

Lec2 - mechanics and simple simulation

- Newton's laws, Euler method
- 1D and 2D examples: projectiles, harmonic motion, damping
- Simulation code and graphics
- More on functions

Newton's laws

One particle, one dimension:

$$\frac{dx}{dt} = v$$
$$\frac{dv}{dt} = a = F/m$$

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:

$$\begin{aligned}x_{n+1} &= x_n + dtv_n \\v_{n+1} &= v_n + dtF_n/m\end{aligned}$$

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 $O(dt)$

Note: F can be $F(t, x, v)$!

Python

```
# no automatic scaling of window
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

- Simple oscillator
- `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.
- `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 here *velocity* which is defined to be a vector.
- Since VPython is a 3D environment we must work with *vectors*
- No automatic scaling of window to simulation. Fix size of window with `range` explicitly

Indentation

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

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 automatically indents codes that follows such a command.

Indentation II

Such structures can be *nested* eg. in the root program:

```
while (u-1)>tol:
    m=(u+1)*0.5
    fm=exp(m)-0.5
    if (fm*fu>0.0):
        u=m
    ...
```

The statements `m=...` and `if(fm*fu>0.0):` are under control of the `while` command.

The statement `u=m` is separately under control of the `if` command

Real oscillators

Can add drag force $f = kv$ Python code line changes:

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

Note:here $k = 0.1$ - what happens larger k ?

Damped oscillations - overdamping

Linear damping analytic solution possible but not in general

But numerically – easy to change to say $f = kv^2$!

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 $v_y(t = 0) \neq 0$

Drawing the path

Nice with 2D,3D problems to visualize the trajectory. Need the track attribute.

```
from visual import *

# no automatic scaling of window
scene.autoscale=0
scene.range=10.0

# use a standard Python 3D object to represent part
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-ball.pos*dt
    ball.track.append(pos=ball.pos)
```

Projectiles revisited

Change force to form $(0, -1, 0)$. Gravity near Earth's surface.

...

```
ball=sphere(radius=0.5,pos=vector(-9,0,0),
track=curve(radius=0.1))
ball.vel=vector(3,3,0)
dt=0.01
t=0
force=vector(0,-1.0,0.0)
true=1

while true:
    rate(100)
    t=t+dt
    ball.pos=ball.pos+ball.vel*dt
    ball.vel=ball.vel+force*dt
    ball.track.append(pos=ball.pos)
    if (ball.y<0.0) :
        true=0
```

General program for 1 particle

```
...
```

```
# damped 2d oscillator
```

```
def force(pos,vel,t):
```

```
    result = vector(0,0,0)
```

```
    result.x=-1.0*pos.x-0.1*vel.x
```

```
    result.y=-1.0*pos.y-0.1*vel.y
```

```
    result.z=0
```

```
    return result
```

```
ball=sphere(...)
```

```
ball.vel=vector(...)
```

```
dt=0.01
```

```
t=0
```

```
while 1:
```

```
    ball.pos=ball.pos+dt*ball.vel
```

```
    ball.vel=ball.vel+dt*force(ball.pos,ball.vel,t)/
```

```
    ball.mass
```

Comments

- mass is a (user-defined) attribute
- force function takes vector arguments and returns objects of type vector
- return is keyword (like if, while etc)