Министерство образования и науки РФ

Новосибирский государственный технический университет

Кафедра прикладной математики

Лабораторная работа №4

по методам оптимизации

Факультет: ПМИ

Группа: ПМ-63

Студенты: Крашенинник Н. А.

Пешкичева А. А.

Шепрут И. И.

Вариант: 4

Преподаватель: Чимитова Е.В.

Новосибирск

2019

1. Цель работы

Ознакомиться со статистическими методами поиска при реше­нии задач нелинейного программирования. Изучить методы случайного поиска при определении глобального экстремума функции.

1. Задание

Условие задачи:

Найти **максимум** заданной функции:



на области , .

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№ варианта** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **4** | 4 | 9 | 1 | 7 | 5 | 6 | 7 | -9 | 6 | -8 | -10 | -2 | 9 | -1 | 5 | -2 | -8 | -4 |

1. Исследования



**Вывод:** При одном и том же параметре m алгоритм2 по вычислительным затратам затрачивает на порядки меньше, однако и точность его меньше на порядки. Данные алгоритмы невозможно сравнить при равном значение параметра m. По исследованию замечены моменты, когда при примерно одинаковом количестве вычислений функции алгоритм1 выдаёт большую точность решения.



**Вывод:** Как при увеличении вероятности получения глобального максимума, так и при увеличении точности решения увеличивается число вычислений функции. Только при 19 миллионах вычислений функции получается значение с точностью 10^-6.

1. **Код**

|  |
| --- |
| #include "../2/funcs.h"  #include "../2/methods.h"  #include <random>  #include <iostream>  #include <vector>  using namespace std;  double func1(Vector v) {  vector<int> a = {7, -9, 6, -8, -10, -2};  vector<int> b = {9, -1, 5, -2, -8, -4 };  vector<int> c = { 4, 9, 1, 7, 5, 6 };  #define sqr(a) ((a)\*(a))  double sum = 0;  for (int i = 0; i < 6; i++) {  sum += c[i] / (1.0 + sqr(v(0) - a[i]) + sqr(v(1) - b[i]));  }  return -sum;  }  struct RandMethodsResult {  int fcount;  double value;  Vector point;  int N; // for simple Random Search  };  RandMethodsResult simpleRandomSearch(const double eps, const double p, const double x0, const double x1, const double y0, const double y1) {  /\*std::default\_random\_engine generator(0);  std::uniform\_real\_distribution<double> xaxis(x0, x1);  std::uniform\_real\_distribution<double> yaxis(y0, y1);\*/  const double g = 1.32471795724474602596090885447809;  const double ax = std::fmod(1.0 / g, 1.0);  const double ay = std::fmod(1.0 / g / g, 1.0);  const double seed = 0.5;  // Calculated necessary amount of points:  double peps = fabs(eps\*eps / ((x1 - x0)\*(y1 - y0)));  int64\_t N = log(1.0-p)/log(1.0-peps)+1; // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  cout << N << endl;  double valuemin = std::numeric\_limits<double>::infinity();  Vector pointmin(2);  for (int i = 0; i < N; i++) {  Vector point(2);  point(0) = x0 + std::fmod(seed + ax \* i, 1.0)\*(x1-x0);  point(1) = y0 + std::fmod(seed + ay \* i, 1.0)\*(y1-y0);  double value = func1(point);  if (value < valuemin) {  valuemin = value;  pointmin = point;  }  }  //set fcount!  RandMethodsResult result = { N, valuemin, pointmin, N };  return result;  }  RandMethodsResult alghoritm1(const int maxMissSearch, const double x0, const double x1, const double y0, const double y1) {  std::default\_random\_engine generator(0);  std::uniform\_real\_distribution<double> xaxis(x0, x1);  std::uniform\_real\_distribution<double> yaxis(y0, y1);  int count = 0;  auto f = setFunctionToCountCalls(&count, func1);  double valuemin = std::numeric\_limits<double>::infinity();  Vector pointmin(2);  bool searching = true;  int missSearch = 0; // current amount searching without result  auto argmin = bindArgmin(optimizeGoldenRatio);  while (missSearch < maxMissSearch) {  // new random point  Vector point(2);  point(0) = xaxis(generator);  point(1) = yaxis(generator);  // Deterministic method  //MethodResult optimizeHookeJeeves(const Function& f1, const ArgMinFunction& argmin, const Vector& x0, const double& eps) {  MethodResult r = optimizeBroyden(f, argmin, point, 0.001);  double value = f(r.answer);  if (value < valuemin) {  missSearch = 0;  pointmin = r.answer;  valuemin = value;  }  else {  missSearch++;  }  }  // set result  RandMethodsResult result = { count, valuemin, pointmin, 0 };  return result;  }  RandMethodsResult alghoritm2(int maxMissSearch, const double x0, const double x1, const double y0, const double y1) {  std::default\_random\_engine generator(0);  std::uniform\_real\_distribution<double> xaxis(x0, x1);  std::uniform\_real\_distribution<double> yaxis(y0, y1);  double valuemin = std::numeric\_limits<double>::infinity();  Vector pointmin(2);  int count = 0;  auto f = setFunctionToCountCalls(&count, func1);  auto argmin = bindArgmin(optimizeGoldenRatio);  // find first local minimum point  Vector point(2);  point(0) = (x1 - x0) / 2;  point(1) = (y1 - y0) / 2;  //MethodResult optimizeHookeJeeves(const Function& f1, const ArgMinFunction& argmin, const Vector& x0, const double& eps) {  MethodResult r = optimizeBroyden(f, argmin, point, 0.001);  pointmin = r.answer;  valuemin = f(pointmin);  // loop:  bool searching = true;  int missSearch = 0; // current amount searching without result  while (missSearch < maxMissSearch) {  missSearch = 0;  while (missSearch < maxMissSearch) {  // new random point  Vector point(2);  point(0) = xaxis(generator);  point(1) = yaxis(generator);  // comparison with pointmin  double value = f(point);  if (value < valuemin) {  // Deterministic method with "point"  //MethodResult optimizeHookeJeeves(const Function& f1, const ArgMinFunction& argmin, const Vector& x0, const double& eps) {  MethodResult r = optimizeBroyden(f, argmin, point, 0.001);  pointmin = r.answer;  valuemin = func1(pointmin);  }  else {  missSearch++;  }  }  }  // set result  RandMethodsResult result = { count, valuemin, pointmin, 0 };  return result;  }  double bestSolution = 0;  void testSimpleRandom() {  std::ofstream fout("research1.txt");  std::vector<double> epss = { 1, 0.5, 0.1, 0.05, 0.01, 0.005};  std::vector<double> ps = { 0.1, 0.3, 0.5, 0.7, 0.9 };  const double x0 = -10, x1 = 10;  const double y0 = -10, y1 = 10;  fout << "\t";  for (double p : ps) fout << p << "\t";  fout << std::endl;  for (double eps : epss) {  fout << eps << "\t";  for (double p : ps) {  RandMethodsResult r = simpleRandomSearch(eps, p, x0, x1, y0, y1);  fout << r.N << " " << r.value << " " << r.point(0) << " " << r.point(1) << "\t";  }  fout << std::endl;  }  fout.close();  }  void testAlgs() {  std::ofstream fout("research2.txt");  std::vector<int> ms = { 1, 3, 5, 7, 10, 50, 100, 500, 1000, 5000, 10000 };  const double x0 = -10, x1 = 10;  const double y0 = -10, y1 = 10;  fout << "m\ta1\_fcount\ta2\_fcount\ta1\_value\ta2\_value\ta1\_prec\ta2\_prec\n";  for (int maxMissSearch : ms) {  RandMethodsResult a1 = alghoritm1(maxMissSearch, x0, x1, y0, y1);  RandMethodsResult a2 = alghoritm2(maxMissSearch, x0, x1, y0, y1);  fout << maxMissSearch << "\t" << a1.fcount << "\t" << a2.fcount  << "\t" << a1.value << "\t" << a2.value  << "\t" << a1.value - bestSolution << "\t" << a2.value - bestSolution;  // precission of found result!?  fout << std::endl;  }  fout.close();  }  int main() {  const double x0 = -10, x1 = 10;  const double y0 = -10, y1 = 10;  Vector point(2);  point << -9, -1;  bestSolution = func1(optimizeBroyden(func1, bindArgmin(optimizeGoldenRatio), point, 1e-14).answer);  testSimpleRandom();  testAlgs();  return 0;  } |