Nonlinear Programming Using CppAD and Ipopt: Example and Test

Purpose

This example program demonstrates how to use ipopt_solve to solve the example problem in the Ipopt documentation; i.e., the problem

Minimize

$x_1 * x_4 * (x_1 + x_2 + x_3) + x_3$

Subject to

$x_1 * x_2 * x_3 * x_4 \geq 25$

$x_{21} + x_{22} + x_{23} + x_{24} = 40$

$1 \leq x_1, x_2, x_3, x_4 \leq 5$

Configuration Requirement

This example will be compiled and tested provided that ipopt_prefix is specified on the cmake command line.

```
# include <cppad/ipopt/solve.hpp>

namespace {
     using CppAD::AD;

     class FG_eval {
     public:
          typedef CPPAD_TESTVECTOR( AD<double> ) ADvector;
          void operator()(ADvector& fg, const ADvector& x)
          {     assert( fg.size() == 3 );
               assert( x.size()  == 4 );

               // Fortran style indexing
               AD<double> x1 = x[0];
               AD<double> x2 = x[1];
               AD<double> x3 = x[2];
               AD<double> x4 = x[3];
               // f(x)
               fg[0] = x1 * x4 * (x1 + x2 + x3) + x3;
               // g_1 (x)
               fg[1] = x1 * x2 * x3 * x4;
               // g_2 (x)
               fg[2] = x1 * x1 + x2 * x2 + x3 * x3 + x4 * x4;
               //
               return;
          }
     };
}

bool get_started(void)
{     bool ok = true;
     size_t i;
     typedef CPPAD_TESTVECTOR( double ) Dvector;

     // number of independent variables (domain dimension for f
and g)
     size_t nx = 4;
     // number of constraints (range dimension for g)
     size_t ng = 2;
     // initial value of the independent variables
     Dvector xi(nx);
     xi[0] = 1.0;
     xi[1] = 5.0;
     xi[2] = 5.0;
     xi[3] = 1.0;
     // lower and upper limits for x
     Dvector xl(nx), xu(nx);
     for(i = 0; i < nx; i++)
     {     xl[i] = 1.0;
          xu[i] = 5.0;
     }
     // lower and upper limits for g
     Dvector gl(ng), gu(ng);
     gl[0] = 25.0;     gu[0] = 1.0e19;
     gl[1] = 40.0;     gu[1] = 40.0;

     // object that computes objective and constraints
     FG_eval fg_eval;

     // options
     std::string options;
     // turn off any printing
     options += "Integer print_level  0\n";
     options += "String  sb           yes\n";
     // maximum number of iterations
     options += "Integer max_iter     10\n";
     // approximate accuracy in first order necessary conditions;
     // see Mathematical Programming, Volume 106, Number 1,
     // Pages 25-57, Equation (6)
     options += "Numeric tol          1e-6\n";
     // derivative testing
     options += "String  derivative_test            second-
order\n";
     // maximum amount of random pertubation; e.g.,
     // when evaluation finite diff
     options += "Numeric point_perturbation_radius  0.\n";

     // place to return solution
     CppAD::ipopt::solve_result<Dvector> solution;

     // solve the problem
     CppAD::ipopt::solve<Dvector, FG_eval>(
          options, xi, xl, xu, gl, gu, fg_eval, solution
     );
     //
     // Check some of the solution values
     //
     ok &= solution.status ==
CppAD::ipopt::solve_result<Dvector>::success;
     //
     double check_x[]  = { 1.000000,4.743000,3.82115,1.379408 };
     double check_zl[] = { 1.087871, 0.,     0.,     0.       };
     double check_zu[] = { 0.,       0.,     0.,     0.       };
     double rel_tol    = 1e-6;  // relative tolerance
     double abs_tol    = 1e-6;  // absolute tolerance
     for(i = 0; i < nx; i++)
     {     ok &= CppAD::NearEqual(
               check_x[i],  solution.x[i],   rel_tol, abs_tol
          );
          ok &= CppAD::NearEqual(
               check_zl[i], solution.zl[i], rel_tol, abs_tol
          );
          ok &= CppAD::NearEqual(
               check_zu[i], solution.zu[i], rel_tol, abs_tol
          );
     }

     return ok;
}
```