```
//  file: ode_test_class.cpp
//
//  C++ Program to test the ode differential equation solver from
//   the gsl numerical library using the Ode class wrapper for gsl.
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//     02/10/09  Original version based on ode_test.cpp
//
//  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 GslOde class files:
//       g++ -Wall -c ode_test_class.cpp
//       g++ -Wall -c GslOde.cpp
//       g++ -Wall -o ode_test_class ode_test_class.o GslOde.o -lgsl -lgslcblas
//
//*****************************************************************

// 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 "GslOde.h"    // Ode class for gsl (include Ode and Rhs)

// Class for the right side of the Van der Pol equation.
//  Derived from the Rhs class.
//  The VdP equation depends on one parameter mu.
class Rhs_VdP : public Rhs
{
  public:
    Rhs_VdP (double mu_passed) {mu = mu_passed; num_eqs = 2;};
    ~Rhs_VdP () {};
    virtual int rhs (double t, const double y[], double f[]);
    virtual int jacobian (double t, const double y[], double *dfdy,
                          double dfdt[]);
    double get_mu () {return mu;};
  private:
    double mu;    // Van der Pol parameter
};

// Function to evolve the differential equation and print the output
int evolve_and_print(Ode &vdp_ode,  Rhs_VdP &vdp_rhs, const double x0,
                     const double v0, const double tmin,
                     const double tmax, const double delta_t);
```



```
//************************** main program ****************************
int
main ()
{
  double x0 = 0.0;          // initial conditions
  double v0 = 0.0;

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

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

  double mu = 2;             // parameter for the diff-eq
  Rhs_VdP vdp_rhs_1 (mu);    // set up the right side of the diff-eq

  Ode vdp_ode_1 (vdp_rhs_1, eps_abs, eps_rel, "rk45");
  x0 = -1.5;
  v0 = 2.0;
  evolve_and_print(vdp_ode_1, vdp_rhs_1, x0, v0, tmin, tmax, delta_t);

  return 0;
}

//*****************************************************
int
evolve_and_print(Ode &vdp_ode, Rhs_VdP &vdp_rhs, const double x0,
                 const double v0, const double tmin, const double tmax,
                 const double delta_t)
{
  double y[2];     // current solution vector
  y[0] = x0;       // initial x value
  y[1] = v0;       // initial v value

  double t = tmin;         // initialize t
  // Set up a file name with the initial values
  ostringstream my_stringstream;  // declare a stringstream object
  my_stringstream << "ode_test_class" << "_mu_" << setprecision(2)
                  << vdp_rhs.get_mu()
                  << "_x0_" << setprecision(2) << y[0]
                  << "_v0_" << setprecision(2) << y[1] << ".dat";
  ofstream my_out;    // now open a stream to a file for output
  my_out.open(my_stringstream.str().c_str());

  // print initial values and column headings
  my_out << "# Running ode_test with x0 = " << setprecision(2) << y[0]
         << " and v0 = " << setprecision(2) << y[1] << endl;
  my_out << "#   t      x      v  " << endl;
  my_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
    {
      vdp_ode.evolve ( &t, t_next, &h, y );
    }

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

  return (0);    // successful completion
}
```

```
//*******************************************************
//*******************************************************
//
// 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])
int
Rhs_VdP::rhs (double, const double y[], double f[])
{
  // std::cout << "rhs called with y[0] = " << y[0] << endl;
  // 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 0;    // successful completion
}

//
// Define the Jacobian matrix of df_i/dy_j for i,j = {0,1}
int
Rhs_VdP::jacobian (double, const double y[], double *, double dfdt[])
{
  // fill the Jacobian matrix
  set_jacobian (0, 0, 0.0);                              // df[0]/dy[0] = 0
  set_jacobian (0, 1, 1.0);                              // df[0]/dy[1] = 1
  set_jacobian (1, 0, -2.0 * mu * y[0] * y[1] - 1.0);  // df[1]/dy[0]
  set_jacobian (1, 1, -mu * (y[0] * y[0] - 1.0));       // df[1]/dy[1]

  // set explicit t dependence of f[i] (none here)
  dfdt[0] = 0.0;
  dfdt[1] = 0.0;

  return 0;    // successful completion
}
```



```
//  file: GslOde.h
//
//  Header file for the GSL versions of the Ode and Rhs C++ classes.
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//      02/10/09  original version
//
//  Notes:
//   * We encapsulate GSL ode functions in this class, based on the
//      documentation for the GSL library under "Differential Equations".
//   * The GSL header file is included in GslOde.cpp and
//      anywhere we want to create Ode and Rhs objects.
//   * See the GSL documentation for error handling details.
//
//*************************************************************
// The ifndef/define macro ensures that the header is only included once
#ifndef GSLODE_H
#define GSLODE_H

// include files
#include <string>     // C++ strings

#include <gsl/gsl_odeiv.h>   // header for gsl ode solvers
#include <gsl/gsl_matrix.h>  // header for gsl matrices

class Rhs
{
  public:
    Rhs () {};
    virtual ~Rhs () {};
    virtual int rhs (double, const double *, double *) {return (1);};
    virtual int jacobian (double, const double *, double *,
                          double *) {return (1);};
    static int gsl_rhs (double t, const double y[], double f[], void *);
    static int gsl_jacobian (double t, const double y[], double *dfdy,
                             double dfdt[], void *);
    int get_num_eqs () {return num_eqs;};

  protected:
    int num_eqs;
    void set_jacobian (int i, int j, double value);
    gsl_matrix_view dfdy_mat;
    gsl_matrix *m_ptr;

  private:
};

class Ode
{
  public:
    Ode (Rhs &this_rhs, double eps_abs, double eps_rel, std::string type);
    ~Ode ();
    void evolve (double *t, double t_next, double *h, double y[]);

  private:
    int ode_dim;                // number of equations
    std::string ode_type;       // type of Ode
    const gsl_odeiv_step_type *type_ptr;
    // The GSL steeper, control function, and the evolution function.
    gsl_odeiv_step *step_ptr;
    gsl_odeiv_control *control_ptr;
    gsl_odeiv_evolve *evolve_ptr;

    gsl_odeiv_system ode_system;  // structure with the rhs function, etc.
};

#endif
```

```
//  file: GslOde.cpp
//
//  Definitions for the GSL versions of the Ode and Rhs C++ classes.
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//     02/10/09  Original version using ode_test.cpp as a guide.
//
//*****************************************************************
// include files
#include <iostream>
#include <string>     // C++ strings
#include <gsl/gsl_errno.h>   // header for gsl error handling
#include <gsl/gsl_matrix.h>  // header for gsl matrices
#include <gsl/gsl_odeiv.h>   // header for gsl ode solvers

#include "GslOde.h"       // include the header for these classes

//*****************************************************************

// Constructor for Ode
Ode::Ode(Rhs &passed_rhs, double eps_abs, double eps_rel, std::string type)
{
  ode_type = type;    // set the private variable for the Ode type
  ode_dim = passed_rhs.get_num_eqs();

  // Allocate the ode according to ode_type
  // some possibilities (see GSL manual):
  //    gsl_odeiv_step_rk4;
  //    gsl_odeiv_step_rkf45;
  //    gsl_odeiv_step_rkck;
  //    gsl_odeiv_step_rk8pd;
  //    gsl_odeiv_step_rk4imp;
  //    gsl_odeiv_step_bsimp;
  //    gsl_odeiv_step_gear1;
  //    gsl_odeiv_step_gear2;
  //
  if (ode_type == "rk4")
  {
    type_ptr = gsl_odeiv_step_rk4;
  }
  else if (ode_type == "rk45")
  {
    type_ptr = gsl_odeiv_step_rkf45;
  }
  else
  {
    std::cout << "Illegal ode type for GSL!" << std::endl;
    exit (1);  // time to quit!
  }

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

  // Load values into the ode_system structure
  ode_system.function = passed_rhs.gsl_rhs;       // functions dy[i]/dt
  ode_system.jacobian = passed_rhs.gsl_jacobian;  //  Jacobian df[i]/dy[j]
  ode_system.dimension = ode_dim;    // number of diffeq's
  ode_system.params = &passed_rhs;   // pass the Rhs object
}

Ode::~Ode () // Destructor for Ode
{
  // 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);
}

void
Ode::evolve (double *t, double t_next, double *h, double *y)
{
  gsl_odeiv_evolve_apply (evolve_ptr, control_ptr, step_ptr,
                          &ode_system, t, t_next, h, y);
}

//*****************************************************************

int
Rhs::gsl_rhs (double t, const double y[], double f[], void * Rhs_ptr)
{
  ((Rhs*)Rhs_ptr)->rhs (t, y, f);
  return 0;    // successful completion
}

int
Rhs::gsl_jacobian (double t, const double y[], double *dfdy,
                   double dfdt[], void * Rhs_ptr)
{
  int ode_dim = ((Rhs*)Rhs_ptr)->get_num_eqs();
  ((Rhs*)Rhs_ptr)->dfdy_mat = gsl_matrix_view_array(dfdy, ode_dim, ode_dim);
  // m_ptr points to the matrix
  ((Rhs*)Rhs_ptr)->m_ptr = &((Rhs*)Rhs_ptr)->dfdy_mat.matrix;

  ((Rhs*)Rhs_ptr)->jacobian (t, y, dfdy, dfdt);
  return 0;   // successful completion
}

void
Rhs::set_jacobian (int i, int j, double value)
{
  // Define the Jacobian matrix using GSL matrix routines.
  //  (see the GSL manual under "Ordinary Differential Equations")
  gsl_matrix_set (m_ptr, i, j, value);
}

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