Listing 1: Convex cutting plane

```
1  /*
2   * kelley_cutting_plane.hpp
3   * Kelley's convex cutting plane algorithm.
4   * Vladimir Rutsky <altsysrq@gmail.com>
5   * 27.04.2009
6   */
7
8  #ifndef NUMERIC_KELLEY_CUTTING_PLANE_HPP
9  #define NUMERIC_KELLEY_CUTTING_PLANE_HPP
10
11 #include "numeric_common.hpp"
12
13 #include "linear_problem.hpp"
14 #include "linear_problem_algs.hpp"
15 #include "simplex_alg.hpp"
16
17 #include <vector>
18
19 #include <boost/assert.hpp>
20 #include <boost/concept/assert.hpp>
21 #include <boost/concept_check.hpp>
22
23 namespace numeric
24 {
25 namespace kelley_cutting_plane
26 {
27   // For details see
28   //   David G. Luenberger, Yinyu Ye
29   //   Linear and Nonlinear Programming, Third Edition
30   //   section 14.8 Kelley's convex cutting plane algorithm (p. 463).
31
32   // Finds linear function minimum prefer to convex differentiable constraints.
33   //   min c^T * x,  g(x) <= 0
34   // g(x) and grad g(x) are defined by coordinates through iterators.
35   // Initial constraints and problem formalization is stored in common linear problem
         structure,
36   // which is expanded by new constraints along algorithm run.
37   // TODO: Handle more cases, return value should be enumeration of different exit
         statuses.
38   template< class S, class CLPTraits, class FuncIterator, class GradFuncIterator >
39   inline
40   vector<S>
41     find_min( FuncIterator     funcBegin,     FuncIterator         funcEnd,
42               GradFuncIterator gradFuncBegin, GradFuncIterator     gradFuncEnd,
43               linear_problem::common_linear_problem<S, CLPTraits> &commonLP )
44   {
45     typedef CLPTraits                                    clp_traits;
46     typedef S                                            scalar_type;
47     typedef vector<scalar_type>                          vector_type;
48     typedef matrix<scalar_type>                          matrix_type;
49     typedef zero_matrix<scalar_type>                     zero_matrix;
50     typedef scalar_traits<scalar_type>                   scalar_traits_type;
51
52     typedef typename FuncIterator::value_type            function_type;
53     typedef typename GradFuncIterator::value_type        gradient_function_type;
54
55     typedef linear_problem::common_linear_problem            <scalar_type>
           common_linear_problem_type;
56     typedef linear_problem::canonical_linear_problem         <scalar_type>
           canonical_linear_problem_type;
57     typedef typename linear_problem::converter_template_type<scalar_type>::type
           converter_type;
58
59     // TODO: Using same type in much places now (like scalar_type).
60     BOOST_CONCEPT_ASSERT((boost::UnaryFunction<function_type,          scalar_type,
           vector_type>));
61     BOOST_CONCEPT_ASSERT((boost::UnaryFunction<gradient_function_type, vector_type,
           vector_type>));
62
63     // TODO: Assert that input constraints are valid (they rise correct LP).
```

```
64      BOOST_ASSERT(linear_problem::is_valid(commonLP));
65
66      size_t const n = linear_problem::variables_count(commonLP);
67
68      BOOST_ASSERT(n > 0);
69
70      // Storing constrain function and its gradient.
71      std::vector<function_type>          g     (funcBegin,     funcEnd);
72      std::vector<gradient_function_type> gGrad(gradFuncBegin, gradFuncEnd);
73
74      size_t nIterations(0);
75      size_t const nMaxIterations(1000); // debug
76      while (nIterations < nMaxIterations)
77      {
78        // Solving linear problem.
79        vector_type commonResult;
80        simplex::simplex_result_type const result = solve_by_simplex(commonLP, commonResult
              );
81        BOOST_ASSERT(result == simplex::srt_min_found); // FIXME: Handle other cases.
82        BOOST_ASSERT(linear_problem::check_linear_problem_solving_correctness(commonLP));
83
84        // Adding new limits to common linear problem according to elements that satisfies
              g_i(x) > 0.
85        bool isInside(true);
86        for (size_t r = 0; r < n; ++r)
87        {
88          scalar_type const gr = g[r](commonResult);
89          if (gr > 0)
90          {
91            isInside = false;
92
93            // Adding new constraint:
94            // g[r](commonResult) + grad g[r](commonResult) * (x - commonResult) <= 0.
95            // or
96            // grad g[r](commonResult) * x <= grad g[r](commonResult) * commonResult - g[r
                  ](commonResult).
97
98            size_t const newRows = commonLP.b().size() + 1;
99            BOOST_ASSERT(commonLP.ASign().size() == newRows - 1);
100           BOOST_ASSERT(commonLP.A().size1()    == newRows - 1);
101           BOOST_ASSERT(commonLP.A().size2()    == n);
102
103           commonLP.b().resize(newRows, true);
104           commonLP.A().resize(newRows, n, true);
105           commonLP.ASign().resize(newRows, true);
106
107           commonLP.ASign()(newRows - 1) = linear_problem::inequality_leq;
108
109           vector_type const grGrad = gGrad[r](commonResult);
110           BOOST_ASSERT(!eq_zero(norm_2(grGrad))); // FIXME: I think this is possible case
                  .
111           row(commonLP.A(), newRows - 1) = grGrad;
112
113           commonLP.b()(newRows - 1) = inner_prod(grGrad, commonResult) - gr;
114
115           // Assert that builded constraint cuts previosly founded minimum point.
116           BOOST_ASSERT(inner_prod(row(commonLP.A(), newRows - 1), commonResult) >
                  commonLP.b()(newRows - 1));
117
118           BOOST_ASSERT(linear_problem::assert_valid(commonLP));
119         }
120       }
121
122       if (isInside)
123       {
124         // Founded minimum of linear problem lies inside convex limits, so this is the
                answer.
125         return commonResult;
126       }
127     }
128
```

```
129       BOOST_ASSERT(0); // FIXME: Handle different cases.
130       return vector_type(0);
131     }
132   } // End of namespace 'kelley_cutting_plane'.
133 } // End of namespace 'numeric'.
134
135 #endif // NUMERIC_KELLEY_CUTTING_PLANE_HPP
```