Решение задачи многомерной минимизации функции

Владимир Руцкий, 3057/2

31 марта 2009 г.

1 Постановка задачи

Требуется найти с наперёд заданной точностью минимум многомерной функции $f(x)$ в некоторой области:

$$\min f(x), \quad x \in \mathbb{R}^n,$$

используя *метод градиентного спуска* и *генетический алгоритм*.

Исходная функция: $f(x) = x_1^3 + 2x_2 + 4\sqrt{2 + x_1^2 + x_2^2}$, заданная на $\mathbb{R}^2$.

2 Исследование применимости методов

2.1 Метод градиентного спуска

Метод градиентного спуска основывается на том, что для гладкой выпуклой функции градиент функции в точке направлен в сторону увеличения функции. Используя этот факт можно построить итерационный процесс. Выберем начальное приближение минимума, далее построим последовательность точек, в которой каждая следующая точка выбирается на луче противоположном градиенту в текущей точке:

$$x_{k+1} = x_k - \lambda_k \nabla f(x_k).$$

Шаг на который двигается текущая точка за одну итерацию равен λ_k и может задаваться различными способами, например:

1. $\lambda_k = \mathrm{const}$, фиксированный шаг;
2. $\lambda_k = c\lambda_{k-1}, \quad 0 < c < 1$, равномерно уменьшающийся шаг;
3. $\lambda_k \in (\,0, q\,): \quad f(x_k - \lambda_k \nabla f(x_k)) = \min\limits_{0<\lambda<q} f(x_k - \lambda \nabla f(x_k))$, в качестве следующей точки выбирается точка в которой достигается минимум на отрезке уменьшения функции, направленным против градиента.

Для того, чтобы к функции можно было применить указанный выше итерационный процесс, необходимо, чтобы функция была гладкой: $f \in C^{1,1}$.

Критерием остановки итерационного является событие, когда следующая точка находится от предыдущей на расстоянии меньше либо равном ε:

$$||x_{k+1} - x_k|| < \varepsilon.$$

Исходная функция в исследуемой области удовлетворяет необходимым для сходимости метода градиентного спуска условиям, указанным в 2.1.

2.2 Генетический алгоритм

3 Описание алгоритма

3.1 Метод градиентного спуска

В используемой реализации алгоритма λ_k выбирается таким согласно последнему методу, указанному в 2.1:

$$\lambda_k \in (\,0, q\,): \quad f(x_k - \lambda_k \nabla f(x_k)) = \min_{0<\lambda<q} f(x_k - \lambda \nabla f(x_k)),$$

значение λ_k ищется методом золотого сечения.

3.2 Генетический алгоритм

4 Код программы

4.1 Метод градиентного спуска

Исходный код 1: Градиентный спуск

```
/*
 * gradient_descent.hpp
 * Searching multidimensional function minimum with gradient descent algorithm.
 * Vladimir Rutsky <altsysrq@gmail.com>
 * 29.03.2009
 */

#ifndef NUMERIC_GRADIENT_DESCENT_HPP
#define NUMERIC_GRADIENT_DESCENT_HPP

#include "numeric_common.hpp"

#include <boost/assert.hpp>
#include <boost/concept/assert.hpp>
#include <boost/concept_check.hpp>
#include <boost/bind.hpp>
#include <boost/function.hpp>

#include "golden_section_search.hpp"
#include "lerp.hpp"

namespace numeric
{
namespace gradient_descent
{
  template< class Func, class FuncGrad, class V, class PointsOut >
  inline
  ublas::vector<typename V::value_type>
    find_min( Func function, FuncGrad functionGrad,
              V const &startPoint,
              typename V::value_type precision,
              typename V::value_type step,
              PointsOut pointsOut )
  {
    // TODO: Now we assume that vector's coordinates and function values are same scalar
        types.
    // TODO: Assert on correctness of 'ostr'.

    BOOST_CONCEPT_ASSERT((ublas::VectorExpressionConcept<V>));

    typedef typename V::value_type              scalar_type;
    typedef ublas::vector<scalar_type>          vector_type;
    typedef ublas::scalar_traits<scalar_type>   scalar_traits_type;

    BOOST_CONCEPT_ASSERT((boost::UnaryFunction<Func,       scalar_type, vector_type>));
    BOOST_CONCEPT_ASSERT((boost::UnaryFunction<FuncGrad, vector_type, vector_type>));

    BOOST_ASSERT(precision > 0);

    // Setting current point to start point.
    vector_type x = startPoint;

    *pointsOut++ = x;

    size_t iteration = 0;
    while (true)
    {
      // Searching next point in direction opposite to gradient.
      vector_type const grad = functionGrad(x);

      scalar_type const gradNorm = ublas::norm_2(grad);
      if (scalar_traits_type::equals(gradNorm, 0))
      {
        // Function gradient is almost zero, found minimum.
        return x;
      }

      vector_type const dir = -grad / gradNorm;
      BOOST_ASSERT(scalar_traits_type::equals(ublas::norm_2(dir), 1));

```

```
      vector_type const s0 = x;
      vector_type const s1 = s0 + dir * step;

      typedef boost::function<scalar_type ( scalar_type )> function_bind_type;
      function_bind_type functionBind =
          boost::bind<scalar_type>(function, boost::bind<vector_type>(Lerp<scalar_type,
              vector_type>(0.0, 1.0, s0, s1), _1));
      scalar_type const section = golden_section::find_min<function_bind_type,
          scalar_type>(functionBind, 0.0, 1.0, precision);
      BOOST_ASSERT(0 <= section && section <= 1);

      // debug
      /*
      std::cout << "x=";
      output_vector_coordinates(std::cout, x);
      std::cout << "f(x0) = " << function(s0 + dir * step * 0) << std::endl;
      std::cout << "f(x)  = " << function(s0 + dir * step * section) << std::endl;
      std::cout << "f(x1) = " << function(s0 + dir * step * 1) << std::endl;
      std::cout << "section=" << section << std::endl; // debug
      */
      // end of debug

      vector_type const nextX = s0 + dir * step * section;
      if (ublas::norm_2(x - nextX) < precision)
      {
        // Next point is equal to current (with precision), seems found minimum.
        return x;
      }

      // Moving to next point.
      x = nextX;
      *pointsOut++ = x;

      ++iteration;

      // debug
      if (iteration > 100)
      {
        std::cerr << "Too many iterations!\n";
        break;
      }
      // end of debug
    }

    return x;
  }
} // End of namespace 'gradient_descent'.
} // End of namespace 'numeric'.

#endif // NUMERIC_GRADIENT_DESCENT_HPP
```

4.2 Генетический алгоритм

5 Результаты решения

5.1 Метод градиентного спуска

Результаты решения приведены в таблице 1.

Начальной точкой была выбрана точка (2.5, 2.5) , шаг для поиска минимума методом золотого сечения был равен 0.5 .

Таблица 1: Результаты работы алгоритма градиентного спуска

Точность	Количество шагов	x	$f(x)$
1e-03	12	(4.61855e-06, -0.81648024)	4.89897949
1e-04	12	(-8.49295e-07, -0.81646860)	4.89897949
1e-05	13	(1.20244e-07, -0.81649645)	4.89897949
1e-06	13	(4.11864e-07, -0.81649654)	4.89897949
1e-07	14	(1.41643e-08, -0.81649657)	4.89897949
1e-08	17	(3.07062e-09, -0.81649657)	4.89897949

5.2 Генетический алгоритм

6 Обоснование достоверности полученного результата

6.1 Генетический алгоритм