|  |  |
| --- | --- |
| **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 №4

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 однопараметрический метод решения СЛАУ для симметричной положительной матрицы, а также исследовать скорость метода в зависимость от параметра τ.

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 <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, float maxv = 20.f) {  vector<vector<float>> res(n, vector<float>(n));  float num = rand() / static\_cast<float>(RAND\_MAX) \* maxv / 100;  for (int i = 0; i < n; i++)  for (int j = 0; j < n; j++)  if (i == j) res[i][j] = num;  else res[i][j] = 0;  return res;  }  vector<float> randomVector(int n, float maxv = 20.f) {  vector<float> res(n);  for (int i = 0; i < n; i++)  res[i] = rand() / static\_cast<float>(RAND\_MAX) \* maxv / 100;  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::VectorXf oneParam(Eigen::MatrixXf Asymn, Eigen::VectorXf f, float e, float t, int &count, int n){  Eigen::MatrixXf tAsymn = t \* Asymn;  vector<vector<float>> arr (n, vector<float> (n));  for (int i = 0; i < n; i++){  for (int j = 0; j < n; j++){  if (i == j) arr[i][j] = 1;  else arr[i][j] = 0;  }  }  Eigen::MatrixXf E = to\_eigen\_m(arr);  Eigen::MatrixXf G = E - tAsymn;  auto g = f \* t;  vector<float> v(n);  for (int i = 0; i < n; i++){  v[i] = 0;  }  Eigen::VectorXf u = to\_eigen\_v(v);  count = 0;  while(true){  count++;  auto uC = u;  u = G \* u + g;  if ((u - uC).norm() < e) break;  }  return u;  }  int main(){    size\_t n = 100;  size\_t steps = 20;  auto arr = randomMatrix(n, 50.f);  Eigen::MatrixXf Asymn = to\_eigen\_m(arr);  Eigen::VectorXf f = to\_eigen\_v(randomVector(n, 50.f));  auto l = Asymn.eigenvalues();  float eigen\_max = l[0].real();  float eigen\_min = l[0].real();  for (size\_t i = 0; i < l.size(); i++) {  if (eigen\_max < l[i].real()) eigen\_max = l[i].real();  if (eigen\_min > l[i].real()) eigen\_min = l[i].real();  }  auto tOpt = 2 / (eigen\_max + eigen\_min);  auto threshold = 2 / eigen\_max;  vector<float> tau;  float step = threshold / steps;  float val = 0.1;  while (val < threshold){  tau.push\_back(val);  val += step;  }  int count;  Eigen::VectorXf u = oneParam(Asymn, f, 0.00001, tOpt, count, n);  vector<int> counts;  Eigen::MatrixXf check = Asymn \* u;  for (int i = 0; i < tau.size(); i++){  oneParam(Asymn, f, 0.00001, tau[i], count, n);  auto c = count;  counts.push\_back(c);  }  string datafile = "plot.dat";  string scriptfile = "script";  ofstream of(datafile);  for (size\_t i = 0; i < counts.size(); i++) {  of << tau[i] << " " << counts[i] << std::endl;  }  of.flush();  of.close();  system(("gnuplot -c " + scriptfile).c\_str());  cout << "Optimal: " << tOpt << " " << count;  return 0;  } |

3 Тестирование

Для тестирования необходимо сгенерировать рандомную матрицу и вектор заданного размера. На рисунке 1 представлен график зависимости скорости (в итерациях) от значения параметра τ.

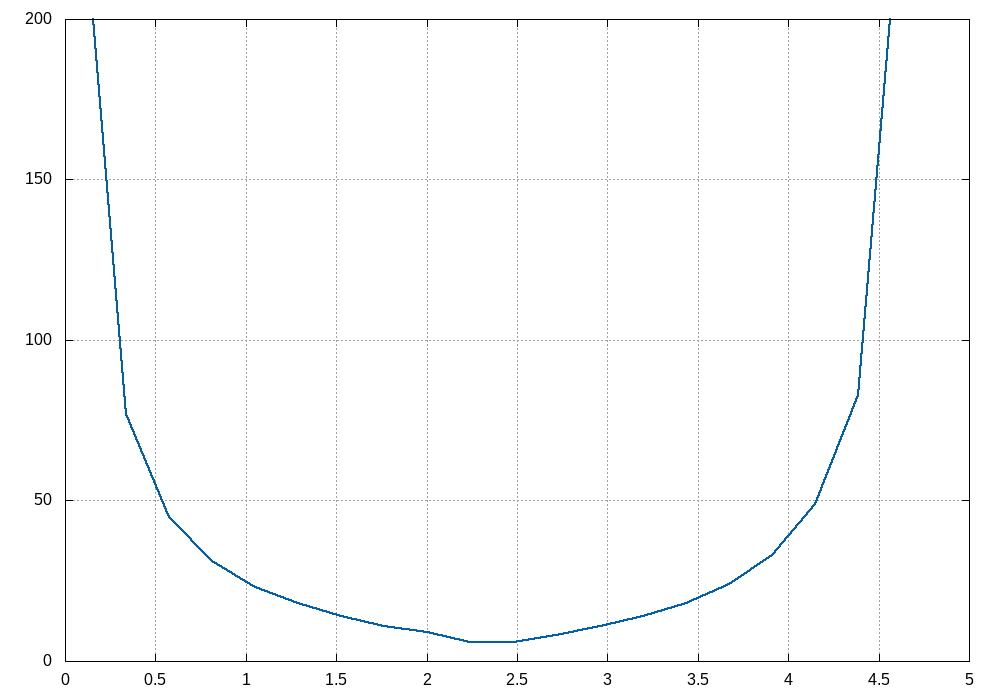


Рисунок 1 - График относительных погрешностей методов

4 Вывод

В результате работы был реализован однопараметрический метод решения СЛАУ для симметричной положительной матрицы, а так же исследована зависимость скорости от значения параметра τ.