

Lorenz Attractor and Hénon Map

The Lorenz Attractor

The weather is a manifestation of fluid flow in Earth's atmosphere. The Navier-Stokes equations of fluid flow are extremely complex, and their solution for weather prediction requires the use of massively parallel supercomputers.

Rayleigh introduced a simplified two-dimensional model of atmospheric fluid flow in a layer of depth H and constant temperature difference ΔT between top and bottom:

$$\frac{\partial}{\partial t} \nabla^2 \psi = -\frac{\partial(\psi, \nabla^2 \psi)}{\partial(x, z)} + \nu \nabla^4 \psi + g\alpha \frac{\partial \theta}{\partial x}, \quad (1)$$

$$\frac{\partial}{\partial t} \theta = -\frac{\partial(\psi, \theta)}{\partial(x, z)} + \kappa \nabla^2 \theta + \frac{\Delta T}{H} \frac{\partial \psi}{\partial x}, \quad (2)$$

where ψ is a stream function which is a type of potential for the fluid velocity, θ is the fluctuation of temperature from the steady state, g is the acceleration of gravity, α the coefficient of thermal expansion, ν the kinematic viscosity, and κ the thermal conductivity.

The atmospheric scientist Lorenz[1] studied a simplified form of these equations

$$\frac{dX}{dt} = -\sigma X + \sigma Y \quad (3)$$

$$\frac{dY}{dt} = -XZ + rX - Y \quad (4)$$

$$\frac{dZ}{dt} = XY - bZ \quad (5)$$

where σ , r and b are constants.

Lorenz discovered that the asymptotic trajectory of this system of equations, for parameter values $\sigma = 10$, $b = 8/3$, and $r = 28$, and initial values $X(0) = 0$, $Y(0) = 1$, and $Z(0) = 0$ is a complicated structure called a *strange attractor*. This structure has the property of *self-similarity*, i.e., it looks the same when parts of it are repeatedly magnified.

He had discovered the famous butterfly effect[2], which implies that the weather is essentially unpredictable.

Poincaré Sections

Solutions of the Lorenz equations are trajectories in 3-dimensional space X, Y, Z . The Lorenz attractor has a complex shape, but it can be visualized in a 3-dimensional plot. Trajectories of ODEs with more than 3 dependent variables cannot be directly visualized. The Kepler problem, for example, has a 4-dimensional phase space in the center of mass system.

Poincaré introduced a simple and powerful technique to study the nature of dynamical trajectories called the Poincaré section. To construct a section of the trajectory, choose a fixed hyperplane in the phase space of the system and record the intersection points of the trajectory with the hyperplane. If the hyperplane is 2-dimensional, the trajectory can easily be visualized as a 2-dimensional plot of points.

The nature of the trajectory can often be inferred from its Poincaré section. If the section points lie on a simple curve, then the trajectory is likely to be periodic. If the section points spread out in two dimensions and tend to fill an area on the hyperplane, then the trajectory is nonperiodic and possibly chaotic.

C++ Program to Solve The Lorenz Equations

The C++ code for solving a system of ODE's can be made even more compact by treating the independent variable as an additional dependent variable with unit slope:

$$\frac{dt}{dt} = 1 \quad (6)$$

$$\frac{dX}{dt} = -\sigma X + \sigma Y \quad (7)$$

$$\frac{dY}{dt} = -XZ + rX - Y \quad (8)$$

$$\frac{dZ}{dt} = XY - bZ \quad (9)$$

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

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

#include "linalg.hpp"
using namespace cpl;

double sigma = 10;           // constants chosen by Lorenz
double b = 8.0 / 3.0;
double r = 28;

vector<double> f(           // The Lorenz equations
  vector<double> txyz      // vector t, x, y, z - input
) {
  double t = txyz[0];
  double x = txyz[1];
  double y = txyz[2];
  double z = txyz[3];
  vector<double> f(4);
  f[0] = 1;
  f[1] = - sigma * x + sigma * y;
  f[2] = - x * z + r * x - y;
  f[3] = x * y - b * z;
  return f;
}

void RK4Step(              // 4th order Runge-Kutta
  vector<double>& y,        // extended solution vector
  double h)                // step size
{
  vector<double> k1 = h * f(y);
  vector<double> k2 = h * f(y + 0.5 * k1);
```

```

    vector<double> k3 = h * f(y + 0.5 * k2);
    vector<double> k4 = h * f(y + k3);
    y += (k1 + 2 * k2 + 2 * k3 + k4) / 6.0;
}

vector<double> txyz(4);          // global variable to hold t,x,y,z

void initialize() {             // initial conditions in textbook
    txyz[0] = 0.0;
    txyz[1] = 0.0;
    txyz[2] = 1.0;
    txyz[3] = 0.0;
}

double dt = 0.001;             // time step for integration

void findNextCrossing() {      // find next Poincare section point

    vector<double> txyzOld = txyz; // save the old point

    while (true) {
        RK4Step(txyz, dt);        // take a step

        // check whether y changes sign, i.e., crosses y = 0
        if (txyz[2] * txyzOld[2] < 0)
            break;                // stop stepping
        else
            txyzOld = txyz;
    }

    // use linear interpolation to find intersection
    double r = txyzOld[2] / txyz[2];
    for (int i = 0; i < 4; i++)
        txyz[i] = (txyzOld[i] + r * txyz[i]) / (r + 1);
}

int main(int argc, char *argv[]) {

    cout << " Trajectory and Poincare Section for the Lorenz Attractor\n"
         << " using 4th order Runge-Kutta with time step dt = " << dt << "\n"
         << " sigma = " << sigma << ", b = " << b << ", r = " << r << endl;

    initialize();
    cout << " initial conditions: x = " << txyz[1] << "\t"
         << ", y = " << txyz[2] << "\t" << ", z = " << txyz[3] << endl;

    // transient trajectory
    double t = 50;
    string fileName = "transient.data";
    cout << " Integrating to time t = " << t << '\n'
         << " trajectory in file " << fileName << endl;
}

```

```

ofstream dataFile(fileName.c_str());
dataFile << txyz[0] << '\t' << txyz[1] << '\t'
    << txyz[2] << '\t' << txyz[3] << '\n';
int step = 0;
int skip = 5;
while (txyz[0] < t) {
    RK4Step(txyz, dt);
    if (++step % skip != 0)        // record every skip steps
        continue;
    dataFile << txyz[0] << '\t' << txyz[1] << '\t'
        << txyz[2] << '\t' << txyz[3] << '\n';
}
dataFile.close();

// Poincare section
fileName = "section.data";
int points = 1000;
cout << " Finding " << points << " Poincare section points at y = 0\n"
    << " section data in file " << fileName << endl;
dataFile.open(fileName.c_str());
for (int point = 0; point < points; point++) {
    findNextCrossing();
    dataFile << txyz[0] << '\t' << txyz[1] << '\t'
        << txyz[2] << '\t' << txyz[3] << '\n';
}
}

```

OpenGL Lorenz Attractor program

The following OpenGL program plots a projection of the Lorenz attractor trajectory in the $x - z$ plane.

The program illustrates the use of a GUI event loop with two callback functions, one to handle redrawing the graphics in a window, and the other to handle changes in window size.

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

```

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

#ifdef __APPLE__
# include <GLUT/glut.h>
#else
# include <GL/glut.h>
#endif

```

```

#include "linalg.hpp"
using namespace cpl;

double sigma = 10;
double b = 8.0 / 3.0;
double r = 28;

vector<double> f(vector<double> txyz) {
    double t = txyz[0];
    double x = txyz[1];
    double y = txyz[2];
    double z = txyz[3];
    vector<double> f(4);
    f[0] = 1;
    f[1] = - sigma * x + sigma * y;
    f[2] = - x * z + r * x - y;
    f[3] = x * y - b * z;
    return f;
}

void RK4Step(vector<double>& y, double h) {
    vector<double> k1 = h * f(y);
    vector<double> k2 = h * f(y + 0.5 * k1);
    vector<double> k3 = h * f(y + 0.5 * k2);
    vector<double> k4 = h * f(y + k3);
    y += (k1 + 2 * k2 + 2 * k3 + k4) / 6.0;
}

vector<double> txyz(4);          // global variable to hold t,x,y,z
bool clear;                     // need to clear the window

void initialize() {
    txyz[0] = 0.0;
    txyz[1] = 0.0;
    txyz[2] = 1.0;
    txyz[3] = 0.0;
    clear = true;
}

double dt = 0.0001;            // time step for integration
int skip = 50;                 // steps to skip while plotting

void step() {
    for (int i = 0; i <= skip; i++)
        RK4Step(txyz, dt);
    glutPostRedisplay();
}

void display() {                // callback function to draw graphics
    if (clear) {
        glClear(GL_COLOR_BUFFER_BIT);
    }
}

```

```

    clear = false;
}
glColor3f(1, 0, 0);
glBegin(GL_POINTS);
    glVertex2d(txyz[1], txyz[3]);
glEnd();
glutSwapBuffers();
glFlush();
}

void reshape(int w, int h) {    // callback to handle window size changes
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-20, 20, 0, 50, -1, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    clear = true;
}

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    initialize();
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(400, 400);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Lorenz Attractor");
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glShadeModel(GL_FLAT);
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutIdleFunc(step);
    glutMainLoop();
}

```

Strange Attractors

The term strange attractor appears to have been coined by Takens and Ruelle around 1971. Here is a quote from Ruelle:

I asked Floris Takens if he had created this remarkably successful expression. Here is his answer: “Did you ever ask God whether he created this damned Universe? ... I don’t remember anything ... I often create without remembering it...” The creation of strange attractors thus seems to be surrounded by clouds and thunder. Anyway, the name is beautiful, and well suited to these astonishing objects, of which we understand so little.

— David Ruelle *The Mathematical Intelligencer* **2**, 126 (1980)

Strange attractors seem to be characterized by the following properties:

- they represent the asymptotic behavior of the system, starting at any point in a continuous "basin" of possible initial conditions, after transients have died out,
- they have fractal dimension,
- the dynamics on the attractor is sensitively dependent on initial conditions, and
- the attractor is invariant in the sense that the system comes arbitrarily close to any point on the attractor independently of the initial conditions.

Visit Sprott's Fractal Gallery[3] for some cool pictures of strange attractors!

The Hénon Map and Attractor

In an effort to understand the Lorenz attractor, the astronomer Hénon[4], discovered a simple map which has similar properties. This is a map in two dimensions that is similar to the Logistic Map:

$$x_{n+1} = y_n + 1 - ax_n^2 \quad (10)$$

$$y_{n+1} = bx_n \quad (11)$$

where a and b are constants. He discovered a strange attractor at parameter values $a = 1.4$ and $b = 0.3$. This strange attractor appears to be a *fractal*, a geometrical object with dimension between 1 and 2, more than a line but less than an area. It provides a very nice demonstration of self-similarity under magnification. Measurements on the fractal give a dimension of 1.21.

OpenGL Hénon Map program

_____ Program 3: <http://www.physics.buffalo.edu/phy410-505/topic4/henon.cpp> _____

```
#include <cstdlib>
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

#ifdef __APPLE__
# include <GLUT/glut.h>
#else
# include <GL/glut.h>
#endif

double a = 1.4;
double b = 0.3;
double x_0 = 0;
double y_0 = 0;

double x, y;          // current map point
```

```

double xmin, xmax, ymin, ymax; // rectangle for graphics

void map() {
    double xOld = x;
    x = y + 1 - a * x * x;
    y = b * xOld;
    if (x > xmin && x < xmax && y > ymin && y < ymax)
        glutPostRedisplay();
}

double xmin0 = -1.5;
double xmax0 = +1.5;
double ymin0 = -0.5;
double ymax0 = +0.5;
bool clear;

void scaleWindow() {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(xmin, xmax, ymin, ymax, -1, 1);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    clear = true;
}

double xMagnification, yMagnification;

void initialize() {
    x = x_0;
    y = y_0;
    xmin = xmin0;
    xmax = xmax0;
    ymin = ymin0;
    ymax = ymax0;
    xMagnification = yMagnification = 1;
    scaleWindow();
}

void reshape(int w, int h) {
    glViewport(0, 0, w, h);
    scaleWindow();
}

double xFromPixel(int x) {
    int w = glutGet(GLUT_WINDOW_WIDTH);
    return xmin + (xmax - xmin) * x / double(w);
}

double yFromPixel(int y) {
    int h = glutGet(GLUT_WINDOW_HEIGHT);
    return ymin + (ymax - ymin) * (h - y) / double(h);
}

```

```

}

int xDrag, yDrag;
bool dragging;

void motion(int x, int y) {
    if (dragging) {
        xDrag = x;
        yDrag = y;
        glutPostRedisplay();
    }
}

int xDown, yDown;

void mouse(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON) {
        if (state == GLUT_DOWN) {
            xDown = x;
            yDown = y;
            dragging = true;
            glutIdleFunc(NULL);
        } else if (state == GLUT_UP) {
            dragging = false;
            if (abs(xDrag - xDown) > 5 && abs(yDrag - yDown) > 5) {
                xMagnification *= (xMax - xMin);
                yMagnification *= (yMax - yMin);
                xMin = xDrag < xDown ? xFromPixel(xDrag) : xFromPixel(xDown);
                xMax = xDrag > xDown ? xFromPixel(xDrag) : xFromPixel(xDown);
                yMin = yDrag > yDown ? yFromPixel(yDrag) : yFromPixel(yDown);
                yMax = yDrag < yDown ? yFromPixel(yDrag) : yFromPixel(yDown);
                xMagnification /= (xMax - xMin);
                yMagnification /= (yMax - yMin);
                scaleWindow();
            }
            glutIdleFunc(map);
        }
    } else if (button == GLUT_RIGHT_BUTTON) {
        initialize();
    }
}

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 display() {
    static bool erase;          // erase rubberband rectangle

```

```

if (clear) {
    glClear(GL_COLOR_BUFFER_BIT);
    erase = clear = false;
    glColor3f(0, 0, 1);
    ostringstream os;
    os << "x magnification = " << xMagnification << " "
        << "y magnification = " << yMagnification << ends;
    drawText(os.str(), xMin + 0.05 * (xMax - xMin),
            yMax - 0.05 * (yMax - yMin));
}
glColor3f(1, 0, 0);
glBegin(GL_POINTS);
    glVertex2d(x, y);
glEnd();

// draw rubberband rectangle while the mouse is being dragged
static double x1, y1, x2, y2; // rectangle coordinates
glColor3f(1, 1, 1);
if (dragging) {
    glEnable(GL_COLOR_LOGIC_OP);
    glLogicOp(GL_XOR); // use XOR mode
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    if (erase) {
        glRectd(x1, y1, x2, y2);
        erase = false;
    } else {
        x1 = xFromPixel(xDown);
        y1 = yFromPixel(yDown);
        x2 = xFromPixel(xDrag);
        y2 = yFromPixel(yDrag);
        glRectd(x1, y1, x2, y2);
        erase = true;
    }
    glDisable(GL_COLOR_LOGIC_OP);
}

glutSwapBuffers();
glFlush();
}

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowSize(400, 400);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Henon Attractor");
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glShadeModel(GL_FLAT);
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMouseFunc(mouse);
}

```

```
glutMotionFunc(motion);
glutIdleFunc(map);
initialize();
glutMainLoop();
}
```

Homework Problem

Generate trajectories and Poincaré sections for different values of the parameter r in the Lorenz equations. For example, it is known that the transition from steady state convection to chaotic behavior occurs at $r = 470/19 \approx 24.74$. Choose values just below and above this transition point and comment on the nature of the trajectories. Another interesting region occurs around $r = 160$. Below this value, the trajectories are periodic and exhibit period doubling behavior similar to the logistic map. Above $r = 160$ some trajectories exhibit intermittency, which is characterized by periodic behavior over many periods that is suddenly interrupted by bursts of activity.

References

- [1] E.N. Lorenz, “Deterministic Nonperiodic Flow”, J. Atmos. Sci. **20**, 130 (1963),
<http://ams.allenpress.com/perlserv/?request=get-abstract&doi=10.1175%2F1520-0469%281963%29020%3C0130%3ADNF%3E2.0.CO%3B2>
- [2] Michael Cross, Chaos Demonstrations, The Butterfly Effect:
<http://www.cmp.caltech.edu/%7Emcc/chaos%5Fnew/Lorenz.html>.
- [3] Professor Julian Sprott’s Fractal Gallery <http://sprott.physics.wisc.edu/fractals.htm>.
- [4] M. Hénon, “A two-dimensional mapping with strange attractor”, Comm. Math. Phys. **50**, 69 (1976),
<http://projecteuclid.org/euclid.cmp/1103900150>.