

Владимир Руцкий, 3057/2

## Код программы

## Исходный код 1: Барьерный метод

```
2
    *\ barrier\_\ method.hpp
   * Constrained minimization using barrier method.
   * \ \ Vladimir \ \ Rutsky \ < altsysrq@gmail.com>
4
5
    * 29.03.2009
6
7
  #ifndef NUMERIC BARRIER METHOD HPP
  #define NUMERIC BARRIER METHOD HPP
10
11
  #include "numeric common.hpp"
12
13 #include < vector >
14
15 #include <boost / assert . hpp>
16 #include <boost/concept/assert.hpp>
17 #include <boost/concept_check.hpp>
18 #include <boost/bind.hpp>
19 #include <boost/lambda/lambda.hpp>
20 #include <boost / function.hpp>
21
22 #include "gradient descent.hpp"
23
24
  namespace numeric
25
26
  namespace barrier_method
27
28
     namespace
29
30
         ' TODO: Use boost::lambda instead.
       // f(x) + mu * Summ(-1 / g_i(x))
31
32
       template < class S >
33
       struct AdditionalFunction
34
       public:
35
36
         typedef S
                                                                               {\tt scalar\_type}\,;
37
         typedef vector < scalar_type>
                                                                               vector type;
38
39
       private:
40
         typedef boost::function<scalar type( vector type )>
                                                                               function type;
         typedef std::vector<function_type>
41
             limit_functions_vec_type;
42
       public:
43
44
         template < class Func, class LimitFuncIterator >
         AdditionalFunction (Func func,
45
                               {\bf LimitFuncIterator\ limitFuncBegin\ ,\ LimitFuncIterator}
46
                                   limitFuncEnd )
47
           : function
                              (func)
             limitFunctions\_(limitFuncBegin\ ,\ limitFuncEnd)
48
49
50
           // TODO: Assertions on input types.
51
52
53
         scalar_type operator()( scalar_type mu, vector_type const &x )
54
55
           scalar_type result (0.0);
56
           57
58
59
             scalar type const denominator = limitFunctions_[i](x);
60
61
62
             // TODO: Use normal constants.
             scalar_type const eps = 1e-8;
scalar_type const inf = 1e+8;
63
64
              if (abs(denominator) < eps)</pre>
```

```
66
                  // Division by zero.
 67
 68
                  result = inf;
 69
               }
 70
               else
 71
               {
 72
                  result += -mu / denominator;
 73
 74
 75
 76
             return result;
 77
           }
 78
 79
        private:
 80
           function\_type
                                        function ;
 81
           limit_functions_vec_type limitFunctions_;
 82
 83
         // TODO: Use boost::lambda instead.
 84
         // f(x) + mu * Summ(-1 / g_i(x))
 85
 86
        template < class S >
        struct AdditionalFunctionGradient
 87
 88
 89
        public:
 90
          typedef S
                                                                                      scalar_type;
 91
           typedef vector < scalar _ type >
                                                                                      vector_type;
 92
 93
           typedef boost::function<scalar_type( vector_type )>
typedef boost::function<vector_type( vector_type )>
 94
                                                                                      function_type;
 95
                                                                                      function_grad_type
 96
           typedef std::vector<function type>
               limit_functions_vec_type;
 97
           typedef std::vector<function_grad_type>
               limit_functions_grads_vec_type;
 98
 99
         public:
           template < class FuncGrad, class LimitFuncIterator, class LimitFuncGradIterator >
100
101
           AdditionalFunctionGradient (FuncGrad funcGrad,
102
                                           LimitFuncIterator
                                                                    limitFuncBegin,
                                                                     limitFuncEnd,
                                                LimitFuncIterator
                                            LimitFuncGradIterator\ limitFuncGradBegin
103
                                               LimitFuncGradIterator limitFuncGradEnd )
                                       (funcGrad)
104
             : functionGrad
105
             , limitFunctions
                                       (limitFuncBegin,
                                                               limitFuncEnd)
              , limitFunctions\overline{G}rads\_(limitFuncGradBegin, limitFuncGradEnd)
106
107
108
               / TODO: Assertions on input types.
109
             BOOST\_ASSERT(limitFunctions\_.size() == limitFunctionsGrads\_.size());
110
111
112
           vector_type operator()( scalar_type mu, vector_type const &x )
113
             vector type result = functionGrad_(x);
114
115
116
             \label{eq:formula} \textbf{for} \ (\, \text{size\_t} \ i \, = \, 0\,; \ i \, < \, limitFunctions\_\,.\,\, \text{size}\,(\,)\,; \, +\!\!\!+\!\!i\,)
117
                                         = limitFunctions
118
               scalar_type const fx
               vector type const fgradx = limitFunctionsGrads [i](x);
119
120
121
               for (size_t r = 0; r < x.size(); ++r)
122
123
                  // TODO: Use normal constants.
                  scalar\_type const eps = 1e-8;
124
                  scalar_type const inf = 1e+8;
125
126
                  if (abs(sqr(fx)) < eps)
127
128
                    // Division by zero.
129
                    result[r] = inf;
130
131
                  else
```

```
132
                                             result[r] += mu / sqr(fx) * fgradx[r];
133
134
135
                                 }
136
137
138
                             return result;
139
140
141
                    private:
142
                        function_grad_type
                                                                                                       functionGrad ;
                        limit\_functions\_vec\_type
143
                                                                                                       limitFunctions
144
                        limit\_functions\_grads\_vec\_type \ limitFunctionsGrads\_;
145
146
                     // TODO: Use boost::lambda instead.
147
                   {f template}<{f class} S >
148
149
                    struct ConstrainPredicate
150
151
                   public:
                        typedef S
152
                                                                                                                                                                                               scalar_type;
153
                        typedef vector < scalar type >
                                                                                                                                                                                               vector type;
154
155
                    private:
                        typedef boost::function<scalar_type( vector_type )>
156
                                                                                                                                                                                              function_type;
157
                        {\bf typedef} \ {\tt std}:: {\tt vector} {<\! \tt function\_type} {>}
                                  limit\_functions\_vec\_type;
158
159
                    public:
160
                        template< class LimitFuncIterator >
                        Constrain Predicate (\ Limit Func Iterator\ limit Func Begin\ ,\ Limit
161
                                  limitFuncEnd )
162
                              : limitFunctions_(limitFuncBegin, limitFuncEnd)
163
                             /\!/ TODO: Assertions on input types.
164
165
                        }
166
                        bool operator()( vector type const &x )
167
168
169
                             for (size_t i = 0; i < limitFunctions_size(); ++i)
                                  if'(limitFunctions_[i](x) > 0)
170
171
                                       return false;
172
173
                             return true;
174
                        }
175
176
                   private:
177
                        limit_functions_vec_type limitFunctions_;
178
                  // End of anonymous namespace
179
180
181
             \mathbf{template} \! < \!\!\! \mathbf{class} \!\!\! \mathbf{S} >
182
              struct PointDebugInfo
183
184
                   typedef S
                                                                                         scalar_type;
185
                   typedef vector < scalar _type > vector _type;
186
187
                    PointDebugInfo()
188
189
190
                    PointDebugInfo( vector_type const &newx, scalar_type newmu, scalar_type newfx,
                            scalar_type newBx )
191
                        : x (newx)
192
                        , mu(newmu)
193
                           fx (newfx)
194
                        , Bx(newBx)
195
196
197
                    vector_type x;
198
                   scalar type mu;
199
                   scalar_type fx;
```

```
200
       scalar type Bx;
201
202
203
      // TODO: Habdle more end cases, not all problems input have solutions.
204
      template < class Func, class FuncGrad,
                class S,
205
206
                {\bf class} \ {\bf LimitFuncIterator} \ , \ {\bf class} \ {\bf LimitFuncGradIterator} \ ,
207
                class PointsOut >
208
     inline
209
      vector\!<\!\!S\!\!>
210
       find min (Func function, FuncGrad functionGrad,
211
                  LimitFuncIterator
                                        gBegin ,
                                                     LimitFuncIterator
                                                                             gEnd,
212
                  LimitFuncGradIterator \ gGradBegin \, , \ LimitFuncGradIterator \ gGradEnd \, , \\
                  vector <S > const &startPoint,
213
214
                  S startMu, S beta,
215
                  S epsilon,
216
                  S gradientDescentPrecision, S gradientDescentStep,
217
                  PointsOut pointsOut )
218
        typedef S
219
                                                    scalar_type;
                                   <scalar_type> vector_type;
220
        typedef ublas::vector
221
        typedef ublas::scalar traits<scalar type> scalar traits type;
222
223
        // TODO: Check for iterators concept assert.
224
225
        typedef typename LimitFuncIterator::value_type
                                                            limit_func_type;
226
       typedef typename LimitFuncGradIterator::value type limit func grad type;
227
228
       BOOST_CONCEPT_ASSERT((boost::UnaryFunction<Func,
                                                                            scalar_type,
            vector_type>));
       {\tt BOOST\_CONCEPT\_ASSERT((boost::UnaryFunction{<}FuncGrad},
229
                                                                            vector_type,
            vector type>));
       BOOST CONCEPT ASSERT((boost::UnaryFunction<limit func type,
230
                                                                            scalar type,
            vector_type>));
       BOOST CONCEPT ASSERT((boost::UnaryFunctionfunc grad type, vector type,
231
            vector_type>));
232
233
       BOOST ASSERT(epsilon > 0);
234
235
        std::vector<limit_func_type>
                                                (gBegin,
                                          g
236
       std::vector<limit_func_grad_type> gGrad(gGradBegin, gGradEnd);
237
       BOOST ASSERT(g.size() == gGrad.size());
238
       BOOST\_ASSERT(beta > 0 \&\& beta < 1);
239
240
241
        // Building additional function and it's gradient.
242
        typedef boost::function<scalar_type( vector_type )> function_type;
243
        typedef boost::function<vector_type( vector_type )> function_gradient_type;
        typedef AdditionalFunction
                                          <scalar_type>
244
                                                              additional_function_type;
        typedef AdditionalFunctionGradient<scalar_type>
245
                                                              additional_function_gradient_type
        typedef ConstrainPredicate<scalar type>
                                                              constrain\_predicate\_type\,;
246
247
        typedef PointDebugInfo<scalar_type>
                                                              points_debug_info_type;
248
        249
                                                                              gBegin, gEnd);
250
           gGradBegin, gGradEnd);
251
        constrain_predicate_type
                                           constrainPred
                                                              (gBegin, gEnd);
252
253
        // Initializing
254
        vector_type x = startPoint;
        scalar\_type\ mu = startMu;
255
256
257
       BOOST_ASSERT(constrainPred(x)); // TODO: Rename 'constrain' by 'constraint'.
258
        points\_debug\_info\_type\ pdi(x,\ mu,\ function(x),\ additionalFunc(mu,\ x)\ -\ function(x));
259
260
        *pointsOut++=pdi;
261
262
        size t iterations = 0;
263
        while (true)
264
```

```
265
                             function\_type
                                                                                                  currFunc
                                                                                                                              = boost::bind<scalar type>(additionalFunc,
                                       mu, _1);
                             function\_gradient\_type \ currFuncGrad = boost::bind < vector\_type > (additionalFuncGrad \ , bind < vector\_typ
266
                                       mu, _1);
267
268
                            // Solving additional unconstrained problem.
269
                             vector_type const &newx =
270
                                        \overline{gradient}_{descent}:: find_{min}
                                              <\!\!function\_type\;,\;\;function\_gradient\_type\;,\;\;vector\_type\!>
271
272
                                                    (currFunc, currFuncGrad,
273
                                                       {\tt gradientDescentPrecision}\;,\;\;{\tt gradientDescentStep}\;,
274
275
                                                       constrainPred , DummyOutputIterator());
276
                            points_debug_info_type pdi(newx, mu, function(newx), currFunc(newx) - function(newx
277
                                       ));
                             *pointsOut++ = pdi;
278
279
280
                             // mu \ k * B(x \ k+1) < epsilon
281
                             if (currFunc(newx) - function(newx) < epsilon)
282
283
                                   // Required precision reached.
284
                                   return newx;
285
                             }
286
                             else
287
                                  // Moving to next point.
288
289
                                  x = newx;
290
                                 mu *= beta;
291
292
293
                            ++iterations;
294
                              // debug
295
296
                             if (iterations >= 1000)
297
                                   std :: cerr << "barrier_method :: find_min() : \Gamma Too \Gamma many \Gamma iterations ! \ n";
298
299
                                  break;
300
301
                             // end of debug
302
303
304
                      return x;
305
306
               // End of namespace 'barrier_method'.
307
          } // End of namespace 'numeric'.
308
         #endif // NUMERIC_BARRIER_METHOD_HPP
```

## Результаты решения

Таблица 1: Результаты работы барьерного метода

|                                     | _                         |                           |                           |                           |                           |                           |                           |                           |                           |
|-------------------------------------|---------------------------|---------------------------|---------------------------|---------------------------|---------------------------|---------------------------|---------------------------|---------------------------|---------------------------|
| $g_2(x)$                            | -0.0531038                | -0.0531038                | -0.0531038                | -0.0531038                | -0.0531038                | -0.0531038                | -0.0531038                | -0.0531038                | -0.0531038                |
| $g_1(x)$                            | -0.079139                 | -0.079139                 | -0.079139                 | -0.079139                 | -0.079139                 | -0.079139                 | -0.079139                 | -0.079139                 | -0.079139                 |
| $\nabla f(x)$                       | (-8.68471, -6.736783e+00) |
| $f(x) \qquad  f_i(x) - f_{i-1}(x) $ |                           | 0.0000000+00              | 0.0000000+00              | 0.0000000+00              | 0.0000000+00              | 0.0000000+00              | 0.0000000+00              | 0.00000000+00             | 0.00000000+00             |
| f(x)                                | -10.79788183              | -10.79788183              | -10.79788183              | -10.79788183              | -10.79788183              | -10.79788183              | -10.79788183              | -10.79788183              | -10.79788183              |
| x                                   | (0.657644, 0.63160858)    | (0.657644, 0.63160858)    | (0.657644, 0.63160858)    | (0.657644, 0.63160858)    | (0.657644, 0.63160858)    | (0.657644, 0.63160858)    | (0.657644, 0.63160858)    | (0.657644, 0.63160858)    | (0.657644, 0.63160858)    |
| Шаги                                | 11                        | 12                        | 13                        | 14                        | 15                        | 16                        | 17                        | 18                        | 19                        |
| Точность   Шаги                     | 1e-01                     | 1e-02                     | 1e-03                     | 1e-04                     | 1e-05                     | 1e-06                     | 1e-07                     | 1e-08                     | 1e-09                     |