```
//  file: multimin_test.cpp
//
//  C++ Program to test multidimensional minimization routines from
//   the gsl numerical library.
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//      04/24/05  original version, based on gsl manual example
//      02/15/05  minor upgrades to documentation
//      02/12/06  minor improvements (inline sqr, cmath, declarations)
//
//  Notes:
//   * Example taken from the GNU Scientific Library Reference Manual
//      Edition 1.4, for GSL Version 1.4, August 2003
//   * Compile and link with:
//       g++ -Wall -o multimin_test multimin_test.cpp -lgsl -lgslcblas
//   * gsl routines have built-in
//       extern "C" {
//          <header stuff>
//       }
//      so they can be called from C++ programs without modification
//
//*****************************************************************//

// The following details are taken from the GSL documentation
//
//
// This example program finds the minimum of a paraboloid function. The
// location of the minimum is offset from the origin in x and y, and
// the function value at the minimum is non-zero.
//
// The initial step-size is chosen as 0.01, a conservative estimate in
// this case, and the line minimization parameter is set at 0.0001. The
// program terminates when the norm of the gradient has been reduced
// below 0.001. The output of the program is shown below,
//
//          x       y         f
//     1 4.99629 6.99072  687.84780
//     2 4.98886 6.97215  683.55456
//     3 4.97400 6.93501  675.01278
//     4 4.94429 6.86073  658.10798
//     5 4.88487 6.71217  625.01340
//     6 4.76602 6.41506  561.68440
//     7 4.52833 5.82083  446.46694
//     8 4.05295 4.63238  261.79422
//     9 3.10219 2.25548   75.49762
//    10 2.85185 1.62963   67.03704
//    11 2.19088 1.76182   45.31640
//    12 0.86892 2.02622   30.18555
// Minimum found at:
//    13 1.00000 2.00000   30.00000
//
// Note that the algorithm gradually increases the step size as it
// successfully moves downhill, as can be seen by plotting the successive
// points.
//
// The conjugate gradient algorithm finds the minimum on its second
// direction because the function is purely quadratic. Additional
// iterations would be needed for a more complicated function.
//
//*****************************************************************

// include files
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cmath>
using namespace std;
```



```
#include <gsl/gsl_errno.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_vector.h>     // gsl vector stuff
#include <gsl/gsl_multimin.h>   // gsl multidimensional minimization

// function prototypes
inline double sqr (double x) {return x*x;};     // square a double

double my_f (const gsl_vector *xvec_ptr, void *params);
void my_df (const gsl_vector *xvec_ptr, void *params, gsl_vector *df_ptr);
void my_fdf (const gsl_vector *xvec_ptr, void *params_ptr,
             double *f_ptr, gsl_vector *df_ptr);


//*****************************************************************
int
main ()
{
  // Exact position of the two-dimensional minimum (1,2).
  double minima[2] = { 1.0, 2.0 };

  // define and set up the gsl multimin function
  gsl_multimin_function_fdf my_func;
  my_func.f = &my_f;          // the function to be minimized
  my_func.df = &my_df;        // the gradient of the function
  my_func.fdf = &my_fdf;      // combined (see the function below)
  my_func.n = 2;              // size of x vectors
  my_func.params = &minima;  // parameters available to the function

  // Allocate x vector and set starting point, e.g., x = (5,7)
  gsl_vector *xvec_ptr = gsl_vector_alloc (2);
  gsl_vector_set (xvec_ptr, 0, 5.0);
  gsl_vector_set (xvec_ptr, 1, 7.0);

  // allocate and set the minimizer and its type (see gsl manual)
  const gsl_multimin_fdfminimizer_type *type_ptr
          = gsl_multimin_fdfminimizer_conjugate_fr;
  gsl_multimin_fdfminimizer *minimizer_ptr
          = gsl_multimin_fdfminimizer_alloc (type_ptr, 2);

  // set the tolerance and starting step size
  double step_size = 0.01;
  double tolerance = 1.e-4;
  gsl_multimin_fdfminimizer_set (minimizer_ptr, &my_func, xvec_ptr,
                                 step_size, tolerance);

  size_t iteration = 0;          // initialize iteration counter
  size_t max_iterations = 100;  // stop at this iteration if not converged
  int status = 0;
  cout << "iter    x      y      value " << endl;
  do
    {
      iteration++;
      status = gsl_multimin_fdfminimizer_iterate (minimizer_ptr);

      if (status)
        {
          break;  // this should only happen if an error code is returned
        }

      // check for convergence (state is either GSL_CONTINUE or GSL_SUCCESS)
      status = gsl_multimin_test_gradient (minimizer_ptr->gradient, tolerance);
      if (status == GSL_SUCCESS)   // if we're done, print out the details
        {
          cout << "Minimum found at: " << endl;
        }

      cout << setw(3) << iteration << " "
```

```
            << fixed << setprecision(5)
            << setw(10) << gsl_vector_get (minimizer_ptr->x, 0) << " "
            << setw(10) << gsl_vector_get (minimizer_ptr->x, 1) << " "
            << setw(12) << minimizer_ptr->f
            << endl;
    }
  while (status == GSL_CONTINUE && iteration < max_iterations);

  gsl_multimin_fdfminimizer_free (minimizer_ptr);   // free the minimizer
  gsl_vector_free (xvec_ptr);     // free the vector

  return 0;
}


//*************************************************************************
// Simple paraboloid centered on (dp[0],dp[1])

double
my_f (const gsl_vector *xvec_ptr, void *params)
{
  double *dp = (double *)params;

  double x = gsl_vector_get(xvec_ptr, 0);
  double y = gsl_vector_get(xvec_ptr, 1);

  return ( 10.0 * sqr(x - dp[0]) + 20.0 * sqr(y - dp[1]) + 30.0 );
}

// The gradient of f, df = (df/dx, df/dy).
void
my_df (const gsl_vector *xvec_ptr, void *params,
       gsl_vector *df_ptr)
{
  double *dp = (double *)params;

  double x = gsl_vector_get(xvec_ptr, 0);
  double y = gsl_vector_get(xvec_ptr, 1);

  gsl_vector_set(df_ptr, 0, 20.0 * (x - dp[0]));
  gsl_vector_set(df_ptr, 1, 40.0 * (y - dp[1]));
}

// Compute both f and df together.
void
my_fdf (const gsl_vector *xvec_ptr, void *params_ptr,
        double *f_ptr, gsl_vector *df_ptr)
{
  *f_ptr = my_f(xvec_ptr, params_ptr);
  my_df(xvec_ptr, params_ptr, df_ptr);
}


//*************************************************************************
```