|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Министерство науки и высшего образования РФ | | | | | | | | |  | |
|  | | | | |  | | |  | | | | |
|  | | | ФГБОУ ВО «Пермский государственный национальный исследовательский университет» | | | | | | |  | | |
|  | | | | |  | | |  | | | | |
|  | | | | **Численные методы**  Лабораторная работа №2  Варианты: №21b, №25b | | | | |  | | | |
|  | | | | |  | | |  | | | | |
|  | Работу выполнили  Студенты гр.ПМИ-4:  Колесников А.С  Пухов Н.А. | | | | |  | Проверил  профессор, доктор физико-математических наук  Русаков С.В  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2020 г. | | | | |  | |
|  | | | | |  | | |  | | | | |
|  | | | | | Пермь 2020 | | |  | | | | |

СОДЕРЖАНИЕ

[Задание 3](#_Toc52827451)

[Исходные данные 4](#_Toc52827452)

[Теоретическая справка 5](#_Toc52827453)

[Решение 8](#_Toc52827454)

[Тестирование 10](#_Toc52827455)

[Краткие выводы 16](#_Toc52827456)

[Текст программы 17](#_Toc52827457)

Задание

1. Решить заданную систему линейных алгебраических уравнений 

методом *LU* разложения с выбором главного элемента по столбцу.

1. Используя полученное ранее *LU* разложение вычислить обратную матрицу к исходной.
2. Вычислить определитель и найти число обусловленности исходной матрицы в кубической, октаэдрической и евклидовой нормах.

*Замечание.*

Для определения собственных значений матрицы (евклидова норма) воспользоваться можно воспользоваться итерационным методом вращений для матрицы *ATA*.

Исходные данные

Вариант №21(b):



Вариант №25(b):



Теоретическая справка





*Элементарной матрицей перестановок*  называется матрица, полученная из единичной матрицы перестановкой *k*-ой и *l*-ой строк. *P* − матрица перестановок, фиксирующая перестановки строк в методе Гаусса с выбором главного элемента, знак определяется числом перестановок.





Число обусловленности матрицы A:





Нормы матриц:

1. Кубическая норма: 
2. Октаэдрическая норма: 
3. 

Матрица A\* называется эрмитово сопряженной с матрицей А, если для  имеем .

Для нахождения собственных значений матрицы использовался итерационный метод вращений.

**Итерационный метод вращений (метод Якоби приведения матрицы к диагональному виду)**

Пусть A=AT>0

Задача: найти матрицу Т такую, что  - диагональная матрица.

Строим последовательность: ,



где  - матрица простого вращения.

С помощью матрицы  исключаем , где







Положим





Решение

Исходная матрица А приводится к произведению двух матриц A = L\*U, где L – нижняя треугольная матрица, а U – верхняя треугольная матрица с единичной диагональю:



Для решения задачи используется пять матриц  при этом  или  . На момент начала матрица  равна матрица  , матрица перестановок  и матрица  - единичные. К матрицам применяется метод исключения: находится максимальный по модулю элемент по столбцу матрицы  и меняются местами строки в матрицах  и  (в случае, если максимальный элемент не совпадает с текущим).

Зная матрицы  и  - решение СЛАУ – находится по формуле:



Вычисление обратной матрицы

Обратная матрица будет найдена с помощью набора из n векторов вида  , (единица – на i-ой позиции).

Находим решение СЛАУ, подставляя в качестве вектора правых частей вектор  . Полученные n векторов решений будут являться столбцами обратной к А матрицы.

Вычисление определителя и нахождение числа обусловленности. В силу равенства  (или  ).

Эвклидова норма матрицы – квадратный корень из максимального собственного значения произведения эрмитово сопряженной матрицы на исходную:

Тестирование

Вариант 21b:

A:

0.1000000 -8.3000000 7.1000000 5.5000000

1.2000000 5.2000000 -9.1000000 -0.2000000

-7.9000000 9.6000000 0.9000000 -1.2000000

3.8000000 -4.7000000 -0.2000000 7.9000000

vector b:

26.8000000 -16.5000000 9.2000000 25.4000000

k = 1 m = 3

U:

1.0000000 -1.2151899 -0.1139241 0.1518987

0.0000000 6.6582278 -8.9632911 -0.3822785

0.0000000 -8.1784810 7.1113924 5.4848101

0.0000000 -0.0822785 0.2329114 7.3227848

L:

-7.9000000 0.0000000 0.0000000 0.0000000

1.2000000 0.0000000 0.0000000 0.0000000

0.1000000 0.0000000 0.0000000 0.0000000

3.8000000 0.0000000 0.0000000 0.0000000

k = 2 m = 3

U:

1.0000000 -1.2151899 -0.1139241 0.1518987

0.0000000 1.0000000 -0.8695248 -0.6706392

0.0000000 0.0000000 -3.1737966 4.0829902

0.0000000 0.0000000 0.1613682 7.2676056

L:

-7.9000000 0.0000000 0.0000000 0.0000000

0.1000000 -8.1784810 0.0000000 0.0000000

1.2000000 6.6582278 0.0000000 0.0000000

3.8000000 -0.0822785 0.0000000 0.0000000

k = 3 m = 3

U:

1.0000000 -1.2151899 -0.1139241 0.1518987

0.0000000 1.0000000 -0.8695248 -0.6706392

0.0000000 0.0000000 1.0000000 -1.2864688

0.0000000 0.0000000 0.0000000 7.4752008

L:

-7.9000000 0.0000000 0.0000000 0.0000000

0.1000000 -8.1784810 0.0000000 0.0000000

1.2000000 6.6582278 -3.1737966 0.0000000

3.8000000 -0.0822785 0.1613682 0.0000000

k = 4 m = 4

U:

1.0000000 -1.2151899 -0.1139241 0.1518987

0.0000000 1.0000000 -0.8695248 -0.6706392

0.0000000 0.0000000 1.0000000 -1.2864688

0.0000000 0.0000000 0.0000000 1.0000000

L:

-7.9000000 0.0000000 0.0000000 0.0000000

0.1000000 -8.1784810 0.0000000 0.0000000

1.2000000 6.6582278 -3.1737966 0.0000000

3.8000000 -0.0822785 0.1613682 7.4752008

Rank = 4

LU-PA:

0.0000000000e+00 0.0000000000e+00 0.0000000000e+00 0.0000000000e+00

0.0000000000e+00 0.0000000000e+00 0.0000000000e+00 0.0000000000e+00

0.0000000000e+00 0.0000000000e+00 1.7763568394e-15 -1.6653345369e-16

0.0000000000e+00 0.0000000000e+00 0.0000000000e+00 0.0000000000e+00

vector x:

1.0000000 2.0000000 3.0000000 4.0000000

Ax – b:

0.0000000000e+00 3.5527136788e-15 3.5527136788e-15 0.0000000000e+00

x\* – x:

4.4408920985e-15 3.1086244690e-15 2.6645352591e-15 0.0000000000e+00

A-1:

-0.4397552 -0.3540682 -0.0463644 0.2901523

-0.3378156 -0.2618000 0.0710914 0.2393589

-0.2511193 -0.3063299 0.0330716 0.1720982

0.0041915 0.0068017 0.0654340 0.1337757

A\*A-1:

1.0000000000e+00 4.1633363423e-17 -1.6653345369e-16 2.2204460493e-16

2.7766417637e-16 1.0000000000e+00 3.4694469520e-17 2.0816681712e-17

-5.4643789493e-17 -6.9388939039e-17 1.0000000000e+00 3.0531133177e-16

-7.6327832943e-17 -1.1102230246e-16 -1.1102230246e-16 1.0000000000e+00

(A\*A-1) – E:

4.4408920985e-16 4.1633363423e-17 -1.6653345369e-16 2.2204460493e-16

2.7766417637e-16 2.2204460493e-16 3.4694469520e-17 2.0816681712e-17

-5.4643789493e-17 -6.9388939039e-17 0.0000000000e+00 3.0531133177e-16

-7.6327832943e-17 -1.1102230246e-16 -1.1102230246e-16 -1.1102230246e-16

det(A) = -1532.857200

Число обусловленности матрицы А:

Кубическая норма: 23.737143

Октаэдрическая норма: 28.714109

Эвклидова норма: 16.856376

Вариант 25b:

A:

6.0000000 -9.4000000 -4.2000000 -9.1000000

3.5000000 -3.5000000 6.4000000 2.0000000

5.8000000 6.0000000 -3.1000000 -7.0000000

0.4000000 -4.7000000 -2.4000000 1.7000000

vector b:

-61.7000000 23.7000000 -19.5000000 -9.4000000

k = 1 m = 1

U:

1.0000000 -1.5666667 -0.7000000 -1.5166667

0.0000000 1.9833333 8.8500000 7.3083333

0.0000000 15.0866667 0.9600000 1.7966667

0.0000000 -4.0733333 -2.1200000 2.3066667

L:

6.0000000 0.0000000 0.0000000 0.0000000

3.5000000 0.0000000 0.0000000 0.0000000

5.8000000 0.0000000 0.0000000 0.0000000

0.4000000 0.0000000 0.0000000 0.0000000

k = 2 m = 3

U:

1.0000000 -1.5666667 -0.7000000 -1.5166667

0.0000000 1.0000000 0.0636323 0.1190897

0.0000000 0.0000000 8.7237958 7.0721388

0.0000000 0.0000000 -1.8608042 2.7917587

L:

6.0000000 0.0000000 0.0000000 0.0000000

5.8000000 15.0866667 0.0000000 0.0000000

3.5000000 1.9833333 0.0000000 0.0000000

0.4000000 -4.0733333 0.0000000 0.0000000

k = 3 m = 3

U:

1.0000000 -1.5666667 -0.7000000 -1.5166667

0.0000000 1.0000000 0.0636323 0.1190897

0.0000000 0.0000000 1.0000000 0.8106722

0.0000000 0.0000000 0.0000000 4.3002610

L:

6.0000000 0.0000000 0.0000000 0.0000000

5.8000000 15.0866667 0.0000000 0.0000000

3.5000000 1.9833333 8.7237958 0.0000000

0.4000000 -4.0733333 -1.8608042 0.0000000

k = 4 m = 4

U:

1.0000000 -1.5666667 -0.7000000 -1.5166667

0.0000000 1.0000000 0.0636323 0.1190897

0.0000000 0.0000000 1.0000000 0.8106722

0.0000000 0.0000000 0.0000000 1.0000000

L:

6.0000000 0.0000000 0.0000000 0.0000000

5.8000000 15.0866667 0.0000000 0.0000000

3.5000000 1.9833333 8.7237958 0.0000000

0.4000000 -4.0733333 -1.8608042 4.3002610

Rank = 4

LU – PA:

0.0000000000e+00 0.0000000000e+00 0.0000000000e+00 0.0000000000e+00

0.0000000000e+00 0.0000000000e+00 0.0000000000e+00 0.0000000000e+00

0.0000000000e+00 0.0000000000e+00 1.7763568394e-15 0.0000000000e+00

0.0000000000e+00 0.0000000000e+00 -4.4408920985e-16 2.2204460493e-16

vector x:

0.9951533 1.9945925 3.0027817 3.9901173

Ax – b:

7.1054273576e-15 0.0000000000e+00 0.0000000000e+00 -1.7763568394e-15

x\* – x:

4.8466917357e-03 5.4074985979e-03 -2.7816538649e-03 9.8827043765e-03

A-1:

-0.0484669 0.1106492 0.1422542 0.1961366

-0.0540750 -0.0106425 0.0634444 -0.0156978

0.0278165 0.0744179 -0.0606819 -0.1885170

-0.0988270 0.0496021 0.0562650 0.2325440

A\*A-1:

1.0000000000e+00 1.1102230246e-16 -1.1102230246e-16 0.0000000000e+00

2.2204460493e-16 1.0000000000e+00 -8.3266726847e-17 0.0000000000e+00

2.2204460493e-16 1.1102230246e-16 1.0000000000e+00 -4.4408920985e-16

2.7755575616e-17 4.1633363423e-17 -1.3877787808e-17 1.0000000000e+00

(A\*A-1) – E:

0.0000000000e+00 1.1102230246e-16 -1.1102230246e-16 0.0000000000e+00

2.2204460493e-16 -1.1102230246e-16 -8.3266726847e-17 0.0000000000e+00

2.2204460493e-16 1.1102230246e-16 -2.2204460493e-16 -4.4408920985e-16

2.7755575616e-17 4.1633363423e-17 -1.3877787808e-17 -2.2204460493e-16

det(A) = -3395.821500

Число обусловленности матрицы А:

Кубическая норма: 14.278451

Октаэдрическая норма: 14.936333

Эвклидова норма: 6.478387

Краткие выводы

Мы разобрались в одном из методов решения СЛАУ – в LU разложении. С помощью этого метода можно не только находить решения систем уравнений, но и вычислять обратную матрицу и определитель, а также находить решения СЛАУ с постоянной исходной матрицей, но с разными векторами правой части, используя одну и ту же LU матрицу, что является важным преимуществом в экономии вычислительных затрат.

Текст программы

#include <iostream>

#include <algorithm>

#define M\_PI 3.14159265358979323846

using namespace std;

const double minValue = 0.00001;

//распечатка вектора

void printVector(double\* vector, int N, bool expFormat = false) {

printf(" ");

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

if (expFormat) {

printf("%.10e ", vector[i]);

}

else {

printf("%.7f ", vector[i]);

}

}

printf("\n");

}

//распечатка матрицы

void printMatrix(double\*\* matrix, int N, bool expFormat = false) {

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

printVector(matrix[i], N, expFormat);

}

printf("\n");

}

//создаем в памяти матрицу

void createMatrix(double\*\*\* pMatrix, int N) {

auto& matrix = \*pMatrix;

matrix = new double\* [N];

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

matrix[i] = new double[N];

}

}

//заполняем матрицу значениями

template<int N>

void fillMatrix(double\*\* matrix, double values[N][N]) {

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

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

matrix[i][j] = values[i][j];

}

}

}

//заполняем матрицу нулями

void fillMatrixAsEmpty(double\*\* matrix, int N) {

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

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

matrix[i][j] = 0.0;

}

}

}

//заполняем матрицу как единичную

void fillMatrixAsE(double\*\* matrix, int N) {

fillMatrixAsEmpty(matrix, N);

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

matrix[i][i] = 1.0;

}

}

//копируем значения одной матрицы в другую

void copyMatrixToMatrix(double\*\* srcMatrix, double\*\* dstMatrix, int N) {

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

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

dstMatrix[i][j] = srcMatrix[i][j];

}

}

}

//умножение матрицы на вектор

void matrixMulVec(double\*\* A, double\* B, double\* C, int N)

{

for (int i = 0; i < N; i++)

for (int j = 0; j < N; j++)

C[i] += A[i][j] \* B[j];

}

//умножение матрицы на матрицу

void matrixMul(double\*\* A, double\*\* B, double\*\* C, int N)

{

for (int i = 0; i < N; i++)

for (int j = 0; j < N; j++)

for (int k = 0; k < N; k++)

C[i][j] += A[i][k] \* B[k][j];

}

//вычитание матрицы

void matrixSub(double\*\* A, double\*\* B, double\*\* C, int N)

{

for (int i = 0; i < N; i++)

for (int j = 0; j < N; j++)

C[i][j] = A[i][j] - B[i][j];

}

//получение матрицы PA

double\*\* matrixPA(double\*\* A, int\* P, int N)

{

double\*\* PA = new double\* [N];

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

PA[i] = A[P[i]];

}

return PA;

}

//вычитание векторов

void vectorSub(double\* A, double\* B, double\* C, int N)

{

for (int i = 0; i < N; i++)

C[i] = A[i] - B[i];

}

//определить строку с главным элементом (который максимален в текущем столбце)

int defineRowIdxWithMainValue(double\*\* matrix, int k, int N) {

int m = k;

double maxValue = 0.0;

for (int i = k; i < N; i++) {

if (abs(matrix[i][k]) > maxValue) {

m = i;

maxValue = abs(matrix[m][k]);

}

}

return m;

}

//LU разложение

void LUdecomposition(double\*\* L, double\*\* U, int\* P, int& rank, double& sign, int N) {

//иницилизируем подстановку P

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

P[i] = i;

}

printf("U = A:\n");

printMatrix(U, N);

printf("L:\n");

printMatrix(L, N);

for (int k = 0; k < N; k++) {

auto rowIdx = defineRowIdxWithMainValue(U, k, N);

if (k != rowIdx) {

//Смена строк

swap(U[k], U[rowIdx]);

swap(L[k], L[rowIdx]);

swap(P[k], P[rowIdx]);

sign \*= -1.0;

}

//главный элемент

double mainValue = U[k][k];

if (abs(mainValue) < minValue) {

//Определяем ранг матрицы

rank = k;

return;

}

//заполняем матрицу L

for (int i = k; i < N; i++) {

L[i][k] = U[i][k];

//printMatrix(L, N);

}

for (int j = k; j < N; j++) {

U[k][j] /= mainValue;

}

//заполняем матрицу U

for (int i = k + 1; i < N; i++) {

for (int j = k; j < N; j++) {

U[i][j] = U[i][j] - L[i][k] \* U[k][j];

//printMatrix(U, N);

}

}

printf("\nk = %i\nm = %i\nU[m][k] = %.7f\nU:\n", k, rowIdx, mainValue);

printMatrix(U, N);

printf("L:\n");

printMatrix(L, N);

}

}

//Решение уравнения Ly = Pb

void SolveLy(double\*\* triangleMatrix, double\* X, double\* B, int N) {

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

X[i] = B[i] / triangleMatrix[i][i];

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

X[i] -= X[j] \* triangleMatrix[i][j] / triangleMatrix[i][i];

}

}

}

//Решение уравнения Ux = y

void SolveUx(double\*\* triangleMatrix, double\* X, double\* B, int N) {

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

X[i] = B[i];

for (int j = N - 1; j > i; j--) {

X[i] -= X[j] \* triangleMatrix[i][j];

}

}

}

//Решение уравнения Ax = b, то есть LUx = Pb

void SolveSOLE(double\*\* L, double\*\* U, double\* X, int\* P, double\* B, int N) {

//Ax = b (A = PLU)

//LUx = Pb (Ux = y)

//Ly = Pb

double\* vectorY = new double[N];

//"умножаем" вектор b на матрицу перестановок

double\* vectorPB = new double[N];

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

vectorPB[i] = B[P[i]];

}

SolveLy(L, vectorY, vectorPB, N);

//printVector(Y, N);

SolveUx(U, X, vectorY, N);

}

void SolveBackwardMatrix(double\*\* L, double\*\* U, double\*\* X, int\* P, int N) {

//LUX = PE

double\* vectorX = new double[N];

double\* vectorE = new double[N];

for (int t = 0; t < N; t++) {

vectorE[t] = 0.0;

}

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

//формируем вектор Ei

if (i != 0) {

vectorE[i - 1] = 0.0;

}

vectorE[i] = 1.0;

//получаем вектор-столбец X

SolveSOLE(L, U, vectorX, P, vectorE, N);

//записываем его в матрицу X

for (int t = 0; t < N; t++) {

X[t][i] = vectorX[t];

}

}

}

//найти определитель

double computeDet(double\*\* L, int N, double sign) {

double det = sign;

for (int t = 0; t < N; t++) {

det \*= L[t][t];

}

return det;

}

//транспонировать матрицу

void transpose(double\*\* matrix, int N) {

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

for (int j = i; j < N; j++) {

swap(matrix[j][i], matrix[i][j]);

}

}

}

//вычислить кубическую норму

double computeCubNorm(double\*\* matrix, int N) {

double result = 0.0;

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

double sum = 0.0;

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

sum += abs(matrix[i][j]);

}

result = max(result, sum);

}

return result;

}

//////////////////////////////////////////////////////////////////////////////////////////////////

//Нахождение максимального элемента в матрице, находящегося не на диагонали

void searchMaxElemMatrix(double\*\* matrix, const int N, int& imax, int& jmax) {

double max = 0.0;

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

if (i != j && abs(matrix[i][j]) > max)

{

max = abs(matrix[i][j]);

imax = i;

jmax = j;

}

}

}

}

//tan(2\*alpha) = 2\*a[i][j]/(a[i][i]-a[j][j])

double getAlpha(double\*\* matrix, int imax, int jmax) {

double alpha;

if (matrix[imax][imax] - matrix[jmax][jmax] == 0)

{

alpha = M\_PI / 4;

}

else

{

alpha = atan(2 \* matrix[imax][jmax] / (matrix[imax][imax] - matrix[jmax][jmax])) / 2;

}

return alpha;

}

//Является ли матрица диагональной

bool isMatrixDiagonal(double\*\* matrix, const int N) {

double kvSum = 0.0;

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

if (i != j)

{

kvSum += matrix[i][j] \* matrix[i][j];

}

}

}

return kvSum < minValue\* minValue;

}

//Получение диагональной матрицы

double\*\* getNewDiagonalMatrixByRotation(double\*\* matrix, const int N) {

double\* vectorI = new double[N];

double\* vectorJ = new double[N];

double\*\* rotatedMatrix = nullptr;

createMatrix(&rotatedMatrix, N);

copyMatrixToMatrix(matrix, rotatedMatrix, N);

//вспомогательная матрица

double\*\* B = nullptr;

createMatrix(&B, N);

while (!isMatrixDiagonal(rotatedMatrix, N)) {

//printMatrix(rotatedMatrix, N);

int imax, jmax;

searchMaxElemMatrix(rotatedMatrix, N, imax, jmax);

double alpha = getAlpha(rotatedMatrix, imax, jmax);

double c = cos(alpha);

double s = sin(alpha);

//результат умножения матрицы A в k-ом состоянии на матрицу вращения справа

copyMatrixToMatrix(rotatedMatrix, B, N);

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

B[m][imax] = c \* rotatedMatrix[m][imax] + s \* rotatedMatrix[m][jmax];

B[m][jmax] = -s \* rotatedMatrix[m][imax] + c \* rotatedMatrix[m][jmax];

}

//результат умножения матрицы B на матрицу вращения слева

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

vectorI[m] = c \* B[imax][m] + s \* B[jmax][m];

vectorJ[m] = -s \* B[imax][m] + c \* B[jmax][m];

}

swap(B[imax], vectorI);

swap(B[jmax], vectorJ);

copyMatrixToMatrix(B, rotatedMatrix, N);

}

return rotatedMatrix;

}

//Получение максимального собственного значения

double getMaxEigenvalue(double\*\* matrix, const int N) {

double max = 0;

for (int i = 0; i < N; i++)

{

if (abs(matrix[i][i]) > max)

max = abs(matrix[i][i]);

}

return max;

}

//Вычисление евклидовой нормы матрицы

double computEuclidNorm(double\*\* A, double\*\* trA, const int N) {

double\*\* newMatrix = nullptr;

createMatrix(&newMatrix, N);

fillMatrixAsEmpty(newMatrix, N);

matrixMul(trA, A, newMatrix, N);

//printMatrix(newMatrix, N);

newMatrix = getNewDiagonalMatrixByRotation(newMatrix, N);

//printMatrix(newMatrix, N);

double eigenvalue = getMaxEigenvalue(newMatrix, N);

return sqrt(eigenvalue);

}

int main()

{

const int N = 4;

/\*double matrixValues[N][N] = {

{ 0.1, -8.3, 7.1, 5.5},

{ 1.2, 5.2, -9.1, -0.2},

{ -7.9, 9.6, 0.9, -1.2},

{ 3.8, -4.7, -0.2, 7.9}

};

double vectorB[N] = {

26.8, -16.5, 9.2, 25.4

};\*/

double matrixValues[N][N] = {

{ 6.0, -9.4, -4.2, -9.1},

{ 3.5, -3.5, 6.4, 2.0},

{ 5.8, 6.0, -3.1, -7.0},

{ 0.4, -4.7, -2.4, 1.7}

};

double vectorB[N] = {

-61.7, 23.7, -19.5, -9.4

};

double vectorX\_[N] = {

1.0, 2.0, 3.0, 4.0

};

//единичная матрица

double\*\* E = nullptr;

createMatrix(&E, N);

fillMatrixAsE(E, N);

//Матрица A

double\*\* A = nullptr;

createMatrix(&A, N);

fillMatrix<N>(A, matrixValues);

//Матрица U

double\*\* U = nullptr;

createMatrix(&U, N);

copyMatrixToMatrix(A, U, N);

//Матрица L

double\*\* L = nullptr;

createMatrix(&L, N);

fillMatrixAsEmpty(L, N);

printf("1) Input\A:\n");

printMatrix(A, N);

printf("B:\n");

printVector(vectorB, N);

//LU разложение

int rank = N; //ранг матрицы

int P[N]; //"матрица" перестановок (на самом деле подстановка)

double sign = 1.0;

printf("\n2) LU decomposition\n", rank);

LUdecomposition(L, U, P, rank, sign, N);

printf("\nRank = %i\n", rank);

if (rank != N)

return 0;

auto PA = matrixPA(A, P, N);

//LU матрица

double\*\* LU = nullptr;

createMatrix(&LU, N);

fillMatrixAsEmpty(LU, N);

matrixMul(L, U, LU, N);

if (false) { //show PA/LU

printf("Matrix PA:\n");

printMatrix(PA, N);

printf("Matrix LU (check to see LU = PA):\n");

printMatrix(LU, N);

}

//LU - PA

printf("\nLU-PA:\n");

double\*\* LUminusPA = nullptr;

createMatrix(&LUminusPA, N);

matrixSub(LU, PA, LUminusPA, N);

printMatrix(LUminusPA, N, true);

//решаем СЛАУ

double vectorX[N];

SolveSOLE(L, U, vectorX, P, vectorB, N);

printf("\n3) SOLE\nVector X:\n");

printVector(vectorX, N);

//вектор невязки

double vectorX2[N];

double vectorR[N];

for (int i = 0; i < N; i++) vectorX2[i] = 0;

matrixMulVec(A, vectorX, vectorX2, N);

vectorSub(vectorX2, vectorB, vectorR, N);

printf("\nAx - b = \n");

printVector(vectorR, N, true);

//погрешность решения

double vectorDelta[N];

vectorSub(vectorX\_, vectorX, vectorDelta, N);

printf("X\* - X = \n");

printVector(vectorDelta, N, true);

//находим обратную матрицу

double\*\* backwardMatrix = nullptr;

createMatrix(&backwardMatrix, N);

SolveBackwardMatrix(L, U, backwardMatrix, P, N);

printf("\n4) Other:\nA^-1:\n");

printMatrix(backwardMatrix, N);

//A \* A^-1

double\*\* AA = nullptr;

createMatrix(&AA, N);

fillMatrixAsEmpty(AA, N);

matrixMul(A, backwardMatrix, AA, N);

printf("\nA \* A^-1:\n");

printMatrix(AA, N, true);

double\*\* AAminusE = nullptr;

createMatrix(&AAminusE, N);

matrixSub(AA, E, AAminusE, N);

printf("(A \* A^-1) - E:\n");

printMatrix(AAminusE, N, true);

//найти определитель

auto det = computeDet(L, N, sign);

printf("\ndet(A) = %f\n", det);

//транспонируем матрицу

double\*\* trA = nullptr;

createMatrix(&trA, N);

copyMatrixToMatrix(A, trA, N);

transpose(trA, N);

//транспонируем обратную матрицу

double\*\* trBackwardMatrix = nullptr;

createMatrix(&trBackwardMatrix, N);

copyMatrixToMatrix(backwardMatrix, trBackwardMatrix, N);

transpose(trBackwardMatrix, N);

//вычисление норм матриц

auto cubNorm1 = computeCubNorm(A, N);

auto cubNorm2 = computeCubNorm(backwardMatrix, N);

auto octNorm1 = computeCubNorm(trA, N);

auto octNorm2 = computeCubNorm(trBackwardMatrix, N);

double euclidNorm1 = computEuclidNorm(A, trA, N);

double euclidNorm2 = computEuclidNorm(backwardMatrix, trBackwardMatrix, N);

//вычисление числа обусловленности

auto cubCond = cubNorm1 \* cubNorm2;

auto octCond = octNorm1 \* octNorm2;

double euclidCond = euclidNorm1 \* euclidNorm2;

printf("cubCond(A) = %f\noctCond(A) = %f\neuclidCond(A) = %f\n", cubCond, octCond, euclidCond);

return 0;

}