Classical Chaos

James O. Thomas

SMU Physics

May 4, 2017

Outline

Periodic Motion

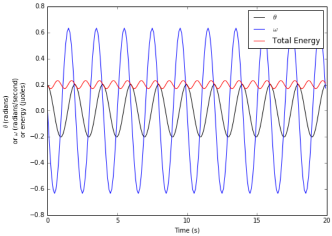
- In Chapter 3 and 6 we discussed periodic motion that results in closed form solutions.
- We could also use perturbation theory (Chapter 12) if the potential can be broken into a main integrable part and a "weak" non-integrable part.
- If the interaction term is not small, the resulting coupled equations can be quite complex.
- The solutions to the resulting equations are often highly sensitive to the initial conditions.
- Classically, the motion is still completely deterministic.

Euler-Cromer Method

$$\ddot{\theta} + \frac{g}{l}\theta = 0$$

- Convert the second order DE into two first order, coupled finite difference equations.
- Use the updated ω when calculating the new θ.
- Step across the time domain.

```
for i in range(nsteps-1):
    omega[i+1] = omega[i] - (g/length)*theta[i]*dt
    theta[i+1] = theta[i] + omega[i+1]*dt
```



Runge-Kutta

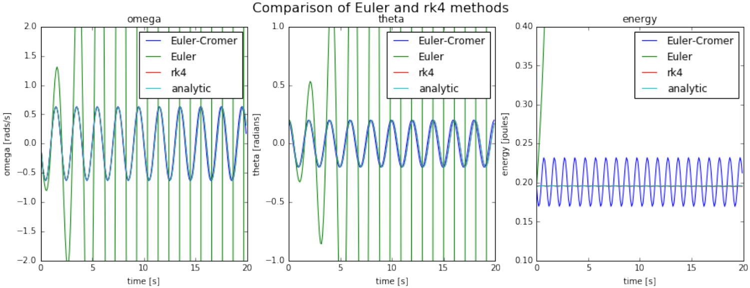
The Runge-Kutta method is samples the slopes of a function at the endpoints as well as interior points of a given interval.
The 4th order Runge-Kutta Method is given by:

$$y_{n+1} = y_n + \left(\frac{1}{6} k_1 + \frac{1}{3} k_2 + \frac{1}{3} k_3 + \frac{1}{6} k_4 \right) h + 0(h^5)$$

$$\begin{aligned} k_1 &= f(x_n, y_n) \\ k_2 &= f(x_n + \frac{1}{2} h, y_n + \frac{1}{2} k_1) \\ k_3 &= f(x_n + h, y_n + k_2) \\ k_4 &= f(x_n + h, y_n + k_3) \end{aligned}$$

```python
def Jrk4(f, h):

    k1 = RHS([f[0], f[1]])                              # use initial point
    k2 = RHS([f[0] + 0.5*k1[0]*h, f[1] + 0.5*k1[1]*h])  # one of the midpoints
    k3 = RHS([f[0] + 0.5*k2[0]*h, f[1] + 0.5*k2[1]*h])  # the other midpoint
    k4 = RHS([f[0] + k3[0]*h, f[1] + k3[1]*h])          # the endpoint
    f_next = [f[0] + (k1[0] + 2*k2[0] + 2*k3[0] + k4[0])*h/6, f[1] + (k1[1] + 2*k2[1] + 2*k3[1] + k4[1])*h/6]
    return f_next
```

Runge-Kutta and Euler-Cromer



Driven Damped Harmonic Oscillators

$$m\ddot{x} = -kx - b\dot{x} + F_0 \cos(\omega t)$$

$$\ddot{x} + 2\gamma\dot{x} + \omega_0^2 x = A\cos\omega t$$

$$x_c(t) = e^{-\gamma t}\left(C_1 \cos\eta t + C_2 \sin\eta t\right)$$

where $\eta = \sqrt{\omega^2 - \gamma^2}$

$$x_p(t) = G(\omega) A \cos(\omega t - \phi)$$

where $G(\omega) = \frac{1}{\sqrt{(\omega_0^2 - \omega^2)^2 + 4\gamma^2\omega^2}}$

$$x(t) = e^{-\gamma t}\left(C_1 \cos\eta t + C_2 \sin\eta t\right) + G(\omega) A \cos(\omega t - \phi)$$

Numeric Driven Damped Harmonic Oscillators

Basic idea: convert the single second order DE into two coupled first order DEs, then solve the first order DEs using one of the numerical techniques (4th order Runge-Kutta).

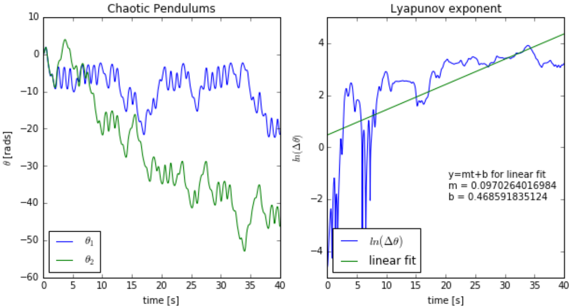
$$\ddot{\theta} = -\frac{g}{l}\sin\theta - q\dot{\theta} + F_D \sin(\Omega_D t)$$

$$\frac{d\omega}{dt} = -\frac{g}{l}\sin\theta + q\omega + F_D \sin(\Omega_D t)$$

$$\frac{d\theta}{dt} = \omega$$

Lyapunov Exponents and Chaos

- Consider two driven-damped harmonic oscillators with only slightly different conditions (maybe slightly different damping factors).
- Integrate both oscillators numerically.
- Calculate the difference in angles.
- The equation $\Delta\theta(t) = \Delta\theta_0 e^{\lambda t}$.
- λ is the Lyapunov exponent. It quantifies the average growth of an initially infinitesimal difference in angle.



Lyapunov Exponents Code

```
# set up the integrators
r1 = ode(f1).set_integrator('dopri5')
r1.set_initial_value([theta_init1, omega_init1], t_init)
r2 = ode(f2).set_integrator('dopri5')
r2.set_initial_value([theta_init2, omega_init2], t_init)

# integrate the 1st pendulum
k = 1
while r1.successful() and k < nsteps:
    r1.integrate(r1.t + dt)
    t1[k] = r1.t                    # should be the same values as the second integrator
    theta1[k] = r1.y[0]
    omega1[k] = r1.y[1]
    k = k + 1

# integrate the 2nd pendulum
k = 1
while r2.successful() and k < nsteps:
    r2.integrate(r2.t + dt)
    t2[k] = r2.t                    # t values should be the same, but just in case
    theta2[k] = r2.y[0]
    omega2[k] = r2.y[1]
    k = k + 1

# calculate the stuff for the Lyapunov exponent
lndtheta = np.log(np.abs(theta1 - theta2)+0.01)
m, b = np.polyfit(t1,lndtheta,1)
# print(m,b)
```

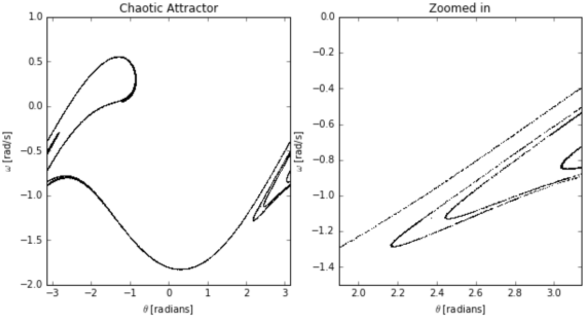
Poincare Maps

- A non-chaotic oscillator will trace out ellipses in phase space (boring).
- A chaotic oscillator will, in general, completely cover the available volume of phase space.
- Rather than study the whole volume (or hypervolume), we can look at a "slice" of this volume.
- We simply calculate the positions of points where the oscillator passes through this surface.

```
while r.successful() and k < nsteps:
    r.integrate(r.t + dt)
    t_val = r.t
    theta_val = r.y[0]
    omega_val = r.y[1]
    # only keep the points that are (almost) in phase with the driving force
    if np.abs(t_val - 2*n*np.pi/omega_drive) < dt/2:   # extra factor of 2 from Giordano book
        omega.append(omega_val)
        theta.append(theta_val%twopi)
        n = n + 1
    k = k + 1
```

Poincare Maps

Consider a chaotic oscillator, but only plot θ vs ω for points in phase with the driving force ($\Omega_D = 2\pi n$). For a non-chaotic oscillator, this would be a single point.



Henon-Heiles Hamiltonian

Consider the Hamiltonian

$$H = \frac{p_x^2}{2m} + \frac{p_y^2}{2m} + \frac{1}{2}k\left(x^2 + y^2\right) + \lambda\left(x^2 y - \frac{1}{3}y^3\right)$$

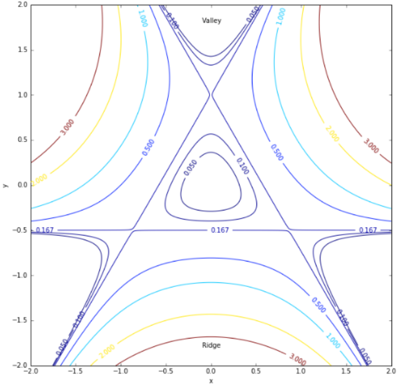
If we consider non-relativistic momentum $p = mv$, and nondimensionalize the Hamiltonian, $k = 1$, $\lambda = 1$, we can get

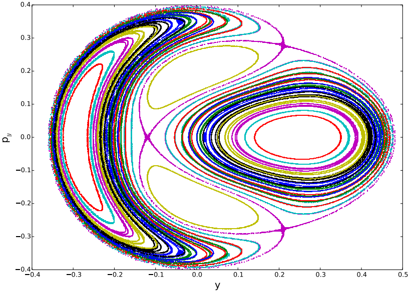
$$E = \frac{1}{2}\dot{x}^2 + \frac{1}{2}\dot{y}^2 + \frac{1}{2}x^2 + \frac{1}{2}y^2 + x^2 y - \frac{1}{3}y^3$$

which leads to the coupled, non-linear equations of motions

$$\ddot{x} = -x - 2xy$$
$$\ddot{y} = -y - x^2 + y^2$$

Henon-Heiles Hamiltonian





Logistics Equation

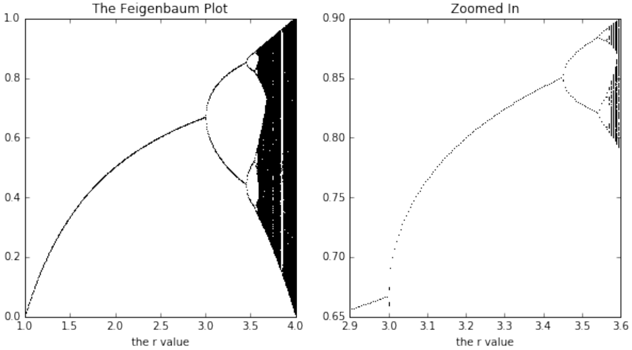
Instead of doing the detailed analysis of a driven damped oscillator, consider the Logistic Equation

$$x_{n+1} = rx_n(1 - x_n)$$

```
def logistic_map(r,x):
    x = r*x*(1-x)
    return x

r = np.arange(1,4+0.01,0.01) # the r values to use
x = np.empty(len(r))        # empty array for x values
x.fill(0.5)                  # fill the x with the initial value

steps = 1000                  # the number of times to run the logistic map

plt.figure()

for i in range(steps):
    x = logistic_map(r,x)       # let it settle down
for i in range(steps):
    x = logistic_map(r,x)       # plot the behavior
    plt.plot(r,x,"k,")
```

Logistics Equation

It exhibits many of the same features of chaotic systems.



- for $\lesssim 3$, there is only a single point.
- for $3 \lesssim r \lesssim 3.4$ there are 2 points (bifurcation).
- exhibits a fractal pattern until $r \approx 3.56$, then the system becomes chaotic.

Mandlebrot Set

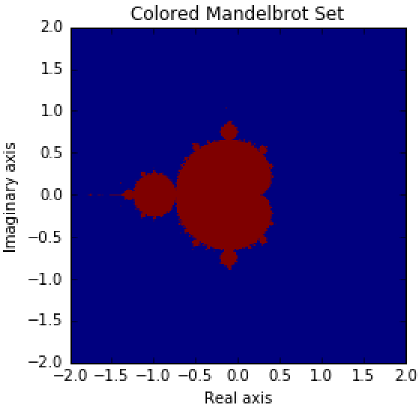
As another example. Consider a complex number C:

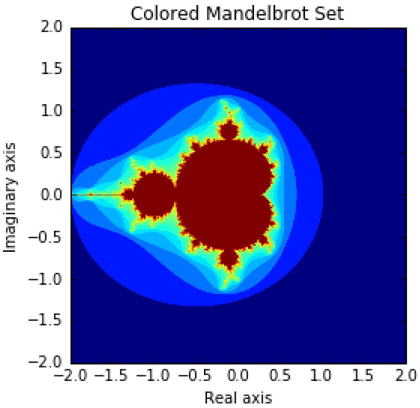
$$z_{n+1} = z_n^2 + C$$

where $z^1 = zz \neq |z|^2$.
If after m iterations, $|z| < 2$ then the point C is in the Mandelbrot set.

```
def Mandelbrot(c):
    z = 0 + 0j
    for i in range(100):
        z = z**2 + c
        if abs(z) > 2:
            break
    if abs(z) >= 2:
        return False
    else:
        return True
```

Mandelbrot Set





References

- **Classical Mechanics** by Goldstein.
- **Computational Physics** by Giordano (Fortran).
- **Computational Physics** by Newman (Python).
- *Classical Mechanics Project* by James O. Thomas, Nathan Brady, and Robert McNamara.
- *Classical Chaos.ipynb* iPython Notebook with the world's okayist python code.