|  |  |
| --- | --- |
| **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 №2

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

Реализовать метод Гаусса для решения систем линейных уравнений (СЛАУ), а также:

1. Метод Гаусса с выбором максимального элемента в столбце

2. Метод Гаусса с выбором максимального элемента в строке

3. Метод Гаусса с выбором главного элемента

Корректность реализованных методов необходимо проверить с использованием внешней библиотеки для линейной алгебры.

Далее необходимо вычислить относительную погрешность каждого метода относительно метода с выбором главного элемента.

Также необходимо исследовать относительную погрешность методов в зависимости от размера входных данных путём построения графика.

2 Программная реализация

Лабораторная работа была выполнена на языке C++ с использованием пакета gnuplot для построения графиков. В качестве дополнительной библиотеки используется библиотека eigen3. Исходный код основной программы приведён в листинге 1.

|  |
| --- |
| Листинг 1 |
| #include <iostream>  #include <vector>  #include <cmath>  #include <fstream>  #include <eigen3/Eigen/Dense>  using namespace std;  void elimination(vector<vector<float>> &A, vector<float> &f, int i){  for (int j = i + 1; j < f.size(); j++){  float c = A[i][j] / A[i][i];  A[i][j] = 0.f;  for (int k = i + 1; k < f.size(); k++){  A[k][j] -= c \* A[k][i];  }  f[j] -= c \* f[i];  }  }  void backward(vector<vector<float>> &A, vector<float> &f) {  for (int i = f.size() - 1; i > -1; i--){  f[i] = f[i] / A[i][i];  for (int j = i - 1; j > -1; j--) {  f[j] -= A[i][j] \* f[i];  }  }  }  vector<float> gauss(vector<vector<float>> A, vector<float> f) {  for (int i = 0; i < f.size(); i++){  elimination(A, f, i);  }  backward(A, f);  return f;  }  vector<float> colGauss(vector<vector<float>> A, vector<float> f) {  for (int i = 0; i < f.size(); i++) {  int iMax = i;  float valMax = abs(A[i][i]);  for (int j = i + 1; j < f.size(); j++){  if (valMax < abs(A[i][j])){  valMax = abs(A[i][j]);  iMax = j;  }  }  if (iMax != i)  {  for (int j = i; j < f.size(); j++)  {  float t = A[j][i];  A[j][i] = A[j][iMax];  A[j][iMax] = t;  }  float t = f[i];  f[i] = f[iMax];  f[iMax] = t;  }  elimination(A, f, i);  }  backward(A, f);  return f;  }  vector<float> rowGauss(vector<vector<float>> A, vector<float> f) {  vector<int> xBack(f.size());  for (int i = 0; i < xBack.size(); i++) {  xBack[i] = i;  }  for (int i = 0; i < xBack.size(); i++) {  int jMax = i;  float valmax = abs(A[i][i]);  for (int j = i + 1; j < f.size(); j++){  if (valmax < abs(A[j][i])){  valmax = abs(A[j][i]);  jMax = j;  }  }  if (jMax != i) {  for (int j = 0; j < f.size(); j++){  float t = A[i][j];  A[i][j] = A[jMax][j];  A[jMax][j] = t;  }  int t = xBack[i];  xBack[i] = xBack[jMax];  xBack[jMax] = t;  }  elimination(A, f, i);  }  backward(A, f);  vector<float> res(f.size());  for (int i = 0; i < res.size(); i++) {  res[xBack[i]] = f[i];  }  return res;  }  vector<float> dualGauss(vector<vector<float>> A, vector<float> f) {  vector<int> xBack(f.size());  for (int i = 0; i < xBack.size(); i++) {  xBack[i] = i;  }  for (int i = 0; i < xBack.size(); i++) {  int maxRow = i;  float valRow = abs(A[i][i]);  for (int j = i + 1; j < f.size(); j++){  if (valRow < abs(A[i][j])){  valRow = abs(A[i][j]);  maxRow = j;  }  }  int maxCol = i;  float valCol = abs(A[i][i]);  for (int j = i + 1; j < f.size(); j++){  if (valRow < abs(A[j][i])){  valRow = abs(A[j][i]);  maxRow = j;  }  }  if (valCol > valRow) {  if (maxCol != i) {  for (int j = 0; j < f.size(); j++){  float t = A[i][j];  A[i][j] = A[maxCol][j];  A[maxCol][j] = t;  }  int t = xBack[i];  xBack[i] = xBack[maxCol];  xBack[maxCol] = t;  }  }  else if (maxRow != i) {  for (int j = i; j < f.size(); j++){  float t = A[j][i];  A[j][i] = A[j][maxRow];  A[j][maxRow] = t;  }  float t = f[i];  f[i] = f[maxRow];  f[maxRow] = t;  }  elimination(A, f, i);  }  backward(A, f);  vector<float> res(f.size());  for (int i = 0; i < res.size(); i++) {  res[xBack[i]] = f[i];  }  return res;  }  bool diagCheck(vector<vector<float>> &A){  float total = 0.f;  for (int i = 0; i < A.size(); i++){  for (int j = 0; j < A.size(); j++){  if (i != j) {  total += abs(A[i][j]);  }  }  }  bool strict = false;  for (int i = 0; i < A.size(); i++){  if (abs(A[i][i]) < total) {  return false;  }  if (!strict && abs(A[i][i]) > total) strict = true;  }  return strict;  }  vector<float> subtraction(vector<float> v1, vector<float> v2){  vector<float> res(v1.size());  for (int i = 0; i < res.size(); i++){  res[i] = v1[i] - v2[i];  }  return res;  }  float norm(vector<float> v){  float sum = 0.f;  for (int i = 0; i < v.size(); i++){  sum += v[i] \* v[i];  }  return sqrtf(sum);  }  vector<float> randomVector(int n, float maxv = 100.f) {  vector<float> res(n);  for (int i = 0; i < n; i++)  res.at(i) = rand() / static\_cast<float>(RAND\_MAX) \* maxv;  return res;  }  vector<vector<float>> randomMatrix(size\_t n, float maxv = 100.f) {  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] = rand() / static\_cast<float>(RAND\_MAX) \* maxv;  return res;  }  vector<float> matrixVectorMultiplication(vector<vector<float>> m, vector<float> v){  vector<float> res;  if (m[0].size() == v.size()) {  for (int i = 0; i < m.size(); i++)  {  float sum = 0;  for (int j = 0; j < v.size(); j++)  {  sum += m[i][j] \* v[j];  }  res.push\_back(sum);  }  }  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;  }  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;  }  float checkResult(vector<vector<float>> &A, vector<float> &f, vector<float> &res){  Eigen::VectorXf check = to\_eigen\_m(A) \* to\_eigen\_v(res);  return (check - to\_eigen\_v(f)).norm();  // auto check = matrixVectorMultiplication(A, res);  // return norm(subtraction(check, f));  }  void test(vector<vector<float>> A, vector<float> f, vector<float> realX){  printf("%s\n", diagCheck(A) ? "Diagonal dominant" : "Normal");  auto gs = gauss(A, f);  auto col = colGauss(A, f);  auto row = rowGauss(A, f);  auto dual = dualGauss(A, f);  printf("Gauss: %e\n", checkResult(A, f, gs));  printf("Col: %e\n", checkResult(A, f, col));  printf("Row: %e\n", checkResult(A, f, row));  printf("Dual: %e\n", checkResult(A, f, dual));  printf("------------------\n");  printf("Gauss rel Dual: %e\n", norm(subtraction(gs, dual)) / norm(dual) \* 100);  printf("Col rel Dual: %e\n", norm(subtraction(col, dual)) / norm(dual) \* 100);  printf("Row rel Dual: %e\n", norm(subtraction(row, dual)) / norm(dual) \* 100);  printf("Gauss rel Real: %e\n", norm(subtraction(gs, realX)) / norm(realX) \* 100);  printf("Col rel Real: %e\n", norm(subtraction(col, realX)) / norm(realX) \* 100);  printf("Row rel Real: %e\n", norm(subtraction(row, realX)) / norm(realX) \* 100);  printf("Dual rel Real: %e\n", norm(subtraction(gs, realX)) / norm(realX) \* 100);  }  int main(){  srand(0xFace);  size\_t n = 5;  auto A = randomMatrix(n, 300.);  auto realX = randomVector(n, 300.);  auto f = matrixVectorMultiplication(A, realX);  ofstream mfile("matrix.txt");  for (int i = 0; i < n; i++){  for (int j = 0; j < n; j++){  mfile << A[i][j] << " ";  }  mfile << "\n";  }  mfile.close();  ofstream vfile("vector.txt");  for (int i = 0; i < n; i++){  vfile << realX[i] << " ";  }  vfile.close();  srand(0xDead0); // 0xDead0f  vector<float> pointsX;  vector<vector<float>> pointsY (3);  string datafile = "plot.dat";  string scriptfile = "script";  for (int i = 6; i < 100; i++) {  auto A = randomMatrix(i);  auto f = randomVector(i);  if (diagCheck(A)) printf("Diagonal dominant\n");  auto gs = gauss(A, f);  auto col = colGauss(A, f);  auto row = rowGauss(A, f);  auto dual = dualGauss(A, f);  float gsP = norm(subtraction(gs, dual)) / norm(dual) \* 100;  float colP = norm(subtraction(col, dual)) / norm(dual) \* 100;  float rowP = norm(subtraction(row, dual)) / norm(dual) \* 100;  pointsX.push\_back(i);  pointsY[0].push\_back(gsP);  pointsY[1].push\_back(colP);  pointsY[2].push\_back(rowP);  }  ofstream of(datafile);  for (size\_t si = 0; si < pointsY.size(); si++) {  for (size\_t i = 0; i < pointsY[0].size(); i++) {  of << pointsX[i] << " " << pointsY[si][i] << std::endl;  }  of << std::endl << std::endl;  }  of.flush();  of.close();    system(("gnuplot -c " + scriptfile).c\_str());  return 0;  } |

3 Тестирование

Для проверки генерировались матрицы и векторы, составленные из случайных чисел с плавающей точкой размера 50. Первая матрица не обладает свойством диагонального преобладания, вторая — обладает. В таблице 1 указаны относительные погрешности различных методов для этих матриц относительно метода с выбором главного элемента.

*Таблица* 1: Относительные погрешности методов

|  |  |  |  |
| --- | --- | --- | --- |
|  | Обычный | По столбцам | По строкам |
| 1 | 4.887390e+06 | 4.887328e+06 | 4.887319e+06 |
| 2 | 1.368176e+06 | 1.368176e+06 | 1.368176e+06 |

Далее, были последовательно сгенерированы матрицы и векторы размеров от 6 до 100, для каждого размера были вычислены относительные погрешности результатов методов относительно результата метода с выбором главного элемента. Из полученных значений был составлен график, показанный на рисунке 1.

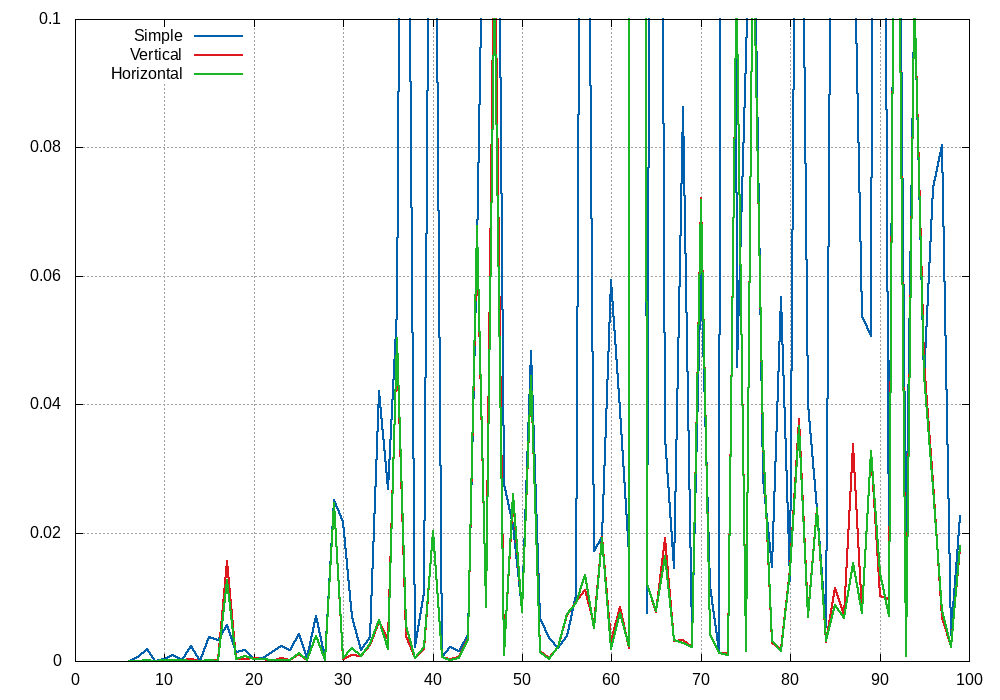


Рисунок 1: График относительных погрешностей

4 Вывод

Была создана и протестирована программа, выполняющая решение систем линейных алгебраических уравнений методом Гаусса с различными модификациями. Была исследована погрешность этих методов.