<int> inverseColPermutations = { 0, 1, 2, 3, 4 };
```

```cpp
    std::vector<double> answer = { 0, 0, 0, 0, 0 };

    double det = 1;

    //Прямой ход
    for (int k = 0; k < systemM.size(); ++k) {
        int maxI = searchMaxI(extendedSystemMat, k, k);
        int maxJ = searchMaxJ(extendedSystemMat, k, k);

        std::swap(permutations[k], permutations[maxJ]);
        std::swap(inverseColPermutations[k], inverseColPermutations[maxI]);

        //Найдя максимальный, свапаем строку, потом столбец
        for (int j = k; j < systemM.size() + 1; ++j) {
            std::swap(extendedSystemMat[k][j], extendedSystemMat[maxI][j]);
        }
        for (int i = 0; i < systemM.size(); ++i) {
            std::swap(extendedSystemMat[i][k], extendedSystemMat[i][maxJ]);
        }

        //Так же меняем строки и столцы для подсчёта обратной матрицы
        for (int j = 0; j < systemM.size(); ++j) {
            std::swap(inverseMatrix[k][j], inverseMatrix[maxI][j]);
        }
        for (int i = 0; i < systemM.size(); ++i) {
            std::swap(inverseMatrix[i][k], inverseMatrix[i][maxJ]);
        }

        //Параллельно считаем определитель
        det *= extendedSystemMat[k][k] * pow(-1, maxI + maxJ - 2 * k);

        //Для обратной матрицы
        for (int i = k + 1; i < systemM.size(); ++i) {
            for (int j = 0; j < systemM.size(); ++j) {
                inverseMatrix[i][j] -= inverseMatrix[k][j] * extendedSystemMat[i][k] /
extendedSystemMat[k][k];
            }
        }

        //Далее k-й шаг метода, не трогая k-й столбец
        for (int i = k + 1; i < systemM.size(); ++i) {
            for (int j = k + 1;  j < systemM.size() + 1; ++j) {
                extendedSystemMat[i][j] -= extendedSystemMat[i][k] *
extendedSystemMat[k][j] / extendedSystemMat[k][k];
            }
        }

        //Зануляем k-й столбец за исключением Х k-й
        for (int i = k + 1; i < systemM.size(); ++i) {
            extendedSystemMat[i][k] = 0;
        }

    } //Прямой ход метода Гаусса завершен, переход к обратному ходу
    inverseRowPermutations = permutations;

    for (int k = systemM.size() - 1; k >= 0; --k) {
        double sum = 0;
        for (int j = k + 1; j < systemM.size(); ++j) {
            sum += extendedSystemMat[k][j] * answer[j];
        }
        answer[k] = (extendedSystemMat[k][systemM.size()] - sum) /
extendedSystemMat[k][k];
    }

    //Расставим переменные по своим местам
```

```cpp
int num = 0;
for (int i = 0; i < answer.size(); ++i) {
    if (permutations[i] == num) {
        std::swap(answer[num], answer[i]);
        std::swap(permutations[num], permutations[i]);
        i = num;
        ++num;
    }
}//Обратный ход метода Гаусса завершён

//Посчитаем невязку
std::vector<double> discrepancy = { 0, 0, 0, 0, 0 };
for (int i = 0; i < systemM.size(); ++i) {
    for (int j = 0; j < systemM.size(); ++j) {
        discrepancy[i] += systemM[i][j] * answer[j];
    }
    discrepancy[i] -= columnN[i];
}

//Досчитаем обратную матрицу
for (int k = systemM.size() - 1; k >= 0; --k) {
    for (int j = 0; j < systemM.size(); ++j) {
        inverseMatrix[k][j] /= extendedSystemMat[k][k];
    }
    for (int i = 0; i < k; ++i) {
        for (int j = 0; j < systemM.size(); ++j) {
            inverseMatrix[i][j] -= inverseMatrix[k][j] * extendedSystemMat[i][k];
        }
    }
}

//Вернём строки и столбцы обратной матрицы на своё место
num = 0;
for (int k = 0; k < inverseRowPermutations.size(); ++k) {
    if (inverseRowPermutations[k] == num) {
        for (int j = 0; j < systemM.size(); ++j) {
            std::swap(inverseMatrix[num][j], inverseMatrix[k][j]);
        }
        std::swap(inverseRowPermutations[num], inverseRowPermutations[k]);
        k = num;
        ++num;
    }
}
num = 0;
for (int k = 0; k < inverseColPermutations.size(); ++k) {
    if (inverseColPermutations[k] == num) {
        for (int i = 0; i < systemM.size(); ++i) {
            std::swap(inverseMatrix[i][num], inverseMatrix[i][k]);
        }
        std::swap(inverseColPermutations[num], inverseColPermutations[k]);
        k = num;
        ++num;
    }
}

//Невязка для обратной матрицы
std::vector<std::vector<double>> neuralMatrix =
{ {0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0} };
for (int i = 0; i < systemM.size(); ++i) {
    for (int j = 0; j < systemM.size(); ++j) {
        double sum = 0;
```

```
            for (int k = 0; k < systemM.size(); ++k) {
                sum += systemM[i][k] * inverseMatrix[k][j];
            }
            neuralMatrix[i][j] = sum;
            if (i == j) {
                neuralMatrix[i][j] -= 1;
            }
        }
    }

    //Найдём октаэдрические нормы матриц и число обусловленности
    double normSystemM = 0;
    for (int i = 0; i < systemM.size(); ++i) {
        for (int j = 0; j < systemM.size(); ++j) {
            normSystemM += abs(systemM[i][j]);
        }
    }

    double normInverseM = 0;
    for (int i = 0; i < inverseMatrix.size(); ++i) {
        for (int j = 0; j < inverseMatrix.size(); ++j) {
            normInverseM += abs(inverseMatrix[i][j]);
        }
    }

    double conditionNumber = normInverseM * normSystemM;

}
```

Выходные данные

```
==================================================Resulting matrix================================================

        0.9367         0.0178         0.0162         0.0565        -0.1454  |         -2.9054
             0        0.93894      0.0505391     0.00711151     -0.0183011  |         -8.3939
             0              0       0.789637       0.253637     -0.0291746  |         2.25072
             0              0              0       0.654868     -0.0115198  |         3.91093
             0              0              0              0       0.514109  |         1.02794

==================================================Decision vector=================================================

                               ( 0.994628, 1.99946, -2.9999, -8.99982, 6.00726)

==================================================Neural vector===================================================

                                           ( -4.44089e-16, 0, 0, 0, 0)

==================================================Determinant=====================================================

                                                    0.233817

==================================================Inverce Matrix====================================================

       1.28824      0.0608751       0.381435      -0.211064      -0.494853
      0.111622        1.94511      0.0824809        0.18276      -0.139368
   -0.00035829       0.298161        1.09488      0.0084231       -0.10527
    -0.0667471      0.0343768        0.11806        1.07957      0.0123722
     -0.055149      0.0342166      -0.386924      0.0510868        1.52457

==================================================Neural Matrix===================================================

  -2.22045e-16   -8.67362e-18              0    1.38778e-17              0
   4.98733e-18   -2.22045e-16    1.73472e-18              0              0
             0    9.97466e-18              0              0    1.38778e-17
   1.38778e-17   -1.38778e-17              0              0              0
  -1.38778e-17    6.93889e-18              0   -1.38778e-17   -1.11022e-16

==================================================Condition number==================================================
                                                       52.4833
```

Вывод

Метод Гаусса является точным методом решения СЛАУ, погрешность которого возникает лишь в хранении переменных с плавающей точкой. Использование выбора главного элемента по матрице позволяет минимизировать погрешность, используя наибольшие элементы по модулю для элементарных преобразований матрицы.