

# Решение задачи многомерной минимизации функции с ограничениями методом барьеров

Владимир Руцкий, 3057/2

# Код программы

Исходный код 1: Барьерный метод

```
1  /*
2  *  barrier_method.hpp
3  *  Constrained minimization using barrier method.
4  *  Vladimir Rutsky <altsysrq@gmail.com>
5  *  29.03.2009
6  */
7
8  #ifndef NUMERIC_BARRIER_METHOD_HPP
9  #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.
31             //  $f(x) + \mu * \text{Summ}(-1 / g_i(x))$ 
32             template< class S >
33             struct AdditionalFunction
34             {
35             public:
36                 typedef S scalar_type;
37                 typedef vector<scalar_type> vector_type;
38
39             private:
40                 typedef boost::function<scalar_type( vector_type )> function_type;
41                 typedef std::vector<function_type> limit_functions_vec_type;
42
43             public:
44                 template< class Func, class LimitFuncIterator >
45                 AdditionalFunction( Func func,
46                                     LimitFuncIterator limitFuncBegin, LimitFuncIterator
47                                     limitFuncEnd )
48                     : function_( func )
49                     , limitFunctions_( limitFuncBegin, limitFuncEnd )
50                 {
51                     // TODO: Assertions on input types.
52                 }
53
54                 scalar_type operator()( scalar_type mu, vector_type const &x )
55                 {
56                     scalar_type result(0.0);
57
58                     result += function_(x);
59                     //std::cout << " F (" << x << ") = " << result << " + "; // debug
60                     for (size_t i = 0; i < limitFunctions_.size(); ++i)
61                     {
62                         scalar_type const denominator = limitFunctions_[i](x);
63
64                         // TODO: Use normal constants.
65                         scalar_type const eps = 1e-15; // FIXME
66                         scalar_type const inf = 1e+8;
```

```

66         if (abs(denominator) < eps)
67         {
68             // Division by zero.
69             // TODO: Break loop and leave value infinite.
70             result = inf;
71         }
72         else
73         {
74             result += -mu / denominator;
75         }
76         //std::cout << -mu / denominator << " + ";// debug
77     }
78     //std::cout << " == " << result << std::endl;// debug
79
80     return result;
81 }
82
83 private:
84     function_type          function_;
85     limit_functions_vec_type limitFunctions_;
86 };
87
88 // TODO: Use boost::lambda instead.
89 // f(x) + mu * Summ(-1 / g_i(x))
90 template< class S >
91 struct AdditionalFunctionGradient
92 {
93 public:
94     typedef S scalar_type;
95     typedef vector<scalar_type> vector_type;
96
97 private:
98     typedef scalar_vector<scalar_type> scalar_vector_type;
99
100     typedef boost::function<scalar_type( vector_type )> function_type;
101     typedef boost::function<vector_type( vector_type )> function_grad_type;
102
103     typedef std::vector<function_type> limit_functions_vec_type;
104     typedef std::vector<function_grad_type> limit_functions_grads_vec_type;
105
106 public:
107     template< class FuncGrad, class LimitFuncIterator, class LimitFuncGradIterator >
108     AdditionalFunctionGradient( FuncGrad funcGrad,
109                                LimitFuncIterator limitFuncBegin,
110                                LimitFuncIterator limitFuncEnd,
111                                LimitFuncGradIterator limitFuncGradBegin,
112                                LimitFuncGradIterator limitFuncGradEnd )
113     : functionGrad_ (funcGrad)
114     , limitFunctions_ (limitFuncBegin, limitFuncEnd)
115     , limitFunctionsGrads_ (limitFuncGradBegin, limitFuncGradEnd)
116     {
117         // TODO: Assertions on input types.
118         BOOST_ASSERT(limitFunctions_.size() == limitFunctionsGrads_.size());
119     }
120
121     vector_type operator()( scalar_type mu, vector_type const &x )
122     {
123         vector_type result = functionGrad_(x);
124
125         for (size_t i = 0; i < limitFunctions_.size(); ++i)
126         {
127             scalar_type const gx = limitFunctions_[i](x);
128             vector_type const gGradx = limitFunctionsGrads_[i](x);
129
130             // TODO: Use normal constants.
131             scalar_type const eps = 1e-30; // FIXME!
132             scalar_type const inf = 1e+8;
133             if (abs(sqr(gx)) < eps)
134             {

```

```

131         // Division by zero.
132         // TODO: Break loop and leave value infinite.
133         return scalar_vector_type(x.size(), inf);
134     }
135     else
136     {
137         result = result + (mu / sqrt(gx)) * gGradx;
138     }
139 }
140
141 return result;
142 }
143
144 private:
145     function_grad_type          functionGrad_;
146     limit_functions_vec_type     limitFunctions_;
147     limit_functions_grads_vec_type limitFunctionsGrads_;
148 };
149
150 // TODO: Use boost::lambda instead.
151 template< class S >
152 struct ConstrainPredicate
153 {
154 public:
155     typedef S                                     scalar_type;
156     typedef vector<scalar_type>                   vector_type;
157
158 private:
159     typedef boost::function<scalar_type( vector_type )> function_type;
160     typedef std::vector<function_type>             limit_functions_vec_type;
161
162 public:
163     template< class LimitFuncIterator >
164     ConstrainPredicate( LimitFuncIterator limitFuncBegin, LimitFuncIterator
165         limitFuncEnd )
166         : limitFunctions_(limitFuncBegin, limitFuncEnd)
167     {
168         // TODO: Assertions on input types.
169     }
170
171     bool operator()( vector_type const &x )
172     {
173         for (size_t i = 0; i < limitFunctions_.size(); ++i)
174             if (limitFunctions_[i](x) > 0)
175                 return false;
176
177         return true;
178     }
179
180 private:
181     limit_functions_vec_type limitFunctions_;
182 };
183 // End of anonymous namespace
184
185 template< class S >
186 struct PointDebugInfo
187 {
188     typedef S                                     scalar_type;
189     typedef vector<scalar_type>                   vector_type;
190
191     PointDebugInfo()
192     {}
193
194     PointDebugInfo( vector_type const &newx, scalar_type newmu, scalar_type newfx,
195         scalar_type newBx )
196         : x(newx)
197         , mu(newmu)
198         , fx(newfx)
199         , Bx(newBx)
200     {}

```

```

199     vector_type x;
200     scalar_type mu;
201     scalar_type fx;
202     scalar_type Bx;
203 };
204
205
206 // TODO: Haddle more end cases, not all problems input have solutions.
207 template< class Func, class FuncGrad,
208           class S,
209           class LimitFuncIterator, class LimitFuncGradIterator,
210           class PointsOut >
211 inline
212 vector<S>
213     find_min( Func function, FuncGrad functionGrad,
214               LimitFuncIterator gBegin, LimitFuncIterator gEnd,
215               LimitFuncGradIterator gGradBegin, LimitFuncGradIterator gGradEnd,
216               vector<S> const &startPoint,
217               S startMu, S beta,
218               S epsilon,
219               S gradientDescentPrecision, S gradientDescentStep,
220               PointsOut pointsOut )
221 {
222     typedef S scalar_type;
223     typedef ublas::vector<scalar_type> vector_type;
224     typedef ublas::scalar_traits<scalar_type> scalar_traits_type;
225
226     // TODO: Check for iterators concept assert.
227     // Note: Input should be accurate so, that start point must be admissible not only
228     // for input function but for
229     // additional function too.
230
231     typedef typename LimitFuncIterator::value_type limit_func_type;
232     typedef typename LimitFuncGradIterator::value_type limit_func_grad_type;
233
234     BOOST_CONCEPT_ASSERT(( boost::UnaryFunction<Func, scalar_type,
235     vector_type> ));
236     BOOST_CONCEPT_ASSERT(( boost::UnaryFunction<FuncGrad, vector_type,
237     vector_type> ));
238     BOOST_CONCEPT_ASSERT(( boost::UnaryFunction<limit_func_type, scalar_type,
239     vector_type> ));
240     BOOST_CONCEPT_ASSERT(( boost::UnaryFunction<limit_func_grad_type, vector_type,
241     vector_type> ));
242
243     BOOST_ASSERT(epsilon > 0);
244
245     std::vector<limit_func_type> g(gBegin, gEnd);
246     std::vector<limit_func_grad_type> gGrad(gGradBegin, gGradEnd);
247
248     BOOST_ASSERT(g.size() == gGrad.size());
249     BOOST_ASSERT(beta > 0 && beta < 1);
250
251     // Building additional function and it's gradient.
252     typedef boost::function<scalar_type( vector_type )> function_type;
253     typedef boost::function<vector_type( vector_type )> function_gradient_type;
254     typedef AdditionalFunction<scalar_type> additional_function_type;
255     typedef AdditionalFunctionGradient<scalar_type> additional_function_gradient_type;
256
257     ;
258     typedef ConstrainPredicate<scalar_type> constrain_predicate_type;
259     typedef PointDebugInfo<scalar_type> points_debug_info_type;
260
261     additional_function_type additionalFunc(function, gBegin, gEnd);
262     additional_function_gradient_type additionalFuncGrad(functionGrad, gBegin, gEnd,
263     gGradBegin, gGradEnd);
264     constrain_predicate_type constrainPred(gBegin, gEnd);
265
266     // Initializing
267     vector_type x = startPoint;
268     scalar_type mu = startMu;
269
270     BOOST_ASSERT(constrainPred(x)); // TODO: Rename 'constrain' by 'constraint'.

```

```

263 mu /= beta;
264 points_debug_info_type pdi(x, mu, function(x), (additionalFunc(mu, x) - function(x))
265 / mu);
266 mu *= beta;
267 *pointsOut++ = pdi;
268
269 size_t iterations = 0;
270 while (true)
271 {
272     // Additional function:  $f(x) + \mu * \text{Summ}(-1 / g_i(x))$ 
273
274     function_type currFunc = boost::bind<scalar_type>(additionalFunc,
275 mu, _1);
276     function_gradient_type currFuncGrad = boost::bind<vector_type>(additionalFuncGrad,
277 mu, _1);
278
279     // Solving additional unconstrained problem.
280     vector_type newx;
281     gradient_descent::gradient_descent_result const result =
282         gradient_descent::find_min
283         <function_type, function_gradient_type, vector_type>
284         (currFunc, currFuncGrad,
285 x,
286         gradientDescentPrecision, gradientDescentStep,
287 newx,
288         constrainPred, DummyOutputIterator());
289     BOOST_ASSERT(result == result); // TODO: Handle result states.
290
291     // debug
292     std::cout << iterations << ":_ " << newx << std::endl;
293     // end of debug
294
295     scalar_type const muBx = currFunc(newx) - function(newx);
296     scalar_type const Bx = muBx / mu;
297     points_debug_info_type pdi(newx, mu, function(newx), Bx);
298     *pointsOut++ = pdi;
299
300     //  $\mu_k * B(x_{k+1}) < \epsilon$ 
301     BOOST_ASSERT(muBx >= 0);
302     if (muBx < epsilon)
303     {
304         // Required precision reached.
305         return newx;
306     }
307     else
308     {
309         // Moving to next point.
310         x = newx;
311         mu *= beta;
312     }
313
314     ++iterations;
315
316     // debug
317     if (iterations >= 100)
318     {
319         std::cerr << "barrier_method::find_min():_Too_many_iterations!\n";
320         break;
321     }
322     // end of debug
323 }
324
325 return x;
326 } // End of namespace 'barrier_method'.
327 } // End of namespace 'numeric'.
328 #endif // NUMERIC_BARRIER_METHOD_HPP

```

## Результаты решения

Таблица 1: Детальная работа алгоритма при точности  $10^{-3}$

k	$x_k$	$f(x_k)$	$\mu_k$	$B(x_k)$	$\mu_k B(x_k)$	$\theta(x_k)$
1	( -20.00000000, -20.00000000 )	1160.00000000	10000000.00000000	0.03225806	322580.64516129	323740.64516129
2	( -52.71337242, -53.52108827 )	6598.50895222	1000000.00000000	0.01239536	12395.36144230	18993.87039452
3	( -23.26405214, -24.05945195 )	1545.18948678	100000.00000000	0.02740361	2740.36085107	4285.55033784
4	( -9.65735618, -10.42556406 )	381.93498853	10000.00000000	0.06226691	622.66912161	1004.60411014
5	( -3.46094033, -4.16915096 )	97.32253865	1000.00000000	0.14885592	148.85592121	246.17845986
6	( -0.77877483, -1.36351004 )	21.16147848	100.00000000	0.38483508	38.48350805	59.64498653
7	( 0.23912614, -0.14916639 )	-1.11849837	10.00000000	1.08409873	10.84098728	9.72248891
8	( 0.54561393, 0.33171155 )	-7.70210458	1.00000000	2.99720197	2.99720197	-4.70490262
9	( 0.54561393, 0.33171155 )	-7.70210458	0.10000000	2.99720197	0.29972020	-7.40238439
10	( 0.66177407, 0.57473345 )	-10.44734479	0.01000000	15.12882066	0.15128821	-10.29605659
11	( 0.66389643, 0.65542767 )	-11.01204170	0.00100000	99.20331113	0.09920331	-10.91283838
12	( 0.66579957, 0.66309668 )	-11.07978282	0.00010000	313.41967897	0.03134197	-11.04844085
13	( 0.66639338, 0.66553611 )	-11.10120421	0.00001000	990.82504340	0.00990825	-11.09129596
14	( 0.66658034, 0.66630897 )	-11.10797821	0.00000100	3132.90451393	0.00313290	-11.10484530
15	( 0.66663938, 0.66655353 )	-11.11012034	0.00000010	9906.20374429	0.00099062	-11.10912972

Таблица 2: Результаты работы барьерного метода

Точность	Шаги	$x$	$f(x)$	$f_i(x) - f_{i-1}(x)$	$\nabla f(x)$	$g_1(x)$	$g_2(x)$
1e-01	11	( 0.66389643, 0.65542767 )	-11.01204170		(-8.672207e+00, -6.689145e+00)	-0.0252482	-0.0167795
1e-02	13	( 0.66639338, 0.66553611 )	-11.10120421	-8.916251e-02	(-8.667213e+00, -6.668928e+00)	-0.0025344	-0.00167714
1e-03	15	( 0.66663938, 0.66655353 )	-11.11012034	-8.916132e-03	(-8.666721e+00, -6.666893e+00)	-0.000253568	-0.000167715
1e-04	17	( 0.66666395, 0.66665534 )	-11.11101202	-8.916856e-04	(-8.666672e+00, -6.666689e+00)	-2.53763e-05	-1.67663e-05
1e-05	19	( 0.66666640, 0.66666553 )	-11.11110120	-8.917952e-05	(-8.666667e+00, -6.666669e+00)	-2.54186e-06	-1.67465e-06
1e-06	21	( 0.66666666, 0.66666651 )	-11.11111000	-8.801393e-06	(-8.666667e+00, -6.666667e+00)	-3.15646e-07	-1.73226e-07