```
//  file: ode_test.cpp
//
//  C++ Program to test the ode differential equation solver from
//   the gsl numerical library.
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//      12/27/03  original C++ version, modified from C version
//      02/13/04  added math.h
//      02/06/06  switched to cmath and tidied up code
//
//  Notes:
//   * Example taken from the GNU Scientific Library Reference Manual
//      Edition 1.1, for GSL Version 1.1 9 January 2002
//      URL: gsl/ref/gsl-ref_23.html#SEC364
//   * Compile and link with:
//       g++ -Wall -o ode_test ode_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
//
// The following program solves the second-order nonlinear
//  Van der Pol oscillator equation (see background notes),
//
//     x"(t) + \mu x'(t) (x(t)^2 - 1) + x(t) = 0
//
// This can be converted into a first order system suitable for
//  use with the library by introducing a separate variable for
//  the velocity, v = x'(t).  We assign x --> y[0] and v --> y[1].
//  So the equations are:
// x' = v                  ==>  dy[0]/dt = f[0] = y[1]
// v' = -x + \mu v (1-x^2) ==>  dy[1]/dt = f[1] = -y[0] + mu*y[1]*(1-y[0]*y[0])
//
//*****************************************************************

// include files
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>    // C++ stringstream class (can omit iostream)
#include <cmath>
using namespace std;

#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_odeiv.h>

// function prototypes
int rhs (double t, const double y[], double f[], void *params_ptr);
int jacobian (double t, const double y[], double *dfdy,
        double dfdt[], void *params_ptr);

//************************* main program ****************************

int
main (void)
{
  const int dimension = 2;  // number of differential equations

  const double eps_abs = 1.e-8;          // absolute error requested
  const double eps_rel = 1.e-10;  // relative error requested
```



```
  // Define the type of routine for making steps:
  const gsl_odeiv_step_type *type_ptr = gsl_odeiv_step_rkf45;
  // some other possibilities (see GSL manual):
  //   = gsl_odeiv_step_rk4;
  //   = gsl_odeiv_step_rkck;
  //   = gsl_odeiv_step_rk8pd;
  //   = gsl_odeiv_step_rk4imp;
  //   = gsl_odeiv_step_bsimp;
  //   = gsl_odeiv_step_gear1;
  //   = gsl_odeiv_step_gear2;

  // Allocate/initialize the stepper, the control function, and the
  //  evolution function.
  gsl_odeiv_step *step_ptr = gsl_odeiv_step_alloc (type_ptr, dimension);
  gsl_odeiv_control *control_ptr = gsl_odeiv_control_y_new (eps_abs, eps_rel);
  gsl_odeiv_evolve *evolve_ptr = gsl_odeiv_evolve_alloc (dimension);

  gsl_odeiv_system my_system;  // structure with the rhs function, etc.

  double mu = 2;    // parameter for the diffeq

  // Load values into the my_system structure
  my_system.function = rhs;  // the right-hand-side functions dy[i]/dt
  my_system.jacobian = jacobian;  // the Jacobian df[i]/dy[j]
  my_system.dimension = dimension;  // number of diffeq's
  my_system.params = &mu;  // parameters to pass to rhs and jacobian

  double tmin = 0.;    // starting t value
  double tmax = 100.;    // final t value
  double delta_t = 0.01;         // step size in time
  double t = tmin;    // initialize t

  double y[2];      // current solution vector
  y[0] = -1.5;      // initial x value
  y[1] = 2.0;      // initial v value

  // Set up a file names with the initial values
  ostringstream my_stringstream;  // declare a stringstream object
  my_stringstream << "ode_test" << "_x0_" << setprecision(2) << y[0]
                  << "_v0_" << setprecision(2) << y[1] << ".dat";

  ofstream ode_out;    // now open a stream to a file for output
  // use .str() to convert to a string, then .c_str() to convert to a char *
  ode_out.open(my_stringstream.str().c_str());

  // print initial values and column headings
  ode_out << "# Running ode_test with x0 = " << setprecision(2) << y[0]
          << " and v0 = " << setprecision(2) << y[1] << endl;
  ode_out << "#   t      x      v  " << endl;
  ode_out << scientific << setprecision (5) << setw (12) << t << " "
          << setw (12) << y[0] << " " << setw (12) << y[1] << endl;

  // step to tmax from tmin
  double h = 1e-6;    // starting step size for ode solver
  for (double t_next = tmin + delta_t; t_next <= tmax; t_next += delta_t)
  {
    while (t < t_next)  // evolve from t to t_next
    {
      gsl_odeiv_evolve_apply (evolve_ptr, control_ptr, step_ptr,
                              &my_system, &t, t_next, &h, y);
    }

    // print at t = t_next
    ode_out << scientific << setprecision (5) << setw (12) << t << " "
            << setw (12) << y[0] << " " << setw (12) << y[1] << endl;
  }
  ode_out.close();

  // all done; free up the gsl_odeiv stuff
```

```
  gsl_odeiv_evolve_free (evolve_ptr);
  gsl_odeiv_control_free (control_ptr);
  gsl_odeiv_step_free (step_ptr);

  return 0;
}

//************************** rhs ***************************
//
// Define the array of right-hand-side functions y[i] to be integrated.
//  The equations are:
// x' = v                  ==>  dy[0]/dt = f[0] = y[1]
// v' = -x + \mu v (1-x^2) ==>  dy[1]/dt = f[1] = -y[0] + mu*y[1]*(1-y[0]*y[0])
//
//  * params is a void pointer that is used in many GSL routines
//     to pass parameters to a function
//
int
rhs (double , const double y[], double f[], void *params_ptr)
{
  // get parameter(s) from params_ptr; here, just a double
  double mu = *(double *) params_ptr;

  // evaluate the right-hand-side functions at t
  f[0] = y[1];
  f[1] = -y[0] + mu * y[1] * (1. - y[0] * y[0]);

  return GSL_SUCCESS;    // GSL_SUCCESS defined in gsl/errno.h as 0
}

//************************** Jacobian **************************
//
// Define the Jacobian matrix using GSL matrix routines.
//  (see the GSL manual under "Ordinary Differential Equations")
//
//  * params is a void pointer that is used in many GSL routines
//     to pass parameters to a function
//
int
jacobian (double , const double y[], double *dfdy,
    double dfdt[], void *params_ptr)
{
  // get parameter(s) from params_ptr; here, just a double
  double mu = *(double *) params_ptr;

  gsl_matrix_view dfdy_mat = gsl_matrix_view_array (dfdy, 2, 2);

  gsl_matrix *m_ptr = &dfdy_mat.matrix;  // m_ptr points to the matrix

  // fill the Jacobian matrix as shown
  gsl_matrix_set (m_ptr, 0, 0, 0.0);  // df[0]/dy[0] = 0
  gsl_matrix_set (m_ptr, 0, 1, 1.0);  // df[0]/dy[1] = 1
  gsl_matrix_set (m_ptr, 1, 0, -2.0 * mu * y[0] * y[1] - 1.0);  // df[1]/dy[0]
  gsl_matrix_set (m_ptr, 1, 1, -mu * (y[0] * y[0] - 1.0));      // df[1]/dy[1]

  // set explicit t dependence of f[i]
  dfdt[0] = 0.0;
  dfdt[1] = 0.0;

  return GSL_SUCCESS;    // GSL_SUCCESS defined in gsl/errno.h as 0
}
```