|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ТЕОРЕТИЧЕСКАЯ ИНФОРМАТИКА И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

КАФЕДРА \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ЛАБОРАТОРНАЯ РАБОТА №5

по дисциплине: «Численные методы линейной алгебры»

ИУ9-72б

Ф. Р. Базартинова

Выполнил студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Д.П. Посевин

Проверил **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

1 Постановка задачи

Необходимо реализовать разложение Холецкого и LU-разложение квадратной симметричной положительно определенной матрицы. Оба разложения нужно проверить решением СЛАУ методом Гаусса. Также необходимо реализовать поиск определителя исходной и обратной к ней матриц с использованием LU-разложения. Полученные данные необходимо проверить с помощью встроенных библиотек линейной алгебры.

2 Программная реализация

Лабораторная работа была выполнена на языке C++. Проверка осуществлялась с помощью библиотеки Eigen. Исходный код основной программы приведён в листинге 1.

|  |
| --- |
| Листинг 1 - Реализация |
| #include <iostream>  #include <vector>  #include <cmath>  #include <fstream>  #include <eigen3/Eigen/Dense>  using namespace std;  Eigen::MatrixXf to\_eigen\_m(vector<vector<float>> m) {  Eigen::MatrixXf res(m.size(), m.size());  for (size\_t x = 0; x < m.size(); x++)  for (size\_t y = 0; y < m.size(); y++)  res(x, y) = m[x][y];  return res;  }  vector<vector<float>> randomMatrix(size\_t n) {  vector<vector<float>> res(n, vector<float>(n));  for (int i = 0; i < n; i++)  for (int j = 0; j < n; j++)  if (i == j) res[i][j] = rand() / static\_cast<float>(RAND\_MAX) \* 10.f ;  return res;  }  vector<float> randomVector(int n) {  vector<float> res(n);  for (int i = 0; i < n; i++)  res[i] = rand() / static\_cast<float>(RAND\_MAX) \* 10.f;  return res;  }  Eigen::VectorXf to\_eigen\_v(vector<float> v) {  Eigen::VectorXf res(v.size());  for (size\_t x = 0; x < v.size(); x++)  res(x) = v[x];  return res;  }  vector<vector<float>> tril(vector<vector<float>> arr){  for (int i = 0; i < arr.size(); i++){  for (int j = 0; j < arr[0].size(); j++){  if (j > i) arr[i][j] = 0;  }  }  return arr;  }  vector<vector<float>> toNormMat(Eigen::MatrixXf m, int n){  vector<vector<float>> res(n, vector<float>(n));  for (size\_t i = 0; i < n; i++)  for (size\_t j = 0; j < n; j++)  res[i][j] = m(i, j);  return res;  }  vector<float> toNormVec(Eigen::VectorXf m, int n){  vector<float> res(n);  for (size\_t i = 0; i < n; i++)  res[i] = m(i);  return res;  }  vector<vector<float>> cholesky(vector<vector<float>> A, int n){  vector<vector<float>> L (n, vector<float>(n));  for (int i = 0; i < n; i++){  for (int j = 0; j < n; j++){  L[i][j] = 0;  }  }  L[0][0] = sqrt(A[0][0]);  for (int i = 0; i < n; i++){  L[i][0] = A[i][0] / L[0][0];  }  for (int i = 1; i < n; i++){  float sum1 = 0;  for (int j = 0; j < i; j++){  sum1 += L[i][j] \* L[i][j];  }  L[i][i] = sqrtf(A[i][i] - sum1);  for (int j = i+1; j < n; j++){  float sum2 = 0;  for (int k = 0; k < i; k++){  sum2 += L[i][k] \* L[j][k];  }  L[j][i] = (A[j][i] - sum2) / L[i][i];  }  }  return L;  }  vector<float> low\_back\_gauss(vector<vector<float>> L, vector<float> b, int n){  vector<float> x (n);  for (int i = 0; i < n; i++) {  float sum0 = b[i];  for (int j = 0; j < i; j++)  sum0 -= L[i][j] \* x[j];  x[i] = sum0 / L[i][i];  }  return x;  }  vector<float> up\_back\_gauss(vector<vector<float>> L, vector<float> b, int n){  vector<float> x (n);  for (int i = n-1; i >-1; i--) {  float sum0 = b[i];  for (int j = n - 1; j >i; j--)  sum0 -= L[i][j] \* x[j];  x[i] = sum0 / L[i][i];  }  return x;  }  pair<vector<vector<float>>, vector<vector<float>>> lu\_decompose(vector<vector<float>> mat, int n) {  vector<vector<float>> U(n, vector<float>(n));  vector<vector<float>> L(n, vector<float>(n));  for (int i = 0; i < n; i++)  L[i][i] = 1;  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  float sum = 0.f;  if (i <= j) {  for (int k = 0; k < i; k++)  sum += L[i][k] \* U[k][j];  U[i][j] = mat[i][j] - sum;  }  else {  for (int k = 0; k < j; k++)  sum += L[i][k] \* U[k][j];  L[i][j] = (mat[i][j] - sum) / U[j][j];  }  }  }  return { L, U };  }  float lu\_det(vector<vector<float>> L, vector<vector<float>> U) {  float res = 1.f;  for (size\_t i = 0; i < L.size(); i++){  res \*= L[i][i] \* U[i][i];  }  return res;  }  vector<vector<float>> lu\_inverse(vector<vector<float>> L, vector<vector<float>> U) {  int n = L.size();  vector<vector<float>> res(n, vector<float>(n));    vector<float> ecol (n);  for (size\_t c = 0; c < n; c++) {  if (c != 0)  ecol[c - 1] = 0.f;  ecol[c] = 1.f;  auto col = up\_back\_gauss(U, low\_back\_gauss(L, ecol, n), n);  for (size\_t i = 0; i < n; i++)  res[i][c] = col[i];  }  return res;  }  vector<vector<float>> getIdentity(int n){  vector<vector<float>> res(n, vector<float>(n));  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++){  if (i == j) res[i][j] = 1;  else res[i][j] = 0;  }  }  return res;  }  float matnorm(vector<vector<float>> mat) {  float max = 0.f;  for (size\_t y = 0; y < mat.size(); y++) {  float cur = 0.f;  for (size\_t x = 0; x < mat.size(); x++)  cur += fabsf(mat[y][x]);  if (max < cur)  max = cur;  }  return max;  }  int main(){  size\_t n = 100;  srand(0xFade18);  vector<vector<float>> arr = randomMatrix(n);  vector<vector<float>> L0 = tril(arr);  vector<vector<float>> A = toNormMat(to\_eigen\_m(L0) \* to\_eigen\_m(L0).transpose(), n);  vector<float> real = randomVector(n);  vector<float> b = toNormVec(to\_eigen\_m(A) \* to\_eigen\_v(real), n);  vector<vector<float>> L = cholesky(A, n);  vector<vector<float>> A0 = toNormMat(to\_eigen\_m(L) \* to\_eigen\_m(L).transpose(), n);  cout << "Cholesky result checking 1: " << abs(to\_eigen\_m(A).norm() - to\_eigen\_m(A0).norm()) << "\n";  vector<float> y = low\_back\_gauss(L, b, n);  vector<vector<float>> LT = toNormMat(to\_eigen\_m(L).transpose(), n);  vector<float> x = up\_back\_gauss(LT, y, n);  cout << "Cholesky result checking 2: " << (to\_eigen\_v(x) - to\_eigen\_v(real)).norm() / to\_eigen\_v(real).norm() << "\n";    n = 15;  vector<vector<float>> A2 = randomMatrix(n);  vector<float> real2 = randomVector(n);  vector<float> b2 = toNormVec(to\_eigen\_m(A2) \* to\_eigen\_v(real2), n);  pair<vector<vector<float>>, vector<vector<float>>> l2u2 = lu\_decompose(A2, n);  vector<vector<float>> l2 = l2u2.first;  vector<vector<float>> u2 = l2u2.second;  cout << "LU result checking 1: " << (to\_eigen\_m(l2) \* to\_eigen\_m(u2) - to\_eigen\_m(A2)).norm() << "\n";  vector<float> y2 = low\_back\_gauss(l2, b2, n);  vector<float> x2 = up\_back\_gauss(u2, y2, n);  cout << "LU result checking 2: " << (to\_eigen\_v(x2) - to\_eigen\_v(real2)).norm() / to\_eigen\_v(real2).norm() << "\n";  float det = lu\_det(l2, u2);  float ldet = to\_eigen\_m(A2).determinant();  cout << "LU result checking 3: " << det - ldet<< "\n";  auto inv = lu\_inverse(l2, u2);  Eigen::MatrixXf check = to\_eigen\_m(A2) \* to\_eigen\_m(inv);  auto m = toNormMat(check - to\_eigen\_m(getIdentity(n)), n);  cout << "LU result checking 4: " << matnorm(m) << "\n";  return 0;  } |

3 Тестирование

Для тестирования необходимо сгенерировать рандомные (симметричные для метода Холецкого) матрицы и векторы. Для тестирования были выбраны матрицы размером 100х100 элементов. Сравнение проводилось на примере норм разности векторов. Результаты представлены в таблицах 1, 2.

Таблица 1. Проверка реализации метода Холецкого.

|  |  |
| --- | --- |
|  | Метод Холецкого |
| Сравнение матриц по норме Фробениуса | 0 |
| Сравнение векторов х по Евклидовой норме | 5.2978e-08 |

Таблица 2. Проверка реализации LU - разложения.

|  |  |
| --- | --- |
|  | LU - разложение |
| Сравнение матриц по норме Фробениуса | 0 |
| Сравнение векторов х по Евклидовой норме | 4.67111e-08 |
| Проверка определителя, вычисленного с помощью LU - разложения | 0 |

|  |  |
| --- | --- |
| Проверка обратной матрицы, вычисленной с помощью LU - разложения | 5.96046e-08 |

4 Вывод

Реализовано разложение Холецкого и LU-разложение квадратной симметричной положительно определенной матрицы. Оба разложения были проверены решением СЛАУ методом Гаусса. Также реализован поиск определителя исходной и обратной к ней матриц с использованием LU-разложения.