**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский Авиационный Институт»**

**(Национальный Исследовательский Университет)**

**Институт: №8 «Информационные технологии   
и прикладная математика»   
Кафедра: 806 «Вычислительная математика   
и программирование»**

Лабораторная работа № 1   
по курсу «Численные методы»

Группа: М8О-308Б-21

Студентка: Шевлякова С. С.

Преподаватель: Ревизников Д. Л.

Дата: 22.05.2024

Москва, 2024

**ОГЛАВЛЕНИЕ**

[1 Тема 3](#_Toc158983147)

[2 Задание 3](#_Toc158983148)

[3 Теория 4](#_Toc158983149)

[4 Ход лабораторной работы 5](#_Toc158983150)

[5 Выводы 6](#_Toc158983151)

# **Тема**

Методы решения задач линейной алгебры

# **Задание**

1.1. Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.



1.2. Реализовать метод прогонки в виде программы, задавая в качестве входных данных ненулевые элементы матрицы системы и вектор правых частей. Используя разработанное программное обеспечение, решить СЛАУ с трехдиагональной матрицей.



1.3. Реализовать метод простых итераций и метод Зейделя в виде программ, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.

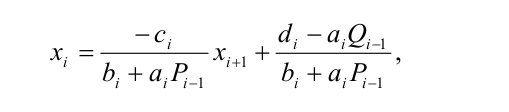


1.4. Реализовать метод вращений в виде программы, задавая в качестве входных данных матрицу и точность вычислений. Используя разработанное программное обеспечение, найти собственные значения и собственные векторы симметрических матриц. Проанализировать зависимость погрешности вычислений от числа итераций.

1.5. Реализовать алгоритм QR – разложения матриц в виде программы. На его основе разработать программу, реализующую QR – алгоритм решения полной проблемы собственных значений произвольных матриц, задавая в качестве входных данных матрицу и точность вычислений. С использованием разработанного программного обеспечения найти собственные значения матрицы.

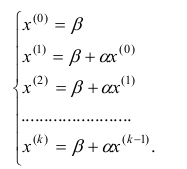
# **Теория**

**LU – разложение** матрицы A представляет собой разложение матрицы A в произведение нижней и верхней треугольных матриц, т.е. A = LU, где L - нижняя треугольная матрица (матрица, у которой все элементы, находящиеся выше главной диагонали равны нулю, l ij = 0 при i < j ), U - верхняя треугольная матрица (матрица, у которой все элементы, находящиеся ниже главной диагонали равны нулю, u ij = 0 при i > j ). В дальнейшем LU – разложение может быть эффективно использовано при решении систем линейных алгебраических уравнений вида Ax = b.

**Метод прогонки** является одним из эффективных методов решения СЛАУ с трех - диагональными матрицами, возникающих при конечно-разностной аппроксимации задач для обыкновенных дифференциальных уравнений (ОДУ) и уравнений в частных производных второго порядка и является частным случаем метода Гаусса.

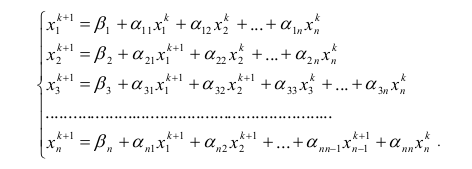


Методы последовательных приближений, в которых при вычислении последующего приближения решения используются предыдущие, уже известные приближенные решения, называются **итерационными.**



Для сходимости итерационного процесса необходимо и достаточно, чтобы спектр матрицы α эквивалентной системы лежал внутри круга с радиусом, равным единице.

Метод простых итераций довольно медленно сходится. Для его ускорения существует **метод Зейделя**, заключающийся в том, что при вычислении компонента вектора неизвестных на (k+1)-й итерации используются x1k+1, x2k+1, …, xi-1k+1, уже вычисленные на (k+1)-й итерации.



При решении полной проблемы собственных значений для несимметричных матриц эффективным является подход, основанный на приведении матриц к подобным, имеющим треугольный или квазитреугольный вид. Одним из наиболее распространенных методов этого класса является **QR-алгоритм**, позволяющий находить как вещественные, так и комплексные собственные значения.

В основе QR-алгоритма лежит представление матрицы в виде A = QR , где Q-ортогональная матрица, а R - верхняя треугольная. Такое разложение существует для любой квадратной матрицы. Одним из возможных подходов к построению QR разложения является использование преобразования Хаусхолдера, позволяющего обратить в нуль группу поддиагональных элементов столбца матрицы.

# **Х****од лабораторной работы**

Код был реализован на языке C++, до этого мной был уже написан класс matrix с необходимыми методами

1.1) LU-разложение:

**pair<Matrix, Matrix> LU() {**

**swp\_.clear();**

**int n = this->GetRows();**

**Matrix U(\*this);**

**Matrix L(n, n);**

**for (int k = 0; k < n; ++k) { // k - номер итерации в методе Гаусса, номер столбца, который зануляем**

**int index = k; // index - индекс max по модулю элемента в k столбце**

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

**if (abs(U(i, k)) > abs(U(index, k))) {**

**index = i;**

**}**

**}**

**swap(U(k), U(index));**

**swap(L(k), L(index));**

**swp\_.push\_back(index);**

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

**double m = U(i, k) / U(k, k);**

**L(i, k) = m;**

**for (int j = k; j < n; ++j) {**

**U(i, j) -= m \* U(k, j);**

**}**

**}**

**}**

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

**L(i, i) = 1;**

**}**

**return {L, U};**

**}**

**Matrix Solve(Matrix &C, Matrix &L, Matrix &U) {**

**Matrix B(C);**

**vector<int> swp = this->GetSwp();**

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

**swap(B(i), B(swp[i]));**

**}**

**int n = this->GetRows();**

**// LUx = b**

**// Lz = b**

**Matrix Z(n, 1);**

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

**Z(i, 0) = B(i, 0);**

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

**Z(i, 0) -= L(i, j) \* Z(j, 0);**

**}**

**}**

**// Ux = z**

**Matrix X(n, 1);**

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

**X(i, 0) = Z(i, 0);**

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

**X(i, 0) -= U(i, j) \* X(j, 0);**

**}**

**X(i, 0) = X(i, 0) / U(i, i);**

**}**

**return X;**

**}**

**double Determinant() {**

**// detA = det(LU) = detL \* detU = detU**

**double result = 1;**

**auto [L, U] = this->LU();**

**for (int i = 0; i < rows\_; ++i) {**

**result \*= U(i, i);**

**}**

**// так как при swap строк меняется знак определителя**

**int sign = 0;**

**vector<int> swp = this->GetSwp();**

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

**if (swp[i] != i)**

**++sign;**

**}**

**if (sign % 2 != 0)**

**result = -result;**

**return result;**

**}**

**Matrix InverseMatrix() {**

**int n = this->GetRows();**

**Matrix B(n, 1);**

**Matrix result(n, n);**

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

**if (i > 0)**

**B(i - 1, 0) = 0;**

**B(i, 0) = 1;**

**auto [L, U] = this->LU();**

**Matrix res\_i = this->Solve(B, L, U);**

**for (int k = 0; k < n; ++k)**

**result(k, i) = res\_i(k, 0);**

**}**

**return result;**

**}**

1.2) Метод прогонки:

**Matrix run\_through\_method(Matrix &B) {**

**int n = this->rows\_;**

**vector<double> P, Q; // x\_n = P\_n \* x\_n+1 + Q\_n**

**P.push\_back((-1) \* (\*this)(0, 1) / (\*this)(0, 0)); // P[0] = -c1/b1**

**Q.push\_back(B(0, 0) / (\*this)(0, 0)); // Q[0] = d1/b1**

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

**if (i == n - 1) {**

**P.push\_back(0); // c\_n = 0**

**} else {**

**P.push\_back((-1) \* (\*this)(i, i + 1) / ((\*this)(i, i) + (\*this)(i, i - 1) \* P[i - 1])); // P\_i = -c\_i / (b\_i + a\_i \* P\_i-1)**

**}**

**Q.push\_back((B(i, 0) - (\*this)(i, i - 1) \* Q[i - 1]) / ((\*this)(i, i) + (\*this)(i, i - 1) \* P[i - 1])); // Q\_i = (d\_i - a\_i \* Q\_i-1) / (b\_i + a\_i \* P\_i-1)**

**}**

**Matrix X(n, 1);**

**X(n - 1, 0) = Q[n - 1];**

**for (int i = n - 2; i >= 0; --i) {**

**X(i, 0) = P[i] \* X(i + 1, 0) + Q[i];**

**}**

**return X;**

**}**

1.3) Метод простых итераций и метод Зейделя:

**pair<Matrix, int> simple\_iterations(Matrix &B, double eps) {**

**int n = this->rows\_;**

**Matrix Alpha(n, n);**

**Matrix Beta(n, 1);**

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

**for (int j = 0; j < n; ++j) {**

**Alpha(i, j) = (-1) \* (\*this)(i, j) / (\*this)(i, i);**

**}**

**Alpha(i, i) = 0;**

**Beta(i, 0) = B(i, 0) / (\*this)(i, i);**

**}**

**Matrix X(n, 1), Prev\_X(n, 1);**

**int k = 1;**

**Prev\_X = Beta;**

**X = Beta + Alpha \* Prev\_X;**

**// eps\_k = ||Alpha|| / (1 - ||Alpha||) \* ||x\_k - x\_k-1||**

**double eps\_k = 0;**

**double norm = Alpha.norm();**

**if (norm >= 1) {**

**eps\_k = (X - Prev\_X).norm();**

**} else {**

**eps\_k = norm / (1 - norm) \* (X - Prev\_X).norm();**

**}**

**while (eps\_k > eps) { // eps\_k <= eps**

**Prev\_X = X;**

**X = Beta + Alpha \* X;**

**if (norm >= 1) {**

**eps\_k = (X - Prev\_X).norm();**

**} else {**

**eps\_k = norm / (1 - norm) \* (X - Prev\_X).norm();**

**}**

**++k;**

**}**

**return {X, k};**

**}**

**pair<Matrix, int> seidel(Matrix &R, double eps) {**

**int n = this->rows\_;**

**Matrix Alpha(n, n), Beta(n, 1), E(n, n);**

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

**for (int j = 0; j < n; ++j) {**

**Alpha(i, j) = (-1) \* (\*this)(i, j) / (\*this)(i, i);**

**}**

**Alpha(i, i) = 0;**

**E(i, i) = 1;**

**Beta(i, 0) = R(i, 0) / (\*this)(i, i);**

**}**

**// Alpha = B + C**

**Matrix C(n, n), B(n, n);**

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

**for (int j = 0; j < n; ++j) {**

**if (j < i)**

**B(i, j) = Alpha(i, j);**

**else**

**C(i, j) = Alpha(i, j);**

**}**

**}**

**// x\_k+1 = (E - B)^-1 \* C \* x\_k + (E - B)^-1 \* Beta**

**Matrix X(n, 1), Prev\_X(n, 1);**

**int k = 1;**

**Prev\_X = Beta;**

**Matrix Tmp\_Beta = (E - B).InverseMatrix() \* Beta;**

**Matrix Tmp\_Alpha = (E - B).InverseMatrix() \* C;**

**X = Tmp\_Alpha \* Prev\_X + Tmp\_Beta;**

**double eps\_k = 0;**

**double norm = Alpha.norm();**

**if (norm >= 1) {**

**eps\_k = (X - Prev\_X).norm();**

**} else {**

**eps\_k = C.norm() / (1 - norm) \* (X - Prev\_X).norm();**

**}**

**// eps\_k = ||C|| / (1 - ||Alpha||) \* ||x\_k - x\_k-1||**

**while (eps\_k > eps) {**

**Prev\_X = X;**

**X = Tmp\_Alpha \* Prev\_X + Tmp\_Beta;**

**if (norm >= 1) {**

**eps\_k = (X - Prev\_X).norm();**

**} else {**

**eps\_k = C.norm() / (1 - norm) \* (X - Prev\_X).norm();**

**}**

**++k;**

**}**

**return {X, k};**

**}**

1.4) Метод вращения:

**pair<pair<Matrix, Matrix>, int> jacobi\_method(double eps) {**

**int n = this->rows\_;**

**int k = 0;**

**pair<int, int> max\_index;**

**Matrix A = \*this;**

**Matrix Self\_Vectors(n, n);**

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

**Self\_Vectors(i, i) = 1;**

**}**

**while (A.sum\_square() > eps) {**

**Matrix U(n, n);**

**max\_index = {1, 0}; // index abs(max\_elem)**

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

**for (int j = 0; j < n; ++j) {**

**if (i != j && abs(A(i, j)) > abs(A(max\_index.first, max\_index.second)))**

**max\_index = {i, j};**

**}**

**}**

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

**U(i, i) = 1;**

**}**

**// phi = 1/2 \* arctg (2 \* a(i, j) / (a(i, i) - a(j, j)))**

**// phi = PI/4, a(i, i) = a(j, j)**

**double phi;**

**if (A(max\_index.first, max\_index.first) == A(max\_index.second, max\_index.second))**

**phi = M\_PI / 4;**

**else**

**phi = 0.5 \* atan(2 \* A(max\_index.first, max\_index.second) / (A(max\_index.first, max\_index.first) - A(max\_index.second, max\_index.second)));**

**U(max\_index.first, max\_index.first) = cos(phi);**

**U(max\_index.first, max\_index.second) = (-1) \* sin(phi);**

**U(max\_index.second, max\_index.first) = sin(phi);**

**U(max\_index.second, max\_index.second) = cos(phi);**

**Matrix U\_T = U.Transpose();**

**// A^k+1 = U\_T^k \* A^k \* U^k**

**A = U\_T.MulMatrixReturn(A).MulMatrixReturn(U);**

**Self\_Vectors.MulMatrix(U);**

**++k;**

**}**

**return {{A, Self\_Vectors}, k};**

**}**

1.5) QR-разложение:

**vector<complex<double>> qr\_method(double eps) {**

**int n = this->rows\_;**

**Matrix A = \*this;**

**vector<complex<double>> lambda;**

**vector<complex<double>> lambda\_prev;**

**int counter = 0;**

**int iter = 50;**

**while (true) {**

**auto [Q, R] = A.qr\_decomposition();**

**A = R.MulMatrixReturn(Q);**

**// cout << "A\n";**

**// A.ShowMatrix();**

**if (counter != iter) {**

**++counter;**

**continue;**

**}**

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

**double sum = 0;**

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

**sum += abs(A(j, i));**

**}**

**if (sum < 0.001) {**

**lambda.push\_back(A(i, i));**

**} else {**

**// (a\_jj - Lambda)(a\_j+1,j+1 - Lambda) = aj,j+1 \* aj+1, j**

**double a = 1;**

**double b = (-1) \* (A(i, i) + A(i + 1, i + 1));**

**double c = A(i, i) \* A(i + 1, i + 1) - A(i, i + 1) \* A(i + 1, i);**

**double d = b \* b - 4 \* c;**

**complex<double> x1, x2;**

**if (d < 0) {**

**x1 = (-b + sqrt((abs(d))) \* complex<double>(0, 1)) / (2 \* a);**

**x2 = (-b - sqrt((abs(d))) \* complex<double>(0, 1)) / (2 \* a);**

**} else {**

**x1 = (-b + sqrt(d)) / (2 \* a);**

**x2 = (-b - sqrt(d)) / (2 \* a);**

**}**

**lambda.push\_back(x1);**

**lambda.push\_back(x2);**

**++i;**

**}**

**}**

**bool exit = true;**

**// исключаем первую итерацию**

**if (lambda\_prev.size() != 0) {**

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

**if (abs(lambda[i] - lambda\_prev[i]) > eps) {**

**exit = false;**

**break;**

**}**

**}**

**if (exit == true)**

**break;**

**}**

**lambda\_prev = lambda;**

**lambda.clear();**

**counter = 0;**

**}**

**return lambda;**

**}**

# **Выводы**

В этой лабораторной работе рассматриваются численные методы решения систем линейных алгебраических уравнений (СЛАУ) и численные методы решения задач на собственные значения и собственные векторы матриц. Среди численных методов алгебры существуют прямые методы, в которых решение получается за конечное фиксированное число операций и итерационные методы, в которых результат достигается в процессе последовательных приближений.

В ходе лабораторной работы были реализованы методы LU-разложения матриц, с помощью которого решается СЛАУ. метод прогонки, метод простых итераций, метод Зейделя, QR-разложение.