|  |  |
| --- | --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ПЕРМСКОГО КРАЯ | |
| Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования  «Пермский государственный национальный исследовательский университет» | |
| **ЧИСЛЕННЫЕ МЕТОДЫ  «ПРЯМЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ»** | |
| *Лабораторная работа №2* | |
|  | |
|  | Работу выполнили студенты группы ПМИ-1-2015 3 курса механико-математического факультета  Толов Я.Ф., Заманов М.Р. |
|  | Проверил:  Профессор, доктор физико-математических наук  С. В. Русаков  “\_\_\_\_” 20\_\_ г. |
|  |  |
| Пермь 2017 | |

*Задание*

1. Решить заданную систему линейных алгебраических уравнений методом *LU* разложения с выбором главного элемента по столбцу.
2. Используя полученное ранее *LU* разложение, вычислить обратную матрицу к исходной.
3. Вычислить определитель и найти число обусловленности исходной матрицы.

*Исходные данные*

Варианты исходных данных: 14 a) и 20 a).

Дана матрица А.

Вариант 14:

a)┌ ┐ bт = [34.1, -35.2, 11.2, 1.3]

│-2.0 3.6 6.7 2.2│

│-3.2 0.2 -8.0 -2.1│

│-0.5 9.9 -8.3 4.2│

│-6.4 -1.9 8.9 -3.8│

└ ┘

Вариант 20:

a)┌ ┐ bт = [-56.5, -27.5, -49.1, -26.3]

│ 6.0 -3.1 -9.7 -6.8│

│ 4.0 9.4 -5.7 -8.3│

│ 8.1 -5.5 -3.8 -8.7│

│ 3.0 -9.5 -6.1 2.0│

└ ┘

Дан вектор X.



***Решение***

Представление матрицы A в виде произведения двух матриц P\*A=L\*U, где L – нижняя треугольная матрица, а U – верхняя треугольная матрица, P – матрица перестановок, получаемая из единичной матрицы путём перестановки строк и столбцов.

┌ ┐ ┌ ┐

│ l11 00 0│ │ 1 u12 u13 u14│ ┌ ┐

L = │ l21 l22 0 0│ U = │ 0 1u23 u24│ │ 0 1 0│

│ l31 l32 l33  0│ │ 0 01 u34│ P = │ 1 0 0│

│ l41 l42 l43 l44│ │ 0 00 1│ │ 0 0 1│

└ ┘ └ ┘ └ ┘

Метод обращения матрицы находит массив x, который при умножении на матрицу LU дает массив b. В данном методе два цикла. Первый цикл использует прямую подстановку в нижней части матрицы LU, а второй цикл — обратную подстановку в верхней части той же матрицы.

= |L| = l11\*l22\*l33\*l44.  
Так как единичной матрицы |P| = 1, определитель матрицы |U| так же равен 1.

Число обусловленности – норма A \* норма A^(-1).

Для нахождения числа обусловленности исходной матрицы необходимо сложить модули всех элементов строки и выбрать максимальное значение.

|A||∞=max1<=i<=n()

***Краткие выводы***

Данная лабораторная работа научила нас решать системы линейных алгебраических уравнений новым методом. Также была написана программа, которая решает СЛАУ посредством LUP разложения.

***Код на C++***

|  |
| --- |
| #include <iostream> |
|  | #include <iomanip> |
|  |  |
|  | using namespace std; |
|  | void fillLU(double\*\*, double\*\*, int); |
|  | void print(double\*\*, int); |
|  | double\*\* mult(double\*\*, double\*\*, int); |
|  | double det(double\*\*, int); |
|  | double\*\* trans(double\*\*, int, int); |
|  | double\*\* init(int, int, bool=false); |
|  | void SwapRows(double\*\*, int, int); |
|  | void LUP(double\*\*, double\*\*, int\*, int); |
|  | double\* Solve(double\*\*, double\*, int); |
|  | double\*\* GetInvert(double\*\*, int); |
|  | void del(double\*\*, int); |
|  | double norm(double\*\*, int); |
|  | void printLU(double\*\*, int); |
|  | bool flag = false; |
|  | int main() |
|  | { |
|  | int n = 0; |
|  | cout << "Input n:"; |
|  | cin >> n; |
|  | double\*\* A = init(n, n); |
|  | double\* b = new double[n]; |
|  |  |
|  | cout << "Input matrix" << endl; |
|  | for (int i = 0; i < n; i++) |
|  | for (int j = 0; j < n; j++) |
|  | cin >> A[i][j]; |
|  |  |
|  | cout << "Input b" << endl; |
|  | for (int i = 0; i < n; i++) |
|  | cin >> b[i]; |
|  | flag = true; |
|  | double\* x = Solve(A, b, n); |
|  | flag = false; |
|  | cout << "x: "; |
|  | for (int i = 0; i < n; i++) |
|  | cout << x[i] << " "; |
|  | cout << endl; |
|  | double\*\* InvA = GetInvert(A, n); |
|  | cout << "Inverse matrix" << endl; |
|  | print(InvA, n); |
|  | double conditionNumber = norm(A, n) \* norm(InvA, n); |
|  | cout << "condition number = " << conditionNumber << endl; |
|  |  |
|  | double deter = det(A, n); |
|  | cout << "det = " << deter << endl; |
|  | cin.get(); |
|  | cin.get(); |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | void fillLU(double\*\* LU, double\*\* A, int n) |
|  | { |
|  | LU[0][0] = A[0][0]; |
|  | for (int j = 1; j < n; j++) |
|  | { |
|  | LU[j][0] = A[j][0]; |
|  | LU[0][j] = A[0][j] / LU[0][0]; |
|  | } |
|  |  |
|  | for (int i = 1; i < n; i++) |
|  | for (int j = i; j < n; j++) |
|  | { |
|  | double sum = 0; |
|  | for (int k = 0; k < i; k++) |
|  | sum += LU[j][k] \* LU[k][i]; |
|  | LU[j][i] = A[j][i] - sum; |
|  | if (i != j) |
|  | { |
|  | sum = 0; |
|  | for (int k = 0; k < i; k++) |
|  | sum += LU[i][k] \* LU[k][j]; |
|  | LU[i][j] = (A[i][j] - sum) / LU[i][i]; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void print(double\*\* M, int n) |
|  | { |
|  | cout.setf(ios::left); |
|  | for (int i = 0; i < n; i++, cout << endl) |
|  | for (int j = 0; j < n; j++) |
|  | cout << setw(8) << setprecision(5) << M[i][j] << " "; |
|  | cout.unsetf(ios::left); |
|  | } |
|  |  |
|  | void printLU(double\*\* LU, int n) |
|  | { |
|  | cout.setf(ios::left); |
|  | double\*\* L = init(n, n, true); |
|  | double\*\* U = init(n, n, true); |
|  | for(int i=0;i<n;i++) |
|  | for (int j = 0; j < n; j++) |
|  | { |
|  | if (i <= j) |
|  | L[j][i] = LU[i][j]; |
|  | if (i > j) |
|  | U[j][i] = LU[i][j]; |
|  | if (i == j) |
|  | U[i][i] = 1; |
|  | } |
|  | cout << "L" << endl; |
|  | print(L, n); |
|  | cout << "U" << endl; |
|  | print(U, n); |
|  | del(L, n); |
|  | del(U, n); |
|  | cout.unsetf(ios::left); |
|  | } |
|  |  |
|  | double\*\* mult(double\*\* A, double\*\* B, int n) |
|  | { |
|  | double\*\* C = new double\*[n]; |
|  | for (int i = 0; i < n; i++) |
|  | C[i] = new double[n]; |
|  | for(int i=0;i<n;i++) |
|  | for (int j = 0; j < n; j++) |
|  | { |
|  | double sum = 0; |
|  | for (int k = 0; k < n; k++) |
|  | sum += A[i][k] \* B[k][j]; |
|  | C[i][j] = sum; |
|  | } |
|  | return C; |
|  | } |
|  |  |
|  |  |
|  | double\*\* trans(double\*\* A, int n,int m) |
|  | { |
|  | double\*\* B = new double\*[m]; |
|  | for (int i = 0; i < m; i++) |
|  | B[i] = new double[n]; |
|  | for (int i = 0; i < n; i++) |
|  | for (int j = 0; j < m; j++) |
|  | B[j][i] = A[i][j]; |
|  | return B; |
|  | } |
|  | double\*\* init(int n, int m, bool fillByZero) |
|  | { |
|  | double\*\* A = new double\*[n]; |
|  | for (int i = 0; i < n; i++) |
|  | A[i] = new double[m]; |
|  | if (fillByZero) |
|  | for (int i = 0; i < n; i++) |
|  | for (int j = 0; j < n; j++) |
|  | A[i][j] = 0.0; |
|  | return A; |
|  | } |
|  |  |
|  | void del(double\*\* M, int n) |
|  | { |
|  | for (int i = 0; i < n; i++) |
|  | delete[] M[i]; |
|  | delete[] M; |
|  | } |
|  |  |
|  | double det(double\*\* M, int n) |
|  | { |
|  | double det = 0; |
|  | for (int p = 0; p < n; p++) |
|  | { |
|  | double mulP = 1; |
|  | double mulM = 1; |
|  | for (int k = 0, i = n - 1 - p, j = p; k < n; k++) |
|  | { |
|  | mulP \*= M[k][j]; |
|  | mulM \*= M[k][i]; |
|  | if (j == n - 1) j = 0; |
|  | else j++; |
|  | if (i == 0) i = n - 1; |
|  | else i--; |
|  | } |
|  | det = det + mulP - mulM; |
|  | } |
|  | return det; |
|  | } |
|  |  |
|  | double\* Solve(double\*\* A, double\* b, int n) |
|  | { |
|  | double\*\* LU = init(n, n);// |
|  | int\* P = new int[n]; |
|  | for (int i = 0; i < n; i++) |
|  | P[i] = i; |
|  | LUP(A, LU, P, n); |
|  |  |
|  | double\* y = new double[n]; |
|  | double\* x = new double[n]; |
|  | double s = 0; |
|  | for (int i = 0; i < n; i++) |
|  | { |
|  | s = 0; |
|  | for (int j = 0; j < i; j++) |
|  | s += LU[i][j] \* y[j]; |
|  | y[i] = b[P[i]] - s; |
|  | } |
|  | for (int i = n - 1; i >= 0; i--) |
|  | { |
|  | s = 0; |
|  | for (int j = i + 1; j < n; j++) |
|  | s += LU[i][j] \* x[j]; |
|  | x[i] = (y[i] - s) / LU[i][i]; |
|  | } |
|  | delete[] y; |
|  | delete[] P; |
|  | del(LU, n); |
|  | return x; |
|  | } |
|  |  |
|  | void LUP(double\*\* A, double\*\* C, int\* P, int n) |
|  | { |
|  | for (int i = 0; i < n; i++) //C = A; |
|  | for (int j = 0; j < n; j++) |
|  | C[i][j] = A[i][j]; |
|  | int rang = 0; |
|  | for (int i = 0; i < n; i++) |
|  | { |
|  |  |
|  | // |
|  | double pivotValue = 0; |
|  | int pivot = -1; |
|  | for (int row = i; row < n; row++) |
|  | { |
|  | if (fabs(C[row][i]) > pivotValue) |
|  | { |
|  | pivotValue = fabs(C[row][i]); |
|  | pivot = row; |
|  | } |
|  | } |
|  | rang++; |
|  | if (pivotValue != 0) |
|  | { |
|  | // |
|  | int t = P[pivot]; P[pivot] = P[i]; P[i] = t; |
|  | SwapRows(C,pivot, i); |
|  | for (int j = i + 1; j < n; j++) |
|  | { |
|  | C[j][i] /= C[i][i]; |
|  | for (int k = i + 1; k < n; k++) |
|  | C[j][k] -= C[j][i] \* C[i][k]; |
|  | } |
|  | } |
|  | if (flag) |
|  | { |
|  | printLU(C, n); |
|  | cout << "rang A = " << rang << endl << endl; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void SwapRows(double\*\* M, int a, int b) |
|  | { |
|  | double\* t= M[a]; |
|  | M[a] = M[b]; |
|  | M[b] = t; |
|  | } |
|  |  |
|  | double\*\* GetInvert(double\*\* A, int n) |
|  | { |
|  | double\*\* Invert = init(n, n); |
|  | double\* E = new double[n]; |
|  | for (int i = 0; i < n; i++) |
|  | E[i] = 0; |
|  |  |
|  | for (int i = 0; i < n; i++) |
|  | { |
|  | E[i]++; |
|  | Invert[i] = Solve(A, E, n); |
|  | E[i]--; |
|  | } |
|  | delete[] E; |
|  | return Invert; |
|  | } |
|  |  |
|  | double norm(double\*\* M, int n) |
|  | { |
|  | double sum = 0; |
|  | for (int i = 0; i < n; i++) |
|  | sum += fabs(M[0][i]); |
|  | double norm = sum; |
|  |  |
|  | for (int i = 1; i < n; i++) |
|  | { |
|  | sum = 0; |
|  | for (int j = 0; j < n; j++) |
|  | sum += fabs(M[i][j]); |
|  | if (sum > norm) |
|  | norm = sum; |
|  | } |
|  | return norm; |
|  | } |