```
//  file: diffeq_oscillations.cpp
//
//  Program to solve the differential equation for a driven anharmonic
//   oscillator, as described in Chapter 9 of the Landau/Paez text
//
//  Programmer:  Dick Furnstahl  furnstahl.1@osu.edu
//
//  Revision history:
//      02/09/04  original version, translated from diffeq_oscillations.c
//      01/28/05  changes to comments plus <math.h> added
//      01/30/06  switched to <cmath>
//
//  Notes:
//   * Based on the discussion of differential equations in Chap. 9
//      of "Computational Physics" by Landau and Paez
//   * Uses the fourth-order Runge-Kutta ode routine (equal step)
//   * As a convention (advocated in "Practical C++"), we'll append
//      "_ptr" to all pointers.
//
//*************************************************************
// include files
#include <iostream>              // note that .h is omitted
#include <iomanip>               // note that .h is omitted
#include <fstream>               // note that .h is omitted
using namespace std;             // we need this when .h is omitted
#include <cmath>
#include "diffeq_routines.h"           // diffeq routine prototypes

// 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 m;                      // mass of particle
  double k;                      // coefficient of potential
  double p;                      // exponent of oscillator
  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 ()
{
  const double pi = 4.*atan (1.);  // fancy definition of pi

  const int N = 2;               // 2nd order equation -->
                                 //   2 coupled 1st order equations
  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.;
  double omega_ext = 1.;
  double phi_ext = 0.;

  double m = 1.;
  double k = pow (2. * pi, 2);
  double p = 2.;
  double x0 = 0.;                // initial position
  double v0 = 1.;                // initial velocity

  double h = 0.001;              // initialize mesh spacing
  double tmin = 0.;              // starting t value
```



```
  double tmax = 15.;             // last t value
  int plot_skip = 10;            // plot every plot_skip points

  int answer2 = 2;               //answer to continue query
  while (answer2 != 0)           // iterate until told to move on
    {
      int answer = 1;            //answer to parameter query
      while (answer != 0)        // iterate until told to move on
        {
          cout << "\nCurrent parameters:\n";
          cout << " [1] m = " << setprecision(4) << m << "\t";
          cout << "[2] k = " << setprecision(7) << k << "\t";
          cout << "[3] p = " << setprecision(2) << p << endl;
          cout << " [4] f_ext = " << setprecision(5) << f_ext << "\t";
          cout << "[5] w_ext = " << setprecision(5) << omega_ext << "\t\t";
          cout << "[6] phi_ext = " << setprecision(2) << phi_ext << endl;
          cout << " [7] x0 = " << setprecision(5) << x0 << "\t";
          cout << "[8] v0 = " << setprecision(5) << v0 << endl;
          cout << " [9] t_min = " << setprecision(5) << tmin << "\t";
          cout << "[10] t_max = " << setprecision(5) << tmax << "\t\t";
          cout << "[11] h = " << setprecision(5) << h << endl;
          cout << "[12] plot_skip = " << plot_skip << endl;
          cout << "\nWhat do you want to change? [0 for none] ";
          cin >> answer;
          cout << endl;

          switch (answer)
            {
            case 0:
              break;
            case 1:
              cout << " enter m: ";
              cin >> m;
              break;
            case 2:
              cout << " enter k: ";
              cin >> k;
              break;
            case 3:
              cout << " enter p: ";
              cin >> p;
              break;
            case 4:
              cout << " enter f_ext: ";
              cin >> f_ext;
              break;
            case 5:
              cout << " enter w_ext: ";
              cin >> omega_ext;
              break;
            case 6:
              cout << " enter phi_ext: ";
              cin >> phi_ext;
              break;
            case 7:
              cout << " enter x0: ";
              cin >> x0;
              break;
            case 8:
              cout << " enter v0: ";
              cin >> v0;
              break;
            case 9:
              cout << " enter t_start: ";
              cin >> tmin;
              break;
            case 10:
              cout << " enter t_end: ";
              cin >> tmax;
```

```
            break;
          case 11:
            cout << " enter h: ";
            cin >> h;
            break;
          case 12:
            cout << " enter plot_skip: ";
            cin >> plot_skip;
            break;
          default:
            break;
          }
      }

    // open the output file in append mode ==> multiple plots
    //   or open a new plot file
    ofstream out;                   // name the output file
    if (answer2 == 2)  // open a new file
      {
        out.open ("diffeq_oscillations.dat", ofstream::trunc);
      }
    else  // append
      {
        out.open ("diffeq_oscillations.dat", ofstream::app);
      }

    //load the force parameters into the structure
    rhs_parameters.k = k;
    rhs_parameters.m = m;
    rhs_parameters.p = p;
    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

    double y_rk4[N];          // vector of y functions
    y_rk4[0] = x0;            // initial condition for y(t)
    y_rk4[1] = v0;            // initial condition for y'(t)

    // print out the parameters, a header, and the first set of points
    out << "# m=" << m << ", k=" << k << ", p=" << p << endl;
    out << "# x0=" << x0 << ", v0=" << v0 << endl;
    out << "# t_start=" << tmin << ", t_end=" << tmax << ", h="
        << h << endl;
    out << "#  t      x(t)        v(t)     ";
    out << "     KE(t)        PE(x(t)) \n";
    out << fixed << setprecision(4) << tmin << " "
        << scientific << setprecision(15) << y_rk4[0] << " "
        << scientific << setprecision(15) << y_rk4[1] << " "
        << scientific << setprecision(15) << m*v0*v0/2. << " "
        << scientific << setprecision(15) <<
             potential (x0, rhs_params_ptr) << endl;

    cout << "Initial KE: " << m * v0 * v0 / 2.
         << "  Initial PE: " << potential (x0, rhs_params_ptr)
         << "  Initial E: " <<
            m * v0 * v0 / 2. + potential (x0, rhs_params_ptr)
         << endl;

    int point_count = 0;              // initialize point counter
    double t;                         // independent variable
    double x, v;                      // local position and velocity
    for (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);
        point_count++;        // increment point counter

        x = y_rk4[0];
```



```
        v = y_rk4[1];

        if ((point_count % plot_skip) == 0)
          {                   // plot every plot_skip points
            out << fixed << setprecision(4) << t+h << " "
                << scientific << setprecision(15) << x << " "
                << scientific << setprecision(15) << v << " "
                << scientific << setprecision(15) << m*v*v/2. << " "
                << scientific << setprecision(15) <<
                     potential (x, rhs_params_ptr) << endl;
          }
      }

    cout << "\n results added to diffeq_oscillations.dat\n\n";
    out << endl;

    out.close ();             // close the output file

    cout << "Again? (no=0, append=1, clear=2) ";
    cin >> answer2;
  }
  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)
//
//*******************************************************
double
rhs (double t, double y[], int i, void *params_ptr)
{
  double x = y[0];            // local x value

  double k = ((force_parameters *) params_ptr)->k;   // local force parameters
  double m = ((force_parameters *) params_ptr)->m;
  double p = ((force_parameters *) params_ptr)->p;

  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;

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

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

  if (i == 1)  // second equation
    {
      if (x == 0)
        {
          return (F_ext / m);
        }
      else if (x < 0)
        {
          return ((F_ext + k * pow (fabs (x), (p - 1))) / m);
        }
      else if (x > 0)
        {
          return ((F_ext - k * pow (fabs (x), (p - 1))) / m);
        }
    }

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

```
//************************* potential *********************
//
//  * potential corresponding to the force law
//       V(x) = (1/p)*k*|x|^p
//
//*********************************************************
double
potential (double x, void *params_ptr)
{
  double k = ((force_parameters *) params_ptr)->k;
  double p = ((force_parameters *) params_ptr)->p;

  return (k * pow (fabs (x), p) / p);
}
```