

Lec2 - More mechanics

- Summary of lec1
- $D > 1$. Python functions
- Drawing tracks
- Lissajou figures
- Better integrators than Euler – leapfrog

Summary

- What is Computational Science ?
 - Discipline concerned with solving mathematical models of physical systems using computers
 - Small systems – “exact” solutions.
 - Large systems – simulations. Modeling. New types of phenomena not visible with few d.o.f
- Earliest origins in trying to predict motions in solar system - mechanics + gravity
- Mechanics: simplest method or *algorithm* to solve Newton’s laws on computer: *Euler method*

Python code

Create a code euler.py. Inside

```
from visual import *
# no automatic scaling of window
scene.autoscale=0
# size of window
scene.range=10.0
# standard Python 3D object
ball=sphere(radius=0.5,pos=(0,8.0,0))
ball.vel=vector(0,0,0)
dt=0.01
t=0
# loop over time (for ever!)
while(1):
    rate(100)
    t=t+dt
    ball.pos=ball.pos+ball.vel*dt
    ball.vel=ball.vel-ball.pos*dt
```

2D Motion

Identical equations for all components eg (x, y)
and (v_x, v_y) eg. new equations for y -motion

$$\frac{dy}{dt} = v_y$$
$$\frac{dv_y}{dt} = F_y/m$$

Same Python code will automatically integrate them!

Just need to change initial conditions to have $v_y(t = 0) \neq 0$ and add y -component to force
What about 3D ??

General forces

Nice to rewrite Python code to make it easier to keep the force definition separate from the Euler update.

```
...  
from force import *  
while(1):  
    rate(100)  
    t=t+dt  
    ball.pos=ball.pos+ball.vel*dt  
    ball.vel=ball.vel+  
    force(ball.pos,ball.vel)/ball.mass*dt
```

Need a new code in file force.py

Force function

```
# 2d oscillator
def force(pos,vel):
    result = vector(0,0,0)
    result.x=-1.0*pos.x
    result.y=-1.0*pos.y
    result.z=0
    return result
```

Comments

- `def force()` is followed by code *defining* what the function does.
- Note indentation showing the lines of code inside the definition.
- `force` function takes *vector arguments* and returns object `result` of type `vector`
- `result.x` gives x-component of vector.
- `return` is *keyword* (like `import`, `def`, `while` etc)

Drawing the path

Nice with 2D,3D problems to visualize the trajectory. Need the `track` attribute which adds an object of type `curve` which is standard in VPython.

```
....
ball=sphere(radius=0.5,pos=(0,8.0,0),
track=curve(radius=0.1))
ball.vel=vector(5,5,0)
dt=0.01
t=0

while 1:
    rate(100)
    t=t+dt
    ball.pos=ball.pos+ball.vel*dt
    ball.vel=ball.vel+force*dt
    ball.track.append(pos=ball.pos)
```

curve()

- `curve()` is really a list of coordinate points which can be dynamically added to.
- Also knows how to plot itself to the screen
- Can be given attributes like `radius`, `color` etc
- Like other standard Vpython objects can be added as an attribute to `sphere()`

Lissajou figures

- Imagine using different harmonic springs in each direction. What is resulting motion ?
- Does motion close on finite length trajectory ? When ? What condition is needed ...
- Finite orbits termed *Lissajou figures*

Time step errors

- The Euler method we have used so far has its limitations – solution accurate to $O(dt)$ only. Why ? error per step $O(dt^2)$. Total steps T/dt . Total error $dt^2 T/dt = Tdt$
- See this by computing *energy*
$$E = 1/2mv^2 + 1/2kx^2$$
- Compute error as function of dt ...
- Note: calculate mean error over some finite period of time - see code.
- Can do better ...

More accurate integrators

Using Taylor

$$x(t + dt) = x(t) + v(t)dt + a(t)dt^2/2 + O(dt^3)$$

leading to

$$x_{n+1} = x_n + v_n dt + a_n dt^2/2$$

Subtracting $x(t - dt)$ from $x(t + dt)$ find

$$v(t) = \frac{x(t + dt) - x(t - dt)}{2dt} + O(dt^3)$$

leading to

$$v_{n+1} = \frac{x_{n+2} - x_n}{2dt}$$

Substituting for x_{n+2} using previous equation:

$$v_{n+1} = v_n + \frac{dt}{2}(a_n + a_{n+1})$$

leapfrog algorithm

Accurate to $O(dt^2)$

Leapfrog algorithm

Almost as simple to implement in code.

- Compute and store new variable `oldforce` to keep initial acceleration.
- `force()` function same as before.
- See *much* smaller errors in energy. Consistent with $\frac{\Delta E}{E} \sim dt^2$
- *Symplectic integrator*. Time reversible. Exactly conserves some “nearby” Hamiltonian.

Summary

- Functions. Separate off code for particular force from code to integrate Newton's laws. Easier to debug. Less code to write when tackling new situation.
- 2D,3D. Drawing tracks.
- Playing with harmonic forces ...
- Better integrators – leapfrog