Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Методы оптимизации

Лабораторная работа №2

Факультет: прикладной математики и информатики

Группа: ПМ-63

Студенты: Кожекин М.В.

Утюганов Д.С.

Преподаватели: Чимитова Е. В.

Новосибирск

2019

1. Цель работы

Ознакомиться с методами поиска минимума функции n переменных в оптимизационных задачах без ограничений.

1. Задание

Реализовать методы дихотомии, золотого сечения и метод Фибоначчи исследовать их сходимость и провести сравнение по числу вычислений функции для достижения заданной точности ε от 10-1 до 10-7. Построить график зависимости количества вычис­лений минимизируемой функции от десятичного логарифма задаваемой точности ε.

Реализовать алгоритм поиска интервала, содержащего минимум.

Реализовать два метода поиска (Розенброк + Бройден) экстремума функции (разного порядка) Включить в реализуемый алгоритм собственную процедуру, реализующую одномерный поиск по направлению. Методы поиска для самостоятельной реализации выбираются студентом в зависимости от уровня сложности. Выбранные методы должны иметь разный порядок (например, метод Гаусса (нулевого порядка) - 1 балл и метод Ньютона (второго порядка) - 3 балла, итого 9 баллов).

С использованием разработанного программного обеспечения исследовать алгоритмы на квадратичной функции , функции Розенброка и на заданной в соответствии с вариантом тестовой функции, осуществляя спуск из различных исходных точек (не менее двух). Исследовать сходимость алгоритма, фиксируя точность определения минимума/максимума, количество итераций метода и количество вычислений функции в зависимости от задаваемой точности поиска. Результатом выполнения данного пункта должны быть выводы об объёме вычислений в зависимости от задаваемой точности и начального приближения.

Построить траекторию спуска различных алгоритмов из одной и той же исходной точки с одинаковой точностью. В отчете наложить эту траекторию на рисунок с линиями равного уровня заданной функции.

Реализовать метод квадратичной интерполяции (метод парабол) для приближенного нахождения экстремума при одномерном поиске. Исследовать влияние точности одномерного поиска на общее количество итераций и вычислений функции при разных методах одномерного поиска.

1. Исследования

* Функция из варианта 1

**Первая таблица**

Метод Розенброка

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| E | Итераций | Вычислений f | x0 | x | f(x,y) |
| 0.0010000 | 2 | 155 | 1.0000000 0.0000000 | 1.1734635 1.2118397 | -3.5001766 |
| 0.0001000 | 3 | 221 | 1.0000000 0.0000000 | 1.1735217 1.2118328 | -3.5001767 |
| 0.0000100 | 3 | 251 | 1.0000000 0.0000000 | 1.1735463 1.2119082 | -3.5001765 |
| 0.0000010 | 3 | 280 | 1.0000000 0.0000000 | 1.1735489 1.2119148 | -3.5001765 |
| 0.0000001 | 3 | 307 | 1.0000000 0.0000000 | 1.1735490 1.2119154 | -3.5001765 |

Метод Бройдена

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| E | Итераций | Вычислений f | x0 | x | f(x,y) |
| 0.0010000 | 3 | 164 | 1.0000000 0.0000000 | 1.1731872 1.2109481 | -3.5001777 |
| 0.0001000 | 3 | 178 | 1.0000000 0.0000000 | 1.1731830 1.2109392 | -3.5001777 |
| 0.0000100 | 3 | 193 | 1.0000000 0.0000000 | 1.1731811 1.2109387 | -3.5001777 |
| 0.0000010 | 4 | 276 | 1.0000000 0.0000000 | 1.1732915 1.2109457 | -3.5001777 |
| 0.0000001 | 4 | 296 | 1.0000000 0.0000000 | 1.1732915 1.2109457 | -3.5001777 |

**Вторая таблица**

Метод Розенброка

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i | (xi, yi) | f(xi, yi) | S | λ | xi – xi-1 | f(xi) – f(xi-1) |
| 1 | 1.1961 1.2133 | -3.4999 | 0.1596 0.9872  1.0000 0.0000 | 0.1961 | 0.1961 1.2133 | -1.6074 |
| 2 | 1.1735 1.2118 | -3.5002 | -0.9980 -0.0632  1.0000 0.0000 | -0.0015 | -0.0226 -0.0014 | -0.0003 |
| 3 | 1.1735 1.2118 | -3.5002 | 0.0316 -0.9995  1.0000 0.0000 | 0.0001 | 0.0000 -0.0000 | -0.0000 |

Метод Бройдена

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i | (xi, yi) | f(xi, yi) | λ | xi – xi-1 | f(xi) – f(xi-1) | ∇ f(xi, yi) | A |
| 1 | 1.0507 1.2024 | -3.4925 | -0.8599 | 0.0507 1.2024 | -1.5999 | -0.1271 0.0055 | 0.9515 0.0823  0.0823 0.8605 |
| 2 | 1.1729 1.2082 | -3.5002 | -1.0142 | 0.1222 0.0058 | -0.0077 | 0.0004 -0.0073 | 0.9744 0.1557  0.1557 1.0959 |
| 3 | 1.1732 1.2109 | -3.5002 | -0.3427 | 0.0003 0.0027 | -0.0000 | -0.0001 0.0000 | 0.9699 0.0987  0.0987 0.3787 |

* Квадратичная функция

100(y – x)2 + (1 – x)2

**Первая таблица**

Метод Розенброка

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| E | Итераций | Вычислений f | x0 | x | f(x,y) |
| 0.0010000 | 3 | 179 | 1.0000000 0.0000000 | 0.0190945 0.0092740 | 0.9718198 |
| 0.0001000 | 3 | 193 | 1.0000000 0.0000000 | 0.0195284 0.0097051 | 0.9709742 |
| 0.0000100 | 4 | 287 | 1.0000000 0.0000000 | 0.0194972 0.0195010 | 0.9613858 |
| 0.0000010 | 7 | 544 | 1.0000000 0.0000000 | 1.0000000 0.9999999 | 0.0000000 |
| 0.0000001 | 7 | 590 | 1.0000000 0.0000000 | 1.0000000 1.0000000 | 0.0000000 |

Метод Бройдена

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| E | Итераций | Вычислений f | x0 | x | f(x,y) |
| 0.0010000 | 3 | 145 | 1.0000000 0.0000000 | 0.9999971 0.9999976 | 0.0000000 |
| 0.0001000 | 3 | 161 | 1.0000000 0.0000000 | 0.9999918 0.9999922 | 0.0000000 |
| 0.0000100 | 3 | 173 | 1.0000000 0.0000000 | 0.9999903 0.9999901 | 0.0000000 |
| 0.0000010 | 3 | 182 | 1.0000000 0.0000000 | 0.9999980 0.9999976 | 0.0000000 |
| 0.0000001 | 3 | 187 | 1.0000000 0.0000000 | 1.0000000 0.9999999 | 0.0000000 |

**Вторая таблица**

Метод Розенброка

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i | (xi, yi) | f(xi, yi) | S | λ | xi – xi-1 | f(xi) – f(xi-1) |
| 1 | 0.0098 0.0098 | 0.9805 | -1.0000 0.0099  -1.0000 -0.0000 | -0.9902 | -0.9902 0.0098 | -99.0195 |
| 2 | 0.0195 0.0097 | 0.9710 | 1.0000 -0.0098  -1.0000 -0.0000 | -0.0097 | 0.0097 -0.0001 | -0.0095 |
| 3 | 0.0195 0.0097 | 0.9710 | 0.0049 1.0000  -1.0000 -0.0000 | -0.0001 | 0.0000 0.0000 | -0.0000 |

Метод Бройдена

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i | (xi, yi) | f(xi, yi) | λ | xi – xi-1 | f(xi) – f(xi-1) | ∇ f(xi, yi) | A |
| 1 | 0.4876 0.5124 | 0.3237 | -0.0026 | -0.5124 0.5124 | -99.6763 | -5.9729 4.9482 | 0.4988 0.4987  0.4987 0.5037 |
| 2 | 0.9997 0.9997 | 0.0000 | -1.0020 | 0.5121 0.4873 | -0.3237 | 0.0104 -0.0109 | 0.5000 0.5000  0.5000 0.5050 |
| 3 | 1.0000 1.0000 | 0.0000 | -1.0076 | 0.0002 0.0003 | -0.0000 | -0.0001 0.0001 | 0.5000 0.5000  0.5000 0.5050 |

* функция Розенброка

100(y – x2)2 + (1 – x)2

**Первая таблица**

Метод Розенброка

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| E | Итераций | Вычислений f | x0 | x | f(x,y) |
| 0.0010000 | 3 | 179 | 1.0000000 0.0000000 | 0.0190945 0.0092740 | 0.9718198 |
| 0.0001000 | 3 | 193 | 1.0000000 0.0000000 | 0.0195284 0.0097051 | 0.9709742 |
| 0.0000100 | 4 | 287 | 1.0000000 0.0000000 | 0.0194972 0.0195010 | 0.9613858 |
| 0.0000010 | 7 | 544 | 1.0000000 0.0000000 | 1.0000000 0.9999999 | 0.0000000 |
| 0.0000001 | 7 | 590 | 1.0000000 0.0000000 | 1.0000000 1.0000000 | 0.0000000 |

Метод Бройдена

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| E | Итераций | Вычислений f | x0 | x | f(x,y) |
| 0.0010000 | 3 | 145 | 1.0000000 0.0000000 | 0.9999971 0.9999976 | 0.0000000 |
| 0.0001000 | 3 | 161 | 1.0000000 0.0000000 | 0.9999918 0.9999922 | 0.0000000 |
| 0.0000100 | 3 | 173 | 1.0000000 0.0000000 | 0.9999903 0.9999901 | 0.0000000 |
| 0.0000010 | 3 | 182 | 1.0000000 0.0000000 | 0.9999980 0.9999976 | 0.0000000 |
| 0.0000001 | 3 | 187 | 1.0000000 0.0000000 | 1.0000000 0.9999999 | 0.0000000 |

**Вторая таблица**

Метод Розенброка

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i | (xi, yi) | f(xi, yi) | S | λ | xi – xi-1 | f(xi) – f(xi-1) |
| 1 | 0.1612 0.0260 | 0.7036 | -0.9995 0.0309  -1.0000 -0.0000 | -0.8388 | -0.8388 0.0260 | -99.2964 |
| 2 | 0.2087 0.0246 | 0.6622 | 0.9996 -0.0293  0.0000 -1.0000 | -0.0450 | 0.0475 -0.0014 | -0.0414 |
| 3 | 0.2063 0.0426 | 0.6299 | -0.1291 0.9916  0.0000 -1.0000 | -0.0023 | -0.0023 0.0180 | -0.0323 |
| 4 | 0.2065 0.0427 | 0.6297 | 0.7386 0.6741  0.0000 -1.0000 | -0.0010 | 0.0001 0.0001 | -0.0002 |
| 5 | 0.2450 0.0601 | 0.5700 | 0.9115 0.4112  0.0000 -1.0000 | 0.0522 | 0.0385 0.0174 | -0.0597 |
| 6 | 0.3729 0.1391 | 0.3932 | 0.8508 0.5255  0.0000 -1.0000 | 0.1403 | 0.1279 0.0790 | -0.1768 |
| 7 | 0.4580 0.2099 | 0.2937 | 0.7689 0.6393  0.0000 -1.0000 | 0.1001 | 0.0851 0.0708 | -0.0995 |
| 8 | 0.5513 0.3041 | 0.2013 | 0.7037 0.7105  0.0000 -1.0000 | 0.1213 | 0.0933 0.0942 | -0.0924 |
| 9 | 0.6333 0.4011 | 0.1345 | 0.6451 0.7641  0.0000 -1.0000 | 0.1164 | 0.0819 0.0970 | -0.0668 |
| 10 | 0.7104 0.5048 | 0.0839 | 0.5970 0.8022  0.0000 -1.0000 | 0.1196 | 0.0772 0.1037 | -0.0506 |
| 11 | 0.7801 0.6086 | 0.0484 | 0.5572 0.8304  0.0000 -1.0000 | 0.1167 | 0.0697 0.1038 | -0.0355 |
| 12 | 0.8421 0.7092 | 0.0249 | 0.5248 0.8512  0.0000 -1.0000 | 0.1113 | 0.0620 0.1006 | -0.0234 |
| 13 | 0.8957 0.8023 | 0.0109 | 0.4987 0.8667  0.0000 -1.0000 | 0.1021 | 0.0536 0.0931 | -0.0140 |
| 14 | 0.9397 0.8830 | 0.0036 | 0.4786 0.8780  0.0000 -1.0000 | 0.0882 | 0.0440 0.0807 | -0.0072 |
| 15 | 0.9723 0.9454 | 0.0008 | 0.4632 0.8862  0.0000 -1.0000 | 0.0681 | 0.0326 0.0623 | -0.0029 |
| 16 | 0.9927 0.9855 | 0.0001 | 0.4534 0.8913  0.0000 -1.0000 | 0.0441 | 0.0204 0.0401 | -0.0007 |
| 17 | 0.9996 0.9992 | 0.0000 | 0.4499 0.8931  0.0000 -1.0000 | 0.0152 | 0.0069 0.0137 | -0.0001 |

Метод Бройдена

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i | (xi, yi) | f(xi, yi) | λ | xi – xi-1 | f(xi) – f(xi-1) | ∇ f(xi, yi) | A |
| 1 | 0.4944 0.2528 | 0.2625 | -0.0013 | -0.5056 0.2528 | -99.7375 | -2.6543 1.6617 | 0.2015 0.3999 0.3999 0.7997 |
| 2 | 0.4911 0.2459 | 0.2612 | -0.0258 | -0.0033 -0.0069 | -0.0013 | -1.9417 0.9407 | 0.0104 0.0149 0.0149 0.0243 |
| 3 | 0.6297 0.3829 | 0.1558 | -22.5309 | 0.1386 0.1370 | -0.1054 | 2.7000 -2.7318 | 0.2279 0.2503 0.2503 0.2790 |
| 4 | 0.7075 0.4812 | 0.1230 | -1.1368 | 0.0778 0.0983 | -0.0327 | 4.8941 -3.8720 | 0.1022 0.1284 0.1284 0.1609 |
| 5 | 0.6990 0.4879 | 0.0906 | -0.0017 | -0.0085 0.0067 | -0.0324 | -0.3975 -0.1462 | 0.3325 0.4699 0.4699 0.6692 |
| 6 | 0.8055 0.6387 | 0.0479 | -0.5298 | 0.1064 0.1508 | -0.0427 | 2.8446 -2.0073 | 0.2111 0.3105 0.3105 0.4599 |
| 7 | 0.9202 0.8386 | 0.0130 | -5.0138 | 0.1147 0.1999 | -0.0349 | 2.8341 -1.6267 | 0.2112 0.3072 0.3072 0.5336 |
| 8 | 0.9159 0.8385 | 0.0071 | -0.0429 | -0.0042 -0.0001 | -0.0059 | -0.0011 -0.0911 | 0.1551 0.2837 0.2837 0.5237 |
| 9 | 0.9722 0.9423 | 0.0016 | -2.1618 | 0.0563 0.1039 | -0.0055 | 1.0434 -0.5652 | 0.3894 0.7393 0.7393 1.4099 |
| 10 | 0.9861 0.9731 | 0.0002 | -1.2060 | 0.0139 0.0307 | -0.0013 | -0.2637 0.1197 | -1.7794 -3.3758 -3.3758 -6.3981 |
| 11 | 0.9864 0.9729 | 0.0002 | -0.0011 | 0.0003 -0.0001 | -0.0000 | 0.0086 -0.0181 | 0.2047 0.4024 0.4024 0.7964 |

1. Выводы

Хотя метод Розенброка и хорошо оптимизирует овражные функции, но в сравнении с методом Бройдена он хуже, т.к. при повышении точности он начинает значительно уступать – растёт как число итераций, так и количество вычислений целевой функции.

Это объясняется тем, что метод Бройдена на 2 порядка выше, т.е. использует информацию не только о функции, но и об обеих её производных.

Т.к. при вычислении приращения матрицы знаменатель дроби может обратиться в ноль, то метод нужно обновлять.

1. Исходный код

**head.h**

|  |
| --- |
| #pragma once  #define \_CRT\_SECURE\_NO\_WARNINGS  #include <fstream>  #include <iostream>  #include <vector>  #include <string>  #include <iomanip>  #include <functional>  #include <cmath>  using namespace std;  // float || double  typedef double real;  typedef vector <real> vector1D;  typedef vector <vector <real>> matrix2D;  // Умножение на константу  inline bool operator==(const vector1D& a, const vector1D& b) {  #ifdef \_DEBUG  if (a.size() != b.size())  throw std::exception();  #endif  for (int i = 0; i < a.size(); ++i)  if (a[i] != b[i])  return false;  return true;  }  // Сложение векторов  inline vector1D operator+(const vector1D& a, const vector1D& b) {  #ifdef \_DEBUG  if (a.size() != b.size())  throw std::exception();  #endif  vector1D result = a;  for (int i = 0; i < b.size(); i++)  result[i] += b[i];  return result;  }  // Сложение матриц  inline matrix2D operator+(const matrix2D& a, const matrix2D& b) {  #ifdef \_DEBUG  if (a.size() != b.size())  throw std::exception();  #endif  matrix2D result = a;  for (int i = 0; i < b.size(); i++)  for (int j = 0; j < b.size(); j++)  result[i][j] += b[i][j];  return result;  }  // Сложение матриц  inline matrix2D operator/(const matrix2D& a, const real& b) {  matrix2D result = a;  for (int i = 0; i < a.size(); i++)  for (int j = 0; j < a.size(); j++)  result[i][j] /= b;  return result;  }  // Вычитание векторов  inline vector1D operator-(const vector1D& a, const vector1D& b) {  #ifdef \_DEBUG  if (a.size() != b.size())  throw std::exception();  #endif  vector1D result = a;  for (int i = 0; i < b.size(); i++)  result[i] -= b[i];  return result;  }  inline vector1D operator-(const vector1D& a) {  vector1D result = a;  for (int i = 0; i < a.size(); i++)  result[i] = -result[i];  return result;  }  // Умножение матрицы на вектор  inline vector1D operator\*(const matrix2D& a, const vector1D& b) {  vector1D result = { 0.0, 0.0 };  for (int i = 0; i < a.size(); i++)  for (int j = 0; j < a.size(); j++)  result[i] += a[i][j] \* b[j];  return result;  }  // Умножение на константу  inline vector1D operator\*(const vector1D& a, double b) {  vector1D result = a;  for (int i = 0; i < result.size(); i++)  result[i] \*= b;  return result;  }  // Умножение на константу  inline vector1D operator\*(double b, const vector1D& a) {  return operator\*(a, b);  }  // Деление на константу  inline vector1D operator/(const vector1D& a, double b) {  vector1D result = a;  for (int i = 0; i < result.size(); i++)  result[i] /= b;  return result;  }  // Деление на константу  inline vector1D operator/(double b, const vector1D& a) {  return operator/(a, b);  }  // Скалярное произведение  inline real operator\*(const vector1D& a, const vector1D& b) {  #ifdef \_DEBUG  if (a.size() != b.size())  throw std::exception();  #endif  real sum = 0;  for (int i = 0; i < a.size(); i++)  sum += a[i] \* b[i];  return sum;  }  // Потоковый вывод вектора  inline std::ostream& operator<<(std::ostream& out, const vector1D& v) {  for (int i = 0; i < v.size() - 1; ++i)  out << v[i] << " ";  out << v.back();  return out;  }  // Потоковый вывод матрицы  inline std::ostream& operator<<(std::ostream& out, const matrix2D& v) {  for (int i = 0; i < v.size() - 1; ++i)  out << v[i] << " ";  out << v.back();  return out;  }  // Евклидова норма  real calcNormE(const vector1D &x) {  return sqrt(x\*x);  }  // Определитель матрицы  real det(const matrix2D &m) {  return m[0][0] \* m[1][1] - m[0][1] \* m[1][0];  } |

**main.cpp**

|  |
| --- |
| #include "head.h"  int fCalcCount;  struct methodResult  {  real E;  int iterationsCount;  int fCalcCount;  vector1D x0, x, xPrev, S, gradf;  matrix2D A;  real fx, fxPrev, lambda;  void printFirstResult(std::ofstream &fout);  };  // Вывод результатов в поток для 1 таблицы  void methodResult::printFirstResult(std::ofstream &fout) {  fout << E << "\t"  << iterationsCount << "\t"  << fCalcCount << "\t"  << x0 << "\t"  << x << "\t"  << fx << endl;  }  /\* Заданная целевая функция  \* x,y - аргументы функции  \*/  real f1(const vector1D &x) {  // - (2 / (1 + (((x - 1) / 2)^ 2) + (((y - 1) / 1)^ 2)) +3 / (1 + (((x - 2) / 3) ^ 2) + (((y - 3) / 2) ^ 2)))  // 1 вариант -3.5  int A1 = 2, A2 = 3, a1 = 1, a2 = 2, b1 = 2, b2 = 3, c1 = 1, c2 = 3, d1 = 1, d2 = 2;  // 2 вариант  //int A1 = 1, A2 = 3, a1 = 2, a2 = 1, b1 = 3, b2 = 1, c1 = 2, c2 = 1, d1 = 3, d2 = 2;  fCalcCount++;  return -(A1 / (1 + pow((x[0] - a1) / b1, 2) + pow((x[1] - c1) / d1, 2))  + A2 / (1 + pow((x[0] - a2) / b2, 2) + pow((x[1] - c2) / d2, 2)));  }  /\*  minimize -(2/(1 + ((x - 1)/2)^2 + ((y - 1) /1)^2) + 3/(1 + ((x - 2)/3)^2 + ((y - 3)/2)^2))  \*/  /\* Функция для исследования min=0 в точке (1,1)  \* x - вектор аргументов функции  \*/  real f2(const vector1D &x) {  fCalcCount++;  return 100 \* pow((x[1] - x[0]), 2) + pow((1 - x[0]), 2);  }  /\* Функция Розенброка min=0 в точке (1,1)  \* x - вектор аргументов функции  \*/  real f3(const vector1D &x) {  fCalcCount++;  return pow((1 - x[0]), 2) + 100 \* pow((x[1] - x[0] \* x[0]), 2);  }  /\* Поиск интервала, содержащего минимум  \* f - целевая одномерная функция  \* a,b - искомые границы отрезка  \* x0 - начальная точка  \* delta - точность  \* fileName - файл, в который будет сохранён результат  \*/  void interval(const function<real(const vector1D &x)> &f, real &a, real &b, vector1D &x, vector1D &S)  {  real lambda0 = 0.0;  real delta = 1.0e-8;  real lambda\_k\_minus\_1 = lambda0;  real f\_k\_minus\_1 = f(x + S \* lambda\_k\_minus\_1);  real lambda\_k;  real f\_k;  real lambda\_k\_plus\_1;  real f\_k\_plus\_1;  real h;  if (f(x + S \* lambda0) > f(x + S \* (lambda0 + delta)))  {  lambda\_k = lambda0 + delta;  h = delta;  }  else  {  lambda\_k = lambda0 - delta;  h = -delta;  }  f\_k = f(x + S \* lambda\_k);  while (true)  {  h \*= 2.0;  lambda\_k\_plus\_1 = lambda\_k + h;  f\_k\_plus\_1 = f(x + S \* lambda\_k\_plus\_1);  if (f\_k > f\_k\_plus\_1)  {  lambda\_k\_minus\_1 = lambda\_k;  f\_k\_minus\_1 = f\_k;  lambda\_k = lambda\_k\_plus\_1;  f\_k = f\_k\_plus\_1;  }  else  {  a = lambda\_k\_minus\_1;  b = lambda\_k\_plus\_1;  if (b < a)  swap(a, b);  return;  }  }  }  /\* Вычисление n-го числа Фибоначии \*/  inline real fib(int n)  {  real sqrt5 = sqrt(5.0), pow2n = pow(2.0, n);  return (pow(1.0 + sqrt5, n) / pow2n - pow(1.0 - sqrt5, n) / pow2n) / sqrt5;  }  /\* Определение коэффициента лямбда методом Фибоначчи  \* f - минимизируемая функция  \* x - начальное значение  \* S - базис  \* E - точность  \*/  real fibonacci(const function<real(const vector1D &x)> &f, vector1D &x, vector1D &S, real E)  {  real a, b;  interval(f, a, b, x, S);  int iter;  real len = fabs(a - b);  int n = 0;  while (fib(n) < (b - a) / E) n++;  iter = n - 3;  real lambda1 = a + (fib(n - 2) / fib(n)) \* (b - a);  real f1 = f(x + S \* lambda1);  real lambda2 = a + (fib(n - 1) / fib(n)) \* (b - a);  real f2 = f(x + S \* lambda2);  for (int k = 0; k < n - 3; k++)  {  if (f1 <= f2)  {  b = lambda2;  lambda2 = lambda1;  f2 = f1;  lambda1 = a + (fib(n - k - 3) / fib(n - k - 1)) \* (b - a);  f1 = f(x + S \* lambda1);  }  else  {  a = lambda1;  lambda1 = lambda2;  f1 = f2;  lambda2 = a + (fib(n - k - 2) / fib(n - k - 1)) \* (b - a);  f2 = f(x + S \* lambda2);  }  len = b - a;  }  lambda2 = lambda1 + E;  f2 = f(x + S \* lambda2);  if (f1 <= f2)  b = lambda1;  else  a = lambda1;  return (a + b) / 2.0;  }  /\* Метод Розенброка  \* f - оптимизиуремая функция  \* x0 - начальное приближение  \* E - точность  \*/  methodResult calcByRosenbrock(const function<real(const vector1D &x)> &f, const vector1D &x0, real E, const string &funcname) {  ofstream fout("report/Rosenbrock\_" + funcname + ".txt");  ofstream steps("report/Steps\_Rosenbrock\_" + funcname + ".txt");  fout << fixed << setprecision(4);  steps << fixed << setprecision(4);  methodResult result;  fCalcCount = 0;  vector1D xPrev, B, x = x0;  int maxiter = 100;  int iterationsCount = 0, count = 0;  real lambda1, lambda2;  matrix2D A(2);  A[0] = { 1.0, 0.0 };  A[1] = { 0.0, 1.0 };  // Начальные ортогональные направления  matrix2D S(2);  S[0] = { 1.0, 0.0 };  S[1] = { 0.0, 1.0 };  do {  xPrev = x;  // Минимализируем функцию в направлениях S^k\_1...S^k\_n  lambda1 = fibonacci(f, x, S[0], E);  x = x + S[0] \* lambda1;  lambda2 = fibonacci(f, x, S[1], E);  x = x + S[1] \* lambda2;  // Построение новых ортогональных направлений при  // сортировке лямбд в порядке убывания по абсолютным значениям  A[0] = S[0] \* lambda1 + S[1] \* lambda2;  if (fabs(lambda1 >= lambda2))  A[1] = S[1] \* lambda2;  else  A[1] = S[0] \* lambda1;  // Ортогонализация Грамма-Шмидта  S[0] = A[0] / calcNormE(A[0]);  B = A[1] - S[1] \* A[1] \* S[1];  if (calcNormE(B) > E)  S[1] = B / calcNormE(B);  iterationsCount++;  fout << iterationsCount << "\t"  << x << "\t"  << f(x) << "\t"  << S << "\t"  << lambda1<<"\t"  << x - xPrev << "\t"  << f(x) - f(xPrev) << "\t"  << A << endl;    steps << x << endl;  } while (abs(f(x) - f(xPrev)) > E && abs(calcNormE(x) - calcNormE(xPrev)) > E && iterationsCount < maxiter);  fout.close();  steps.close();  result.E = E;  result.iterationsCount = iterationsCount;  result.fCalcCount = fCalcCount;  result.x0 = x0;  result.x = x;  result.fx = f(x);  return result;  }  //-----------------------------------------------------------------------------  // Градиент  vector1D grad(const function<real(const vector1D &x)> &f, const vector1D &x1) {  const real eps = 1e-7;  const real fx1 = f(x1);  vector1D result(x1.size());  vector1D x = x1;  for (int i = 0; i < x.size(); ++i) {  x[i] += eps;  result[i] = (f(x) - fx1) / eps;  x[i] -= eps;  }  return result;  }  // Умножение вектора a' на вектор b и получение матрицы  matrix2D multTwoVectorsToMatrix(const vector1D &a, const vector1D &b) {  matrix2D tmp;  tmp.resize(a.size());  for (size\_t i = 0; i < tmp.size(); i++)  {  tmp[i] = a[i] \* b;  }  return tmp;  }  /\* Метод Бройдена  \* f - оптимизиуремая функция  \* x0 - начальное приближение  \* E - точность  \*/  methodResult calcByBroyden(const function<real(const vector1D &x)> &f, const vector1D &x0, real E, const string &funcname) {  ofstream fout("report/Broyden\_" + funcname + ".txt");  ofstream steps("report/Steps\_Broyden\_" + funcname + ".txt");  fout << fixed << setprecision(4);  steps << fixed << setprecision(4);  methodResult result;  fCalcCount = 0;  matrix2D A(2);  A[0] = { 1.0, 0.0 };  A[1] = { 0.0, 1.0 };  vector1D x = x0, xPrev = x;  vector1D gradf = grad(f, x);  int iterationsCount = 0, maxiter = 100;  do  {  if (det(A) <= 0 || !isfinite(det(A))) {  A[0] = { 1.0, 0.0 };  A[1] = { 0.0, 1.0 };  cout << "Матрица не положительно определена" << endl;  }  if (calcNormE(gradf) <= E || iterationsCount >= maxiter) {  result.E = E;  result.iterationsCount = iterationsCount;  result.fCalcCount = fCalcCount;  result.x0 = x0;  result.x = x;  result.fx = f(x);  return result;  }  vector1D ngradf = A \* gradf;  // получаем лямбду  real lambda = fibonacci(f, x, ngradf, E);  //cout << " lambda = "<< lambda << "\t";  vector1D dx = lambda \* ngradf;  xPrev = x;  x = x + dx;  vector1D old\_gradf = gradf;  gradf = grad(f, x);  vector1D dg = gradf - old\_gradf;  vector1D temp = dx - A \* dg;  // Находится очередное приближение матрицы H^(-1)  matrix2D dA = multTwoVectorsToMatrix(temp, temp) / (temp \* dg);  if (temp \* dg == 0) {  A[0] = { 1.0, 0.0 };  A[1] = { 0.0, 1.0 };  cout << "Деление на ноль" << endl;  }  else  A = A + dA;  iterationsCount++;    fout << iterationsCount << "\t"  << x << "\t"  << f(x) << "\t"  << lambda << "\t"  << x - xPrev << "\t"  << f(x) - f(xPrev) << "\t"  << gradf << "\t"  << A << endl;  steps << x << endl;  } while (abs(f(x) - f(xPrev)) > E && abs(calcNormE(x) - calcNormE(xPrev)) > E && iterationsCount < maxiter);  fout.close();  steps.close();  result.E = E;  result.iterationsCount = iterationsCount;  result.fCalcCount = fCalcCount;  result.x0 = x0;  result.x = x;  result.fx = f(x);  return result;  }  // Генерация первых таблиц  void makeFirstTables(const function<real(const vector1D &x)> &f, const vector1D &x0, const string &funcname) {  ofstream foutR("report/FirstTableRosenbrock\_" + funcname + ".txt");  ofstream foutB("report/FirstTableBroyden\_" + funcname + ".txt");  foutR << fixed << setprecision(7);  foutB << fixed << setprecision(7);  methodResult result;  for (double E = 1e-3; E >= 1e-7; E /= 10)  {  result = calcByRosenbrock(f, x0, E, funcname);  result.printFirstResult(foutR);  result = calcByBroyden(f, x0, E, funcname);  result.printFirstResult(foutB);  }  foutR.close();  foutB.close();  }  void main() {  vector1D x0 = { 1, 0 };  real E = 1e-4;  /\*makeFirstTables(f1, x0, "f1");  makeFirstTables(f2, x0, "f2");  makeFirstTables(f3, x0, "f3");\*/  calcByRosenbrock(f1, x0, E, "f1");  calcByRosenbrock(f2, x0, E, "f2");  calcByRosenbrock(f3, x0, E, "f3");  calcByBroyden(f1, x0, E, "f1");  calcByBroyden(f2, x0, E, "f2");  calcByBroyden(f3, x0, E, "f3");  } |