

Nonlinear Driven Damped Pendulum

Simple Pendulum with Damping and Driving Force

The simple pendulum is perhaps the most familiar physical system that exhibits simple harmonic motion. The pendulum clock was the most accurate timekeeper from its invention by Huygens in 1656 until the invention of the quartz oscillator clock in 1927.

In addition to the force of gravity, a realistic pendulum clock experiences friction from the air and its bearings, and a periodic driving force usually supplied by a wound spring.

The equation of motion for a driven and damped simple pendulum is

$$\frac{d^2\theta}{dt^2} = -\frac{g}{\ell}\theta - q\frac{d\theta}{dt} + F_D \sin(\Omega_D t), \quad (1)$$

where ℓ is the length of the pendulum, g the acceleration due to gravity, q is the damping constant, and F_D is the amplitude of a periodic driving force which has angular frequency Ω_D .

The natural angular frequency of the pendulum in the absence of damping is

$$\Omega = \sqrt{\frac{g}{\ell}}. \quad (2)$$

The damping force causes initial *transient* behavior

$$\theta(t) = \theta_0 e^{-qt/2} \sin\left(\sqrt{\Omega^2 - q^2/4} t + \phi\right). \quad (3)$$

This transient motion is underdamped if $\Omega > q/2$, critically damped if $\Omega = q/2$, and overdamped if $\Omega < q/2$.

After transients have died out the motion is determined by driving force which supplies energy to overcome frictional forces. The equation for forced oscillations can be solved analytically:

$$\theta(t) = \frac{F_D \sin(\Omega_D t + \phi)}{\sqrt{(\Omega^2 - \Omega_D^2)^2 + (q\Omega_D)^2}}. \quad (4)$$

Nonlinear Damped and Driven Pendulum

The pendulum model becomes even more interesting if the amplitude of motion is not limited to small oscillations:

$$\frac{d^2\theta}{dt^2} = -\frac{g}{\ell} \sin \theta - q\frac{d\theta}{dt} + F_D \sin(\Omega_D t). \quad (5)$$

The pendulum has an equilibrium point at $\theta = 0$ with the bob vertically below the pivot point. The point $\theta = \pm\pi$ is an unstable equilibrium point with the bob vertically above the pivot point. In addition to oscillations, the pendulum can now undergo rotational motion.

It is interesting to study the behavior of the pendulum as the driving force is varied. For small F_D , friction dominates and the motion is damped. For large F_D , the driving force dominates and the motion is generally periodic with period determined by Ω_D . For intermediate values of F_D , there is a complicated interaction between driving, nonlinearity, and dissipation: for particular parameter values, the motion becomes *chaotic* (numerically unpredictable).

The Kolmogorov-Arnold-Moser Theorem

This theorem addresses the fundamental problem of Hamiltonian dynamics.

The phase trajectories of an integrable system are confined to an invariant torus, which is an N -dimensional donut shaped surface on which N invariants have constant values. In general, these trajectories are multiply-period and stable.

Now suppose that *non-integrable* perturbations are added to H . Does the system remain stable and periodic, or can it become chaotic? Will Earth orbit the Sun forever, or can the orbit suddenly be destabilized by perturbing tugs from the other planets?

Kolmogorov, Arnold and Moser[3] proved that if the perturbation is “sufficiently weak”, then

- Some invariant tori of the integrable system are *deformed*, but trajectories on them remain multiply periodic and stable. If Earth is on such a torus, it will orbit the Sun forever.
- Other invariant tori are destroyed, and motion on them becomes non-periodic. If neighboring phase points diverge exponentially, then the motion is said to be chaotic.
- Tori with “sufficiently irrational” periods ratios survive. Tori with periods whose ratios are rational numbers, which cause *resonances*, tend to disintegrate.

Phase Space Terminology

The State Space[1] of a system is a space in which each possible state of the system is represented by a unique point. For a time-independent Hamiltonian system points in phase space represent unique states of the system.

The physical pendulum with moment of inertia I , and angle θ with the downward vertical direction, is described the Hamiltonian

$$H(\theta, p_\theta) = \frac{p_\theta^2}{2I} + mg(1 - \cos \theta), \quad (6)$$

in the absence of driving force and friction. The invariant-torus structure of the phase space has the following characteristics:

- A *stable elliptic fixed point* at $\theta = 0$.
- An *unstable hyperbolic fixed point* at $\theta = \pi$.
- *Bound trajectories*, which are oscillations around the elliptic fixed point.
- *Unbounded trajectories*, which are rotations about the pendulum pivot in the clockwise or counterclockwise directions.
- Phase regions of bound and unbounded trajectories are delimited by a *separatrix*.

A periodic driving force is a non-integrable perturbation. When its amplitude F_D is increased from zero, invariant tori near a separatrix are first to disintegrate. Disintegrating tori develop *island chains* of stable regions, which disintegrate in turn as the perturbation increases. All of these features are present in the Chirikov Standard Map!

C++ Program to Simulate the Nonlinear Pendulum

Program 1: <http://www.physics.buffalo.edu/phy410-505/topic4/pendulum.cpp>

```
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

#include "linalg.hpp" // linear algebra for vector<double>
#include "odeint.hpp" // Fourth order Runge-Kutta routines
using namespace cpl;

const double pi = 4 * atan(1.0);

const double g = 9.8; // acceleration of gravity

double L = 9.8; // length of pendulum
double q = 0.2; // damping coefficient
double Omega_D = 2.0/3.0; // frequency of driving force
double F_D = 1.2; // amplitude of driving force
bool nonlinear; // linear if false

vector<double> f( // extended derivative vector
  const vector<double>& x // extended solution vector
) {
  double t = x[0];
  double theta = x[1];
  double omega = x[2];
  vector<double> f(3);
  f[0] = 1;
  f[1] = omega;
  if (nonlinear)
    f[2] = - (g/L) * sin(theta) - q * omega + F_D * sin(Omega_D * t);
  else
    f[2] = - (g/L) * theta - q * omega + F_D * sin(Omega_D * t);
  return f;
}

int main() {
  cout << " Nonlinear damped driven pendulum\n"
    << " -----\n"
    << " Enter linear or nonlinear: ";
  string response;
  cin >> response;
  nonlinear = (response[0] == 'n');
  cout << " Length of pendulum L: ";
  cin >> L;
```

```

cout<< " Enter damping coefficient q: ";
cin >> q;
cout << " Enter driving frequency Omega_D: ";
cin >> Omega_D;
cout << " Enter driving amplitude F_D: ";
cin >> F_D;
cout << " Enter theta(0) and omega(0): ";
double theta, omega, tMax;
cin >> theta >> omega;
cout << " Enter integration time t_max: ";
cin >> tMax;

double t = 0;
vector<double> x(3);
x[0] = t;
x[1] = theta;
x[2] = omega;

double dt = 0.05;
double accuracy = 1e-6;
RK4 rk4; // RK4 object defined in odeint.hpp
rk4.set_step_size(dt);
rk4.set_accuracy(accuracy);
ofstream dataFile("pendulum.data");

while (t < tMax) {
    rk4.adaptive_step(f, x);
    t = x[0], theta = x[1], omega = x[2];
    if (nonlinear) {
        while (theta >= pi) theta -= 2 * pi;
        while (theta < -pi) theta += 2 * pi;
    }
    dataFile << t << '\t' << theta << '\t' << omega << '\n';
}

cout << " Output data to file pendulum.data" << endl;
dataFile.close();
}

```

C++ OpenGL Pendulum Program

Program 2: <http://www.physics.buffalo.edu/phy410-505/topic4/pendulum-gl.cpp>

```

#include <cmath>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <sstream>
#include <string>

```

```

using namespace std;

#ifdef __APPLE__          // Mac OS X uses different header
# include <GLUT/glut.h>
#else                    // Unix and Windows
# include <GL/glut.h>
#endif

#include "linalg.hpp"     // linear algebra for vector<double>
#include "odeint.hpp"     // ODE integration routines, Runge-Kutta ...
using namespace cpl;

const double pi = 4 * atan(1.0);

const double g = 9.8;    // acceleration of gravity

double L = 9.8;         // length of pendulum
double q = 0.5;         // damping coefficient
double Omega_D = 2.0/3.0; // frequency of driving force
double F_D = 1.2;       // amplitude of driving force
bool nonlinear;         // linear if false

vector<double> f(const vector<double>& x) { // extended derivative vector
    double t = x[0];
    double theta = x[1];
    double omega = x[2];
    vector<double> f(3); // vector with 3 components
    f[0] = 1;
    f[1] = omega;
    if (nonlinear)
        f[2] = - (g/L) * sin(theta) - q * omega + F_D * sin(Omega_D * t);
    else
        f[2] = - (g/L) * theta - q * omega + F_D * sin(Omega_D * t);
    return f;
}

vector<double> x(3);

double dt = 0.05;
double accuracy = 1e-6;
RK4 rk4;

void getInput() {
    cout << " Nonlinear damped driven pendulum\n"
         << " -----\n"
         << " Enter linear or nonlinear: ";
    string response;
    cin >> response;
    nonlinear = (response[0] == 'n');
    cout<< " Length of pendulum L: ";
    cin >> L;
}

```

```

cout<< " Enter damping coefficient q: ";
cin >> q;
cout << " Enter driving frequency Omega_D: ";
cin >> Omega_D;
cout << " Enter driving amplitude F_D: ";
cin >> F_D;
cout << " Enter theta(0) and omega(0): ";
double theta, omega;
cin >> theta >> omega;

x[0] = 0;
x[1] = theta;
x[2] = omega;

rk4.set_step_size(dt);
rk4.set_accuracy(accuracy);
}

void step() {
    rk4.adaptive_step(f, x);
}

double frames_per_second = 30;    // for animation in real time

void animation_step() {
    double start = x[0];
    clock_t start_time = clock();
    step();
    double tau = 1.0 / frames_per_second;
    while (x[0] - start < tau)
        step();
    while ((double(clock()) - start_time)/CLOCKS_PER_SEC < tau)
        ;
    glutPostRedisplay();
}

void drawText(const string& str, double x, double y) {
    glRasterPos2d(x, y);
    int len = str.find('\0');
    for (int i = 0; i < len; i++)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, str[i]);
}

void drawVariables() {
    // write t, theta, omega values
    glColor3ub(0, 0, 255);
    ostringstream os;
    os << "t = " << x[0] << ends;
    drawText(os.str(), -1.1, -1.1);
    os.seekp(0);
    os << "theta = " << x[1] << ends;
}

```

```

    drawText(os.str(), -1.1, 1.1);
    os.seekp(0);
    os << "omega = " << x[2] << ends;
    drawText(os.str(), 0.3, 1.1);
}

void displayPendulum() {
    glClear(GL_COLOR_BUFFER_BIT);

    // draw the pendulum rod
    double xp = sin(x[1]);
    double yp = -cos(x[1]);
    glColor3ub(0, 255, 0);
    glBegin(GL_LINES);
        glVertex2d(0, 0);
        glVertex2d(xp, yp);
    glEnd();

    // draw the pendulum bob
    glPushMatrix();
    glTranslated(sin(x[1]), -cos(x[1]), 0);
    glColor3ub(255, 0, 0);
    const double r = 0.1;
    glPolygonMode(GL_FRONT, GL_FILL);
    glBegin(GL_POLYGON);
        for (int i = 0; i < 12; i++) {
            double theta = 2 * pi * i / 12.0;
            glVertex2d(r * cos(theta), r * sin(theta));
        }
    glEnd();
    glPopMatrix();

    // write t, theta, and omega
    drawVariables();

    // we are using double buffering - write buffer to screen
    glutSwapBuffers();
}

bool clearPhasePlot;

void displayPhasePlot() {
    static double thetaOld, omegaOld;
    if (clearPhasePlot) {
        glClear(GL_COLOR_BUFFER_BIT);
        clearPhasePlot = false;
        thetaOld = 2 * pi, omegaOld = 2 * pi;

        // draw axes
        glColor3ub(0, 255, 0);
        glBegin(GL_LINES);

```

```

        glVertex2d(0, -1); glVertex2d(0, 1);
        glVertex2d(-1, 0); glVertex2d(1, 0);
    glEnd();
}

// draw the phase point
double theta = x[1];
while (theta >= pi) theta -= 2 * pi;
while (theta < -pi) theta += 2 * pi;
double omega = x[2];
glColor3ub(255, 0, 0);
if (abs(theta - thetaOld) < pi && abs(omega) < pi
    && abs(omega - omegaOld) < pi) {
    glBegin(GL_LINES);
        glVertex2d(thetaOld / pi, omegaOld / pi);
        glVertex2d(theta / pi, omega / pi);
    glEnd();
}
thetaOld = theta, omegaOld = omega;
glPopMatrix();

// write t, theta, and omega
glColor3ub(255, 255, 255);
glRectd(-1.2, 1, 1.2, 1.2);
glRectd(-1.2, -1, 1.2, -1.2);
drawVariables();

glutSwapBuffers();
}

void reshape(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    double aspectRatio = w/double(h);
    double d = 1.2;
    if (aspectRatio > 1)
        glOrtho(-d*aspectRatio, d*aspectRatio, -d, d, -1.0, 1.0);
    else
        glOrtho(-d, d, -d/aspectRatio, d/aspectRatio, -1.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
}

bool running;           // is the animation running?
bool phasePlot;        // switch to phase plot if true

void mouse(int button, int state, int x, int y) {
    switch (button) {
        case GLUT_LEFT_BUTTON:
            if (state == GLUT_DOWN) {
                if (running) {

```

```

        glutIdleFunc(NULL);
        running = false;
    } else {
        glutIdleFunc(animation_step);
        running = true;
    }
}
break;
case GLUT_RIGHT_BUTTON:
if (state == GLUT_DOWN) {
    if (phasePlot) {
        glutDisplayFunc(displayPendulum);
        phasePlot = false;
    } else {
        glutDisplayFunc(displayPhasePlot);
        clearPhasePlot = phasePlot = true;
    }
    glutPostRedisplay();
}
break;
}
}

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    getInput();
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow(argv[0]);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glShadeModel(GL_FLAT);
    glutDisplayFunc(displayPendulum);
    glutReshapeFunc(reshape);
    glutMouseFunc(mouse);
    glutIdleFunc(NULL);
    glutMainLoop();
}

```

Homework Problem

Explore the dynamics of the pendulum and describe 5 types of motion that you find most interesting, providing examples of real space and/or phase space plots. Possibilities include under, critical, and overdamping; oscillations and rotations; period doubling (try the range $1.35 < F_D < 1.5$); chaotic motion; strange attractors; and intermittency. The model has several parameters $g, \ell, q, F_D, \Omega_D$. The critical parameter to vary is the driving force amplitude F_D . Try fixing $\ell = 9.8$ so the natural frequency $\Omega = 1$. Many references use a driving frequency $\Omega_D = 2/3$. The damping constant q is not critical except for very small driving frequency: choose $q = 0.5$. Then try varying $F_D = 0.1, 0.5, 0.9, 1.2, 1.35-1.5, \dots$

References

- [1] D.H. Terman and E.M. Izhikevich, “State Space”, Scholarpedia, http://www.scholarpedia.org/article/Phase_space.
- [2] R. DeSerio, “Chaotic pendulum: The complete attractor”, Am. J. Phys. **71**, 250 (2003), <http://dx.doi.org/10.1119/1.1526465>.
- [3] http://en.wikipedia.org/wiki/Kolmogorov-Arnold-Moser_theorem.
- [4] E.G. Gwinn and R.M. Westervelt, “Fractal basin boundaries and intermittency in the driven damped pendulum”, Phys. Rev. A **33**, 4143 (1986), <http://link.aps.org/doi/10.1103/PhysRevA.33.4143>.