

Lec1 - Intro, Python

- Mechanics, syllabus
- General comments
- What is computational science ?
- Simple mechanics and intro to Python

General comments

- Not programming course. Not traditional physics course.
- Topics drawn from many different areas of physics – mechanics, waves, statistical physics, quantum physics
- Brief discussion of programming issues - *only as much as needed to do the science*
- Python/VPython makes life easy. Don't need to know much to do some quite complicated stuff (graphics built-in)

Computational Science

2 main facets:

- Provide set of tools (algorithms, efficient code) to produce (numerical) *quantitative* solutions to known scientific laws (expressed in math form). Typically small number of degrees of freedom Essentially zero error.
- Simulate a complex model of the system. Extract *qualitative* as well as quantitative information. Often statistical errors - *computer experiment*. Sometimes coming up with the model is part of the game. Use to explore different possible theories. Again good tools, algorithms needed.

Examples

Type 1: throwing a ball into the air - time to reach ground ?

a) Newton's laws, analytic solution (neglect air resistance, model as point particle)

b) Real world: put in air resistance, spin of ball, can *integrate the equations numerically* with high accuracy.

Type 2: Show that water freezes when cooled.

Features:

- Many d.o.f, complex interactions
- Qualitative change of state of system.

Errors: finite simulation time. Finite number of particles.

Newton's laws

One particle, one dimension:

$$\frac{dx}{dt} = v$$
$$\frac{dv}{dt} = a = F(x, v, t)/m$$

First order form important!

Generalize to three dimensions and many particles later

Definitions of velocity and acceleration.

Euler I

What is meant by derivatives ?

Nothing more than:

$$\frac{dx}{dt} = \lim_{dt \rightarrow 0} \frac{x(t + dt) - x(t)}{dt}$$

Insert in first of Newton's laws:

$$dtv(t) \sim x(t + dt) - x(t)$$

turning this around

$$x(t + dt) = x(t) + dtv(t)$$

If know $v(t)$ and use a *small enough* time step dt will give approx for $x(t + dt)$

Do same for $v(t + dt)$

Euler II

Divide time into *discrete* steps $t = ndt, n = 0, 1 \dots$

Full algorithm:

$$x_{n+1} = x_n + dtv_n$$

$$v_{n+1} = v_n + dtF_n/m$$

Note $x_n \equiv x(ndt)$

Start from some x_0, v_0 . Generate x_1, v_1 , then x_2, v_2 , keep going

Accurate to $O(dt)$

Simple example – unit mass harmonic oscillator

$$F = -x$$

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
```

Comments

- `import` statement allows us to use the 3D graphics module called `VPython`.
- No automatic scaling of window to simulation. Fix size of window with `range` explicitly.
- `rate` is a Python function which controls the speed at which graphics is drawn to the screen. A large argument means a faster simulation in real time.
- Simple assignment and arithmetic. The statement `t=t+dt` should be read as “ add the variable `dt` to the variable `t` and put the result back in the variable `t`”.

VPython Objects

- `sphere` is a standard Python *object*. It may have a number of attributes such as position, color, size etc. You may also add attributes such as velocity (`vel`) which is defined to be a vector.
- `ball` is the name of a specific one of these `sphere` objects created in this code with the line
`ball=sphere(...)`
- Attributes such as `pos` are accessed using the dot operator eg. `ball.pos`.

Control and Indentation

`while(arg):` command tells the computer to do something until `arg` is true (1 in computer speak).

Structures like `while` typically act on a group of Python commands

Notice the colon

You can tell which ones by the level of indentation.

Eg. in the last code the lines

```
    rate(100)
    t=t+dt
    ball.pos=...
    ball.vel=...
```

are all carried out in the `while` loop.

IDLE (Python editor) automatically indents codes that follows such a command.

Real oscillators

Can add drag force $f = -kv$ (why minus sign ?)

Python code line changes:

```
ball.vel=ball.vel-ball.pos-0.1*ball.vel*dt
```

What is k ? - what happens larger k ?

Damped oscillations - overdamping

Linear damping analytic solution possible but not in general

But numerical solution *easy* – just change force to say $f = -kv^2$!

Summary

- General comments.
- Newton's laws - numerical solution via Euler alg.
- Simple assignment, arithmetic, while loops.
Indentation