|  |  |
| --- | --- |
| **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 №3

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 методы Зейделя и Якоби для решения СЛАУ. Также необходимо проверить корректность методов и сравнить относительные погрешности обоих методов и количество занимаемых итераций при заданной точности.

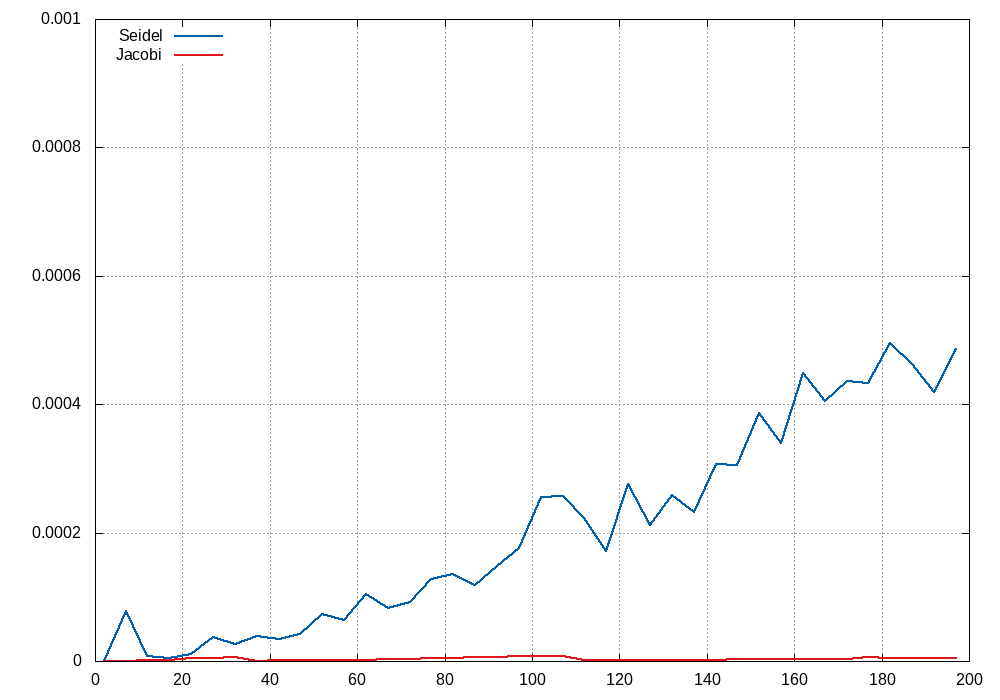
2 Программная реализация

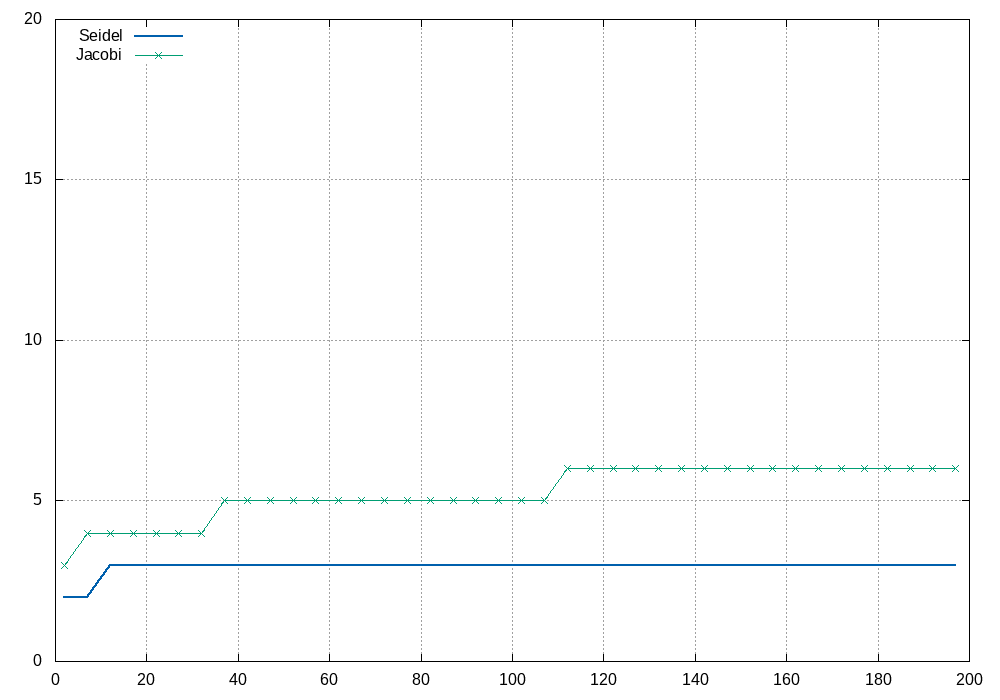
Лабораторная работа была выполнена на языке C++ с использованием пакета gnuplot для построения графиков. Исходный код основной программы приведён в листинге 1.

|  |
| --- |
| Листинг 1 |
| #include <iostream>  #include <cmath>  #include <vector>  #include <fstream>  using namespace std;  int iterations;  vector<float> randomVector(int n, float maxv = 100.f) {  vector<float> res(n);  for (int i = 0; i < n; i++)  res[i] = rand() / static\_cast<float>(RAND\_MAX) \* 10;  return res;  }  vector<vector<float>> randomMatrix(size\_t n, float maxv) {  vector<vector<float>> res(n, vector<float>(n));  for (size\_t i = 0; i < n; i++)  for (size\_t j = 0; j < n; j++)  if (i != j) res[i][j] = -10 + (rand() / ( RAND\_MAX / ( maxv +10) ) ) ;  else res[i][j] = 600.f;  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;  }  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);  }  float checkResult(vector<vector<float>> A, vector<float> f, vector<float> x){  auto check = matrixVectorMultiplication(A, x);  auto sub = subtraction(check, f);  // for (int i = 0; i < f.size(); i++){  // cout << check[i] << " " << f[i] << "\n";  // }  return norm(subtraction(check, f)) / norm(f);  }  vector<float> stepS(vector<vector<float>> A, vector<float> f, vector<float> x, int n){  auto next = f;  for (int i = 0; i < n; i++) {  for (int j = 0; j < i; j++) {  next[i] -= A[i][j] \* next[j];  }  for (int j = i + 1; j < n; j++) {  next[i] -= A[i][j] \* x[j];  }  next[i] /= A[i][i];  }  return next;  }  vector<float> stepJ(vector<vector<float>> A, vector<float> f, vector<float> x, int n){  auto next = f;  for (size\_t i = 0; i < n; i++) {  for (size\_t j = 0; j < n; j++) {  if (i == j) continue;  next[i] -= A[i][j] \* x[j];  }  next[i] /= A[i][i];  }  return next;  }  vector<float> seidel(vector<vector<float>> A, vector<float> f, int n){  auto eps = 0.001;    vector<float> curApprox(n);  vector<float> nextApprox(n);  for (size\_t i = 0; i < n; i++)  nextApprox[i] = 0.f;  int iter = 0;  do {  curApprox = nextApprox;  nextApprox = stepS(A, f, curApprox, n);  iter++;  } while (norm(subtraction(nextApprox ,curApprox)) > 1e-3);  iterations = iter;  // cout << "Итераций по методу Зейделя " << iter << "\n";  return nextApprox;  }  vector<float> jacobi(vector<vector<float>> A, vector<float> f, int n){  vector<float> curApprox(n);  vector<float> nextApprox(n);  for (size\_t i = 0; i < n; i++)  nextApprox[i] = 0.f;  int iter = 0;  do {  curApprox = nextApprox;  nextApprox = stepJ(A, f, curApprox, n);  iter++;  } while (norm(subtraction(nextApprox ,curApprox)) > 1e-5);  iterations = iter;  // cout << "Итераций по методу Якоби " << iter << "\n";  return nextApprox;  }  int main()  {  // int n = 10;  // auto A = randomMatrix(n, 10.);  // auto realX = randomVector(n, 10.);  // 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();  // vector<float> res = seidel(A, f, n);  // cout << checkResult(A, f, res) << "\n";  // vector<float> res2 = jacobi(A, f, n);  // cout << checkResult(A, f, res2);  string datafile = "plot.dat";  string datafile2 = "plot2.dat";  string scriptfile = "script";  string scriptfile2 = "script2";  vector<float> pointsX;  vector<vector<float>> pointsY(2);  vector<vector<float>> pointsY2(2);  for (int i = 2; i < 200; i+=5) {  auto A = randomMatrix(i, 10.f);  auto f = randomVector(i, 10.f);  vector<float> res = seidel(A, f, i);  pointsY2[0].push\_back(iterations);  // cout << checkResult(A, f, res) << "\n";  vector<float> res2 = jacobi(A, f, i);  pointsY2[1].push\_back(iterations);  // cout << checkResult(A, f, res2);  pointsX.push\_back(i);  pointsY[0].push\_back(checkResult(A, f, res));  pointsY[1].push\_back(checkResult(A, f, res2));  }  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();  ofstream of2(datafile2);  for (size\_t si = 0; si < pointsY.size(); si++) {  for (size\_t i = 0; i < pointsY[0].size(); i++) {  of2 << pointsX[i] << " " << pointsY2[si][i] << std::endl;  }  of2 << std::endl << std::endl;  }  of2.flush();  of2.close();    system(("gnuplot -c " + scriptfile).c\_str());  system(("gnuplot -c " + scriptfile2).c\_str());    return 0;  } |

3 Тестирование

На рисунке 1 представлен график относительной погрешности методов Зейделя и Якоби соответственно на примерах матриц размера от 2 до 200. На рисунке 2 представлен график изменения количества итераций.

Рисунок 1 - График относительных погрешностей методов

 Рисунок 2 - График изменения количества итераций

4 Вывод

В результате работы были реализованы методы Зейделя и Якоби для решения СЛАУ, а так же был произведен анализ относительных погрешностей и скоростей вычисления. По итогам тестирования, можно сделать вывод, что метод Зейделя быстрее метода Якоби, но метод Якоби является более точным.