```
//  file: diffeq_pendulum.cpp
//
//  Program to solve the differential equation for a physical
//   (chaotic) pendulum, as in Chapter 14 of the Landau/Paez text
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//      02/10/04  original version, translated from diffeq_pendulum.c
//      02/17/04  including direct piping to gnuplot (from gnuplot_i.c)
//      01/30/06  put declarations and initializations together;
//                 switched to <cmath>
//      02/05/06  switched to GnuplotPipe class
//
//  Notes:
//   * Based on the discussion of differential equations in Chap. 9
//      of "Computational Physics" by Landau and Paez and of
//      differential chaos in phase space in Chap. 14.
//   * Uses the fourth-order Runge-Kutta ode routine (equal step)
//   * Angular position is theta(t) and angular velocity is theta_dot(t)
//   * We've added _ext to the driving force (for "external")
//
//*****************************************************************
// include files
#include <iostream>     // note that .h is omitted
#include <iomanip>     // note that .h is omitted
#include <fstream>     // note that .h is omitted
#include <string>
using namespace std;     // we need this when .h is omitted
#include <cmath>
#include "diffeq_routines.h"   // diffeq routine prototypes
#include "GnuplotPipe.h"   // direct piping

// function prototypes
double rhs (double t, double y[], int i, void *params_ptr);
double potential (double x, void *params_ptr);

// structures
typedef struct      // define a type to hold parameters
{
  double omega0;     // natural frequency
  double alpha;       // coefficient of friction
  double f_ext;       // amplitude of external force
  double omega_ext;     // frequency of external force
  double phi_ext;     // phase angle for external force
}
force_parameters;     // now we can define a structure of this type
        //   using the keyword "force_parameters"

//************************** main program **************************
int
main (void)
{
  const double pi = M_PI;  // use the system-defined 3.14159...

  const int N = 2;     // 2nd order equation --> 2 coupled 1st
  double y_rk4[N];     // vector of y functions

  void *rhs_params_ptr;     // void pointer passed to functions
  force_parameters rhs_parameters;  // parameters for the function

  // initialize force parameters and initial conditions
  double f_ext = 0.2;
  double omega_ext = 0.689;
  double T_ext = 2. * pi / omega_ext;  // period for external frequency
  double phi_ext = 0.;
  double omega0 = 1.;
  double alpha = 0.2;
  double theta0 = 0.8;     // initial (angular) position
```



```
  double theta_dot0 = 0.0;      // initial (angular) velocity

  int T_skip = 1000;     // every T_skip points means once every T_ext
  double h = T_ext / double(T_skip);  // initialize mesh spacing
  double tmin = 0.;       // starting t value
  double tmax = 50.;       // last t value
  double plot_min = tmin;     // first t value to plot
  double plot_max = tmax;     // last t value to plot
  int plot_skip = 10;       // plot every plot_skip points
  int plot_delay = 10;                  // wait plot_delay msec between points

  // declare a GnuplotPipe object and set some properties
  GnuplotPipe myPipe;
  myPipe.set_title ("Pendulum Phase Space");
  myPipe.set_xlabel ("theta");
  myPipe.set_ylabel ("theta_dot");

  int answer = 1, answer2 = 1;  // answer to parameter and continue queries
  while (answer2 != 0)     // iterate until told to move on
  {
    answer = 1;
    while (answer != 0)  // iterate until told to move on
    {
      cout << "\nCurrent parameters:\n";
      cout << " [1] omega0 = " << omega0 << "\t\t";
      cout << "[2] alpha = " << alpha << endl;
      cout << " [3] f_ext = " << f_ext << "\t";
      cout << "[4] w_ext = " << omega_ext << "\t";
      cout << "[5] phi_ext = " << phi_ext << endl;
      cout << " [6] theta0 = " << theta0 << "\t";
      cout << "[7] theta_dot0 = " << theta_dot0 << endl;
      cout << " [8] t_start = " << tmin << "\t";
      cout << "[9] t_end = " << tmax << "\t\t";
      cout << "[10] h = " << h << endl;
      cout << "[11] plot_start = " << plot_min << "\t";
      cout << "[12] plot_end = " << plot_max << "\t";
      cout << "[13] plot_skip = " << plot_skip << endl;
      cout << "[14] Gnuplot_delay = " << plot_delay << endl;
      cout << "\nWhat do you want to change? [0 for none] ";

      cin >> answer;
      cout << endl;

      switch (answer)
      {
        case 0:
          break;
        case 1:
          cout << " enter omega0: "; cin >> omega0;
          break;
        case 2:
          cout << " enter alpha: "; cin >> alpha;
          break;
        case 3:
          cout << " enter f_ext: "; cin >> f_ext;
          break;
        case 4:
          cout << " enter w_ext: "; cin >> omega_ext;
          T_ext = 2. * pi / omega_ext;
          h = T_ext / 1000.;  // set h according to external period
          break;
        case 5:
          cout << " enter phi_ext: "; cin >> phi_ext;
          break;
        case 6:
          cout << " enter theta0: "; cin >> theta0;
          break;
        case 7:
          cout << " enter theta_dot0: "; cin >> theta_dot0;
```

```
        break;
      case 8:
        cout << " enter t_start: "; cin >> tmin;
        break;
      case 9:
        cout << " enter t_end: "; cin >> tmax;
        break;
      case 10:
        cout << " enter h: "; cin >> h;
        break;
      case 11:
        cout << " enter plot_start: "; cin >> plot_min;
        break;
      case 12:
        cout << " enter plot_end: "; cin >> plot_max;
        break;
      case 13:
        cout << " enter plot_skip: "; cin >> plot_skip;
        break;
      case 14:
        cout << " enter Gnuplot_delay (in msec): "; cin >> plot_delay;
        myPipe.set_delay (1000*plot_delay);   // set_delay in usec
      default:
        break;
    }  // end switch answer
  }    // end answer while

  cout << "Plotting now (wait until complete) . . ." << endl;

  // open the output file
  ofstream out;    // declare the output file
  out.open ("diffeq_pendulum.dat", ofstream::trunc);

  // load the force parameters into the structure
  rhs_parameters.omega0 = omega0;
  rhs_parameters.alpha = alpha;
  rhs_parameters.f_ext = f_ext;
  rhs_parameters.omega_ext = omega_ext;
  rhs_parameters.phi_ext = phi_ext;
  rhs_params_ptr = &rhs_parameters;  // structure to pass to function

  myPipe.init ();  // start up piping to gnuplot

  y_rk4[0] = theta0;    // initial condition for y0(t)
  y_rk4[1] = theta_dot0;    // initial condition for y1(t)

  // print out the parameters, a header, and the first set of points
  out << "# omega0=" << omega0 << ", alpha=" << alpha << endl;
  out << "# theta0=" << theta0 << ", theta_dot0=" << theta_dot0 << endl;
  out << "# t_start=" << tmin << ", t_end=" << tmax << ", h="
      << h << endl;
  out << "#  t     x(t)        v(t)    " << endl;

  if (tmin >= plot_min)
  {
    out << tmin << " " << scientific << setprecision (15)
        << y_rk4[0] << " " << y_rk4[1] << endl;
    myPipe.plot (theta0, theta_dot0);  // plot 1st point
    myPipe.plot2 (theta0, theta_dot0);  // plot 1st point
  }

  int point_count = 0;    // initialize point counter
  for (double t = tmin; t <= tmax; t += h)
  {
    // find y(t+h) by a 4th order Runge-Kutta step
    runge4 (N, t, y_rk4, h, rhs, rhs_params_ptr);

    if ((t >= plot_min) & (t <= plot_max))
    {
```



```
      point_count++;  // increment point counter

      double theta = y_rk4[0];       // current angle
      double theta_dot = y_rk4[1];   // current angular velocity

      if ((point_count % plot_skip) == 0)
      {    // plot every plot_skip points
        out << t + h << " " << scientific << setprecision (15)
            << theta << " " << theta_dot << endl;
                // send points to gnuplot
        myPipe.plot (theta, theta_dot);
      }
      if ((point_count % T_skip) == 0)
      {    // plot every T_skip points
           // send points to gnuplot
        myPipe.plot2 (theta, theta_dot);
      }
    }
  }  // end for loop over t

  out << endl;
  out.close ();    // close the output file
  cout << "\n results added to diffeq_pendulum.dat\n\n";

  cout << "Again? (no=0, clear=1) ";
  cin >> answer2;

  cout << "Wait while the pipe is closed . . . " << flush;
  myPipe.finish ();  // close the pipe to gnuplot
  cout << "ok, all done!" << endl;
}  // end answer2 while loop

  return (0);      // successful completion!
}

//************************** rhs ***************************
//
//  * This is the function defining the i'th right hand side of
//     the diffential equations:
//            dy[i]/dt = rhs(t,y[],i)
//  * We take this from eqs. (14.5) through (14.7) in Landau/Paez
//
//**********************************************************
double
rhs (double t, double y[], int i, void *params_ptr)
{
  // define local force parameters from passed structure
  double omega0 = ((force_parameters *) params_ptr)->omega0;
  double alpha = ((force_parameters *) params_ptr)->alpha;
  double f_ext = ((force_parameters *) params_ptr)->f_ext;
  double omega_ext = ((force_parameters *) params_ptr)->omega_ext;
  double phi_ext = ((force_parameters *) params_ptr)->phi_ext;

  // External force
  double F_ext = f_ext * cos (omega_ext * t + phi_ext);

  if (i == 0)
  {
    return (y[1]);
  }

  if (i == 1)
  {
    return (-omega0 * omega0 * sin (y[0]) - alpha * y[1] + F_ext);
  }

  return (1);      // something's wrong if we get here
}
```