Euler and Runge-Kutta Method

Mojtaba Alaei

October 4, 2013

Content of the course

Radioactive Decays

- Physics problem

$$\frac{dN}{dt} = -\frac{N}{\tau} \quad (1)$$

- analytical solution

$$N(t) = N(0)e^{-t/\tau} \quad (2)$$

Numerical solution, Euler method

- differential of f(x)

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} \equiv \lim_{\Delta t \to 0} \frac{x(t+\Delta t) - x(t)}{\Delta t} \Longrightarrow \frac{x(t+\Delta t) - x(t)}{\Delta t} , \tag{3}$$

- Euler method

$$x(t+\Delta t) \approx x(t) + \frac{\mathrm{d}x(t)}{\mathrm{d}t}\Delta t . \tag{4}$$

- Numerical solution of $\frac{\mathrm{d}N}{\mathrm{d}t} = -\frac{N}{\tau}$:

$$N(t+\Delta t) \approx N(t) - \frac{N(t)}{\tau}\Delta t . \tag{5}$$

Error in Euler method

- Taylor expansion:

$$x(t+\Delta t) = x(t) + \frac{\mathrm{d}x(t)}{\mathrm{d}x}\Delta t + \frac{\mathrm{d}^2x(t)}{\mathrm{d}x^2}\frac{(\Delta t)^2}{2} + \cdots + \frac{\mathrm{d}^n x(t)}{\mathrm{d}x^n}\frac{(\Delta t)^n}{n!} + \ldots \tag{6}$$

- Error per step $\propto (\Delta t)^2$. But in the N steps ($\propto (t_{out} - t_{in})/\Delta t$), the total error $\propto \Delta t$

exercise 1

Write a program to solve $\frac{dN}{dt} = -\frac{N(t)}{\tau}$

- Structure of the program:
  - initialize t_0, t_{end}, N_0, τ ... (read from input)
  - Solve ($\frac{dN}{dt} = -\frac{N(t)}{\tau}$)
  - time steps: $t_{i+1} = t_i + \Delta t \quad \mathrm{if\ t > t_{end} : stop}$
  - In each step: $N(t_{i+1}) = N_{i+1} = N_i - N_i/\tau\Delta t$
- plot Numerical Solution for different time step (Δt) and compare with exact solution.
- plot the error for different steps

Runge-Kutta methods

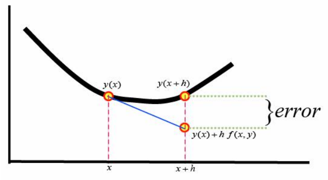

Figure: Euler method

How to improve the Euler method:

- use more terms in Taylor expansion
- use derivative at $x + \Delta t/2$

second-order Runge-Kutta

- $$x(t+\Delta t) = x(t+\frac{\Delta t}{2})+\frac{1}{2}\Delta t(\frac{dx}{dt})_{t+\frac{\Delta t}{2}}+\frac{1}{8}(\frac{d^2x}{dt^2})_{t+\Delta t}+O(\Delta t^3)$$

- $$x(t) = x(t+\frac{\Delta t}{2}) - \frac{1}{2}\Delta t(\frac{dx}{dt})_{t+\frac{\Delta}{t}} + \frac{1}{8}(\frac{d^2x}{dt^2})_{t+\frac{\Delta t}{2}} + O(\Delta t^3)$$

- Subtracting the second expression from the first and rearranging:

$$\begin{aligned} x(t+\Delta t) &= x(t) + \Delta t(\frac{dx}{dt})_{t+\frac{\Delta t}{2}} + O(\Delta t^3) \qquad (7) \\ &= x(t) + \Delta t\, f(x(t+\frac{\Delta t}{2}), t+\frac{1}{2}) + O(\Delta t^3) \end{aligned}$$

(where $\frac{dx}{dt} = f(x,t)$)

Second-order Runge-Kutta, practical algorithm

- We do not have $x(t + \frac{\Delta}{2})$
- Solution: Using Euler method for it:

$$x(t + \frac{\Delta t}{2}) = x(t) + \frac{1}{2}\Delta t\, f(x, t) \tag{8}$$

- practical algorithm
  - $k_1 = \Delta t\, f(x, t)$,
  - $k_2 = \Delta t\, f(x + \frac{1}{2}k_1, t + \frac{1}{2}\Delta t)$,
  - $x(t + \Delta t) = x(t) + k_2$

Error in Second-order Runge-Kutta

Error $\propto O(\Delta t^3)$, Why?

An example

```
from math import sin
from numpy import arange
from pylab import plot, xlabel, ylabel, show
# dx/dt=-x^3+sin(t)
def f(x,t):
    return -x**3 + sin(t)

a= 0.0
b= 10.0
N=200
dt= (b-a)/N

tpoints = arange(a,b,dt)
xpoints = []

x = 0.0
for t in tpoints:
    xpoints.append(x)
    k1 = dt* f(x,t)
    k2 = dt* f(x+0.5*k1, t+0.5*dt)
    x += k2

plot(tpoints, xpoints)
xlabel("t")
ylabel("x(t)")
show()
```

The fourth-order runge-kutta method

If we use more higher term $(\Delta t^3, \Delta t^4)$ in Taylor expansions, we can derive the fourth-order runge-kutta method:

$$\begin{aligned}
k_1 &= \Delta t \, f(x, t) && (9)\\
k_2 &= \Delta t \, f(x + \frac{1}{2}k_1, t + \frac{1}{2}\Delta t) && (10)\\
k_3 &= \Delta t \, f(x + \frac{1}{2}k_2, t + \frac{1}{2}\Delta t) && (11)\\
k_4 &= \Delta t \, f(x + k_3, t + \Delta t) && (12)\\
x(t + \Delta t) &= x(t) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) && (13)
\end{aligned}$$

An example

```
from math import sin
from numpy import arange
from pylab import plot,xlabel,ylabel,show

def f(x,t):
    return -x**3 + sin(t)

a = 0.0
b = 10.0
N = 100
dt = (b-a)/N

tpoints = arange(a,b,dt)
xpoints = []
x = 0.0

for t in tpoints:
    xpoints.append(x)
    k1 = dt*f(x,t)
    k2 = dt*f(x+0.5*k1, t+0.5*dt)
    k3 = dt*f(x+0.5*k2, t+0.5*dt)
    k4 = dt*f(x+k3, t+dt)
    x += (k1+2*k2+2*k3+k4)/6

plot(tpoints,xpoints)
xlabel("t")
ylabel("x(t)")
show()
```

Trajectory with air resistance

For a spherical cannonball, the drag force is:

$$F = \frac{1}{2}\pi R^2 \rho C v^2 \tag{14}$$

Where R is the sphere's radius, ρ is the density of air, v is the velocity, amd C is the so-called cofficient of drag.

The equations of motion for the position (x,y) of the cannonball are:

$$a_x = \dot{v}_x = -\frac{\pi R^2 \rho C}{2m} v_x \sqrt{v_x^2 + v_y^2} \tag{15}$$

$$a_y = \dot{v}_y = -g - \frac{\pi R^2 \rho C}{2m} v_y \sqrt{v_x^2 + v_y^2} \tag{16}$$

Solution with Euler method

$B_{drag} = \frac{1}{2}\pi R^2 \rho C v^2$

$$\begin{aligned}
\dot{x} &= \frac{\mathrm{d}x}{\mathrm{d}t} = v_x \\
\dot{y} &= \frac{\mathrm{d}y}{\mathrm{d}t} = v_y \\
\dot{v_x} &= \frac{\mathrm{d}v_x}{\mathrm{d}t} = -\frac{B_{\mathrm{drag}} v v_x}{m} \\
\dot{v_y} &= \frac{\mathrm{d}v_y}{\mathrm{d}t} = -g - \frac{B_{\mathrm{drag}} v v_y}{m}
\end{aligned}$$

The solution above equations in Euler method is as follows:

$$\begin{aligned}
x_{i+1} &= x_i + v_{x,i}\Delta t \\
y_{i+1} &= y_i + v_{y,i}\Delta t \\
v_{x,i+1} &= v_{x,i} - \frac{B_{\mathrm{drag}} v_i v_{x,i}}{m}\Delta t \\
v_{y,i+1} &= v_{y,i} - g\Delta t - \frac{B_{\mathrm{drag}} v_i v_{y,i}}{m}\Delta t\,,
\end{aligned}$$

Where

$$v_i = \sqrt{v_{x,i}^2 + v_{y,i}^2}\,.$$

Exercise 2

Suppose $R = 8cm$, $m = 1Kg$, $\rho = 1.22kgkhm^{-3}$, $C = 0.47$

- plot (x,y) for different θ and v_0 with Euler method
- plot (x,y) for different θ and v_0 with the second and fourth-order runge-kutta methods
- plot (x,y) for a given θ and v_0 with the Euler, the second and fourth-order runge-kutta methods in a same graph
- Compare the total distance traveled by the cannonball when $C = 0.0$ for different methods with the exact value $(2v_0^2 sin(\theta)cos(\theta)/g\ )$

hint

```
def rk4(x, v, a, dt):
    """Returns final (position, velocity) tuple after
    time dt has passed.

    x: initial position (number-like object)
    v: initial velocity (number-like object)
    a: acceleration function a(x,v,dt) (must be callable)
    dt: timestep (number)"""
    x1 = x
    v1 = v
    a1 = a(x1, v1, 0)

    x2 = x + 0.5*v1*dt
    v2 = v + 0.5*a1*dt
    a2 = a(x2, v2, dt/2.0)

    x3 = x + 0.5*v2*dt
    v3 = v + 0.5*a2*dt
    a3 = a(x3, v3, dt/2.0)

    x4 = x + v3*dt
    v4 = v + a3*dt
    a4 = a(x4, v4, dt)

    xf = x + (dt/6.0)*(v1 + 2*v2 + 2*v3 + v4)
    vf = v + (dt/6.0)*(a1 + 2*a2 + 2*a3 + a4)

    return xf, vf
```