Lecture 3

The Motion of Falling Objects

Hai-Qing Lin

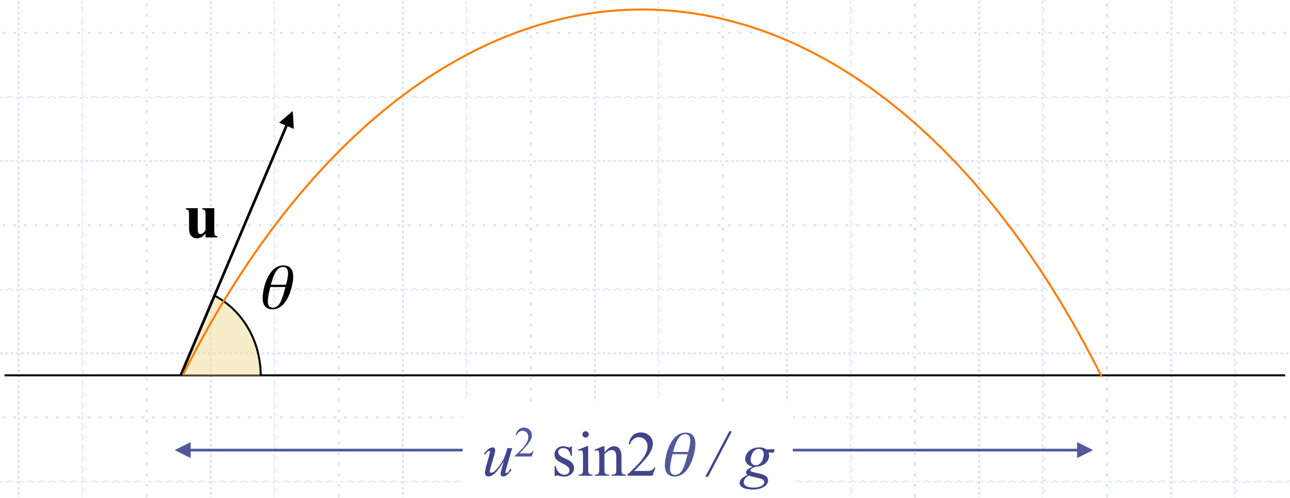
Beijing Computational Science Research Center

This PowerPoint Notes Is Based on the Textbook '***An Introduction to Computer Simulation Methods : Applications to Physical Systems***', 2nd Edition, Harvey Gould and Jan Tobochnik, Addison-Wesley(1996); "A First Course in Computational Physics"; "Numerical Recipes"; "Elementary Numerical Analysis"; "Computational Methods in Physics and Engineering".

Required for Lecture 3

- **Solve 2nd order differential equation as two coupled first order differential equations.**
- **The Euler method and its modifications, e.g., the Euler-Richardson algorithm.**
- **Program for Newton's Law of motion.**
- Projectile motion w/wo drag force, etc.
- Error analysis of ER algorithm.

Projectile Motion



1. The familiar solution: constant acceleration, etc.
2. Slightly modification may cause analytical difficulties
3. Complication in realities

An Easy Question:

Is it more dangerous for a cat fall from the 20th floor than the 3rd floor?

The answer requires knowledge of mechanics, relativity, and common sense!

And how smart the cat is.

Question and Objective

- Newton's mechanics is probably the earliest quantitative science, resulted in elementary calculus. How do we simulate it?
- How to describe accurately the motion of an object falling near the earth's surface?
- We will develop finite difference methods for obtaining numerical solutions to Newton's equations of motion.

Background

- Variables: *Displacement* : $y(t)$,

  Velocity : $v(t) = \frac{dy(t)}{dt}$,

  Acceleration : $a(t) = \frac{dv(t)}{dt} = \frac{d^2 y(t)}{dt^2}$.

- Newton's 2nd law of motion is
  a *2nd-order differential equation*: $a(t) = \frac{F(y,v,t)}{m}$

$$\frac{d^2 y(t)}{dt^2} = \frac{F(y,v,t)}{m}.$$

Numeric Approach

A second-order differential equation $\rightarrow$ **2** coupled first-order differential equations.

$$\frac{d^2 y(t)}{dt^2} = \frac{F(y,v,t)}{m} \quad \rightarrow \quad \frac{dv}{dt} = \frac{F(y,v,t)}{m}, \qquad \frac{dy}{dt} = v.$$

*We can solve a **2nd** order differential equation as **2** coupled **1st** order differential equations.*

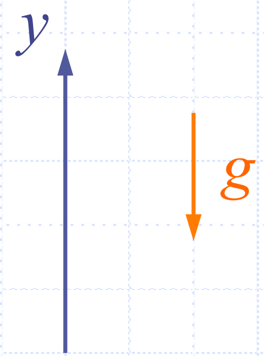
*We can solve a **nth** order differential equation as **n** coupled **1st** order differential equations.*

The Force on a Falling Object

- "Free fall": $F_G / m = g \approx 9.8$ m/s^2.

$$v(t) = v_0 - gt,$$

$$y(t) = y_0 + v_0 t - \frac{1}{2} g t^2.$$



- In general, force is a complicated function of coordinates and its derivatives, e.g.,

  Drag force : (Empirical)

$$F_d(v) = k_1 v, \qquad F_d(v) = k_2 v^2,$$

  k_1 and k_2 depend on the properties of the medium and the shape of the object.

The Force on a Falling Object

- Terminal speed:

$$F_G = F_d \Rightarrow a(t) = 0.$$

$$F_1 = -F_G + F_d \qquad F_2 = -F_G + F_d$$

$$v_t = \frac{mg}{k_1} \qquad v_t = \sqrt{\frac{mg}{k_2}}$$

$$F_1 = -mg\left(1 - \frac{v}{v_t}\right) \qquad F_2 = -mg\left(1 - \frac{v^2}{v_t^2}\right).$$

The Euler Method for Newton's Laws of Motion

Two coupled **differential** equations:

difference

$$\frac{dv}{dt} = a, \qquad \longrightarrow \qquad \frac{\Delta v}{\Delta t} = a,$$

$$\frac{dy}{dt} = v. \qquad \qquad \frac{\Delta y}{\Delta t} = v.$$

Euler Method

$$t_n = t_0 + n\Delta t = t_{n-1} + \Delta t$$
$$v_{n+1} = v_n + a_n \Delta t$$
$$y_{n+1} = y_n + \boldsymbol{v_n} \Delta t$$

Renew *y* by **old *v***

Cromer Method

$$t_n = t_0 + n\Delta t = t_{n-1} + \Delta t$$
$$v_{n+1} = v_n + a_n \Delta t$$
$$y_{n+1} = y_n + \boldsymbol{v_{n+1}} \Delta t$$

Renew *y* by **new *v***

A Program for 1D Motion

```
program free_fall                      ! no air resistance
use common
   integer :: nshow,counter
   call initial()                      ! initial conditions and parameters
   call print_table(nshow)             ! print initial conditions
   counter = 0
   iterate: do
      if (y <= 0) then
         exit iterate
      end if
      call euler()
      counter = counter + 1
      if (modulo(counter,nshow) == 0) then
         call print_table(nshow)
      end if
   end do iterate
   call print_table(nshow)   ! print values at surface
end program free_fall
```

```
module common

public :: initial,euler,print_table

real (selected_real_kind(15,307)), public :: y,v,a,t,dt
real (selected_real_kind(15,307)), public, parameter :: g = 9.8

contains

!all subroutines here

end module common
```

Again, you can read in

```
subroutine initial()
   t = 0              ! initial time (sec)
   y = 10             ! initial height (m)
   v = 0              ! initial velocity
   a = -g
   print *, "time step dt ="
   read *, dt
end subroutine initial

subroutine print_table(nshow)
   integer, intent (in out) :: nshow
   if (t == 0) then
      print *, "number of time steps between output = "
      read *, nshow
      print "(t8,a,t24,a,t34,a,t48,a)", "time","y","velocity","accel"
      print *, ""
   end if
   print "(4f13.4)", t,y,v,a
end subroutine print_table
```

```
subroutine euler()

! following included to remind us that acceleration is constant
   a = -g             ! y positive upward

   y = y + v*dt       ! use velocity at beginning of interval
   v = v + a*dt       ! exchange these two lines leads to ?

    t = t + dt
end subroutine euler

subroutine show_particle()
   …
   see particle motion
end subroutine show_particle
```

Euler-Richardson Algorithm: (Better)

$$t_n = t_0 + n\Delta t = t_{n-1} + \Delta t$$

$$a_n = F(y_n, v_n, t_n)/m, \qquad a_{\mathrm{mid}} = F(y_{\mathrm{mid}}, v_{\mathrm{mid}}, t_n + \tfrac{1}{2}\Delta t)/m,$$

$$v_{\mathrm{mid}} = v_n + a_n \Delta t/2, \qquad v_{n+1} = v_n + a_{\mathrm{mid}} \Delta t,$$

$$y_{\mathrm{mid}} = y_n + v_n \Delta t/2. \qquad y_{n+1} = y_n + v_{\mathrm{mid}} \Delta t.$$

Use Euler Method to find $\boldsymbol{y}_{\mathbf{mid}}$, $\boldsymbol{v}_{\mathbf{mid}}$ by $\boldsymbol{y}_{\boldsymbol{n}}$, $\boldsymbol{v}_{\boldsymbol{n}}$, $\boldsymbol{t}_{\boldsymbol{n}}$.

Use Euler Method to find $\boldsymbol{y}_{\boldsymbol{n}+\mathbf{1}}$, $\boldsymbol{v}_{\boldsymbol{n}+\mathbf{1}}$ by $\boldsymbol{y}_{\mathbf{mid}}$, $\boldsymbol{v}_{\mathbf{mid}}$, $\boldsymbol{t}_{\mathbf{mid}}$.

Drag Force Included

```
program drag
! assume drag force proportional to v^2
use common
real (selected_real_kind(15,307)) :: y0
integer :: nshow,counter

call initial(y0,t)
call print_table(y0,nshow)
iterate: do
   if (t >= 0) then
      exit iterate
   end if
   call Euler_Richardson()
end do iterate
call print_table(y0,nshow)
! values at t = 0

counter = 0
loop: do
   if (t > 0.8) then
      exit loop
   end if
   call Euler_Richardson()
   counter = counter + 1
   if (modulo(counter,nshow) == 0) then
      call print_table(y0,nshow)
   end if
end do loop
end program drag
```

```
module common

public :: initial,print_table,Euler_Richardson

real (selected_real_kind(15,307)), public :: y,v,a,t,vt2,dt,dt_2
real (selected_real_kind(15,307)), public, parameter :: g = 9.8

contains

!all subroutines here

end module common
```

```
subroutine initial(y0,t0)
   real (selected_real_kind(15,307)), intent (in out) :: y0,t0
   real (selected_real_kind(15,307)) :: vt
   t0 = -0.132                ! initial time (see Table 3.1) note
   y0 = 0
   y = y0
   v = 0                      ! initial velocity
   print *, "terminal velocity = "
   read *, vt
   vt2 = vt*vt
   dt = 0.001
   dt_2 = 0.5*dt
end subroutine initial
```

```
subroutine print_table(y0,nshow)
   real (selected_real_kind(15,307)), intent (in out) :: y0
   integer, intent (out) :: nshow
   real (selected_real_kind(15,307)) :: show_time,distance_fallen
   if (t < 0) then
      show_time = 0.1         ! time interval between output
                              ! choice of dt = 0.001 convenient
      nshow = int(show_time/dt)
      print "(t8,a,t20,a)", "time","displacement"
      print *, ""
   end if
   distance_fallen = y0 - y
   print "(2f13.4)", t,distance_fallen
end subroutine print_table
```

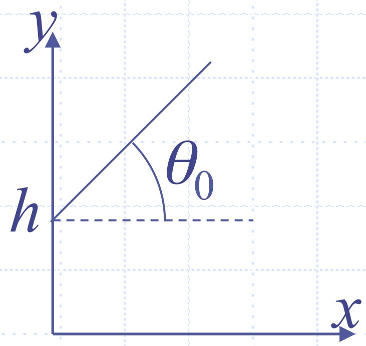
```
subroutine Euler_Richardson()  ! Euler-Richardson method
  real (selected_real_kind(15,307)) :: v2,vmid,v2mid,ymid,a,amid
  v2 = v*v
  a = -g*(1 - v2/vt2)
  vmid = v + a*dt_2                   ! velocity at midpoint
  ymid = y + v*dt_2                   ! position at midpoint
  v2mid = vmid*vmid
  amid = -g*(1 - v2mid/vt2)           ! acceleration at midpoint
  v = v + amid*dt
  y = y + vmid*dt
  t = t + dt
end subroutine Euler_Richardson
```

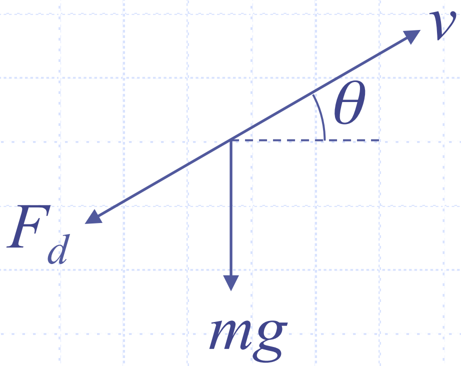
Note similarities and difference between programs!

Two-Dimensional Trajectories

$$m\frac{dv_x}{dt} = -F_d\cos\theta,$$

$$m\frac{dv_y}{dt} = -F_d\sin\theta - mg$$





$$\text{e.g., } F_d = k_2 v^2,$$

$$\frac{dv_x}{dt} = -Cvv_x,$$

$$\frac{dv_y}{dt} = -g - Cvv_y, \quad \text{where } C = \frac{k_2}{m}$$

```
subroutine Euler_Richardson(x,y,vx,vy,t,C,g,t,dt,dt_2)
  real*4 :: x,y,vx,vy,t,C,g,t,dt,dt_2
  real*4 :: v2,v,ax,ay,vxmid,vymid,xmid,ymid
  v2 = vx*vx+vy*vy
  v = sqrt(v2)
  ax = -C*v*vx
  ay = -g - C*v*vy
  vxmid = vx + ax*dt_2            ! velocity at midpoint
  vymid = vy + ay*dt_2
  xmid = x + vx*dt_2              ! position at midpoint
  ymid = y + vy*dt_2
  vmid2 = vxmid*vxmid + vymid*vymid
  vmid = sqrt(vmid2)
  axmid = -C*vmid*vxmid        ! acceleration at midpoint
    … (vx,vy,x,y)
  t = t + dt
end subroutine Euler_Richardson
```

Levels of Simulation

- Simplicity and Applicability.
- Approximation always made in modelling, check validity of those approximation.
- The level of simulation that is needed in a model depends on the accuracy of the corresponding experimental data and the type of information in which we are interested.
- The level of details that we can simulate also depends on the available computer resources.

Lecture 3 Review

- Solve second order differential equation as two coupled first order differential equations.
- Drag force, terminal speed, etc.
- **The Euler method and its modifications**.
- **Program for Newton's Law of motion**.
- Projectile motion.
- Simulation level and applications.

Further Applications

- All phenomena that can be described by second-order differential equations.
  - e.g., The Black-Scholes Model

$$\frac{\partial f}{\partial t} + rS\frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} = rf.$$

S : Stock price
f : Price of a derivative security contingent on S
r : Risk - free interest rate
μ : Drift rate
σ : Variance rate of S