L2 Computational Physics
Week 4 – 2nd order ODEs

This week...

- 2^{nd} order differential equations
  - Harmonic oscillator
  - Ballistics
  - Pendulum

- Black Box ODE solvers

- Phase space and Orbits

2^{nd} order ODEs

2nd order ODEs

- 1st order (last week)

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

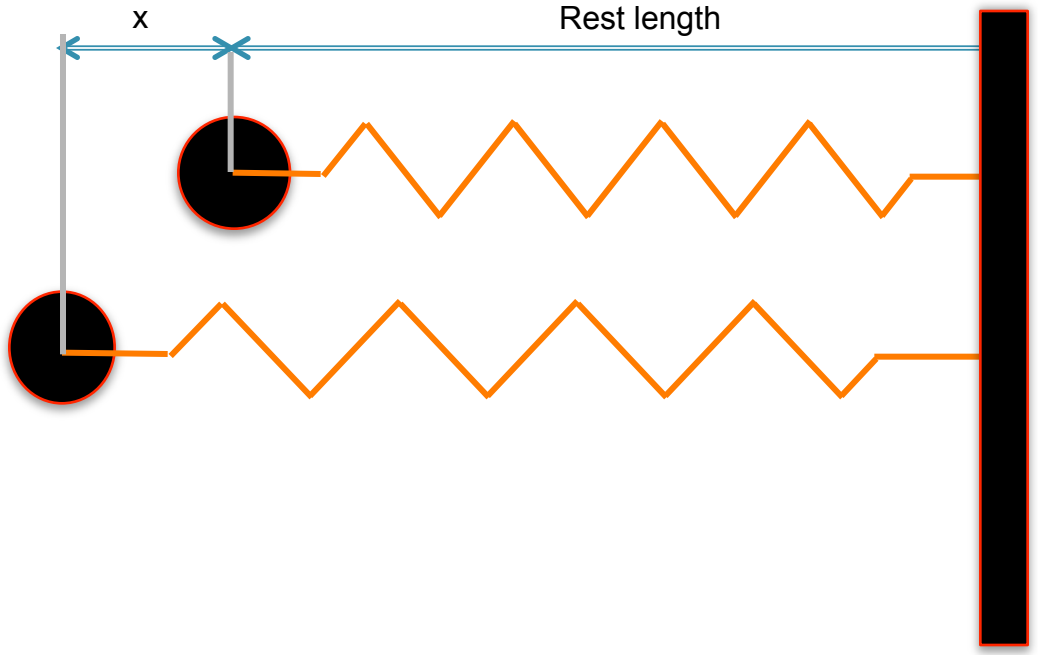
- 2nd order (this week)

$$\frac{d^2x}{dt^2} = f(x,t)$$

Harmonic Oscillator

Mass on a Spring

- Mass m
- Spring constant k



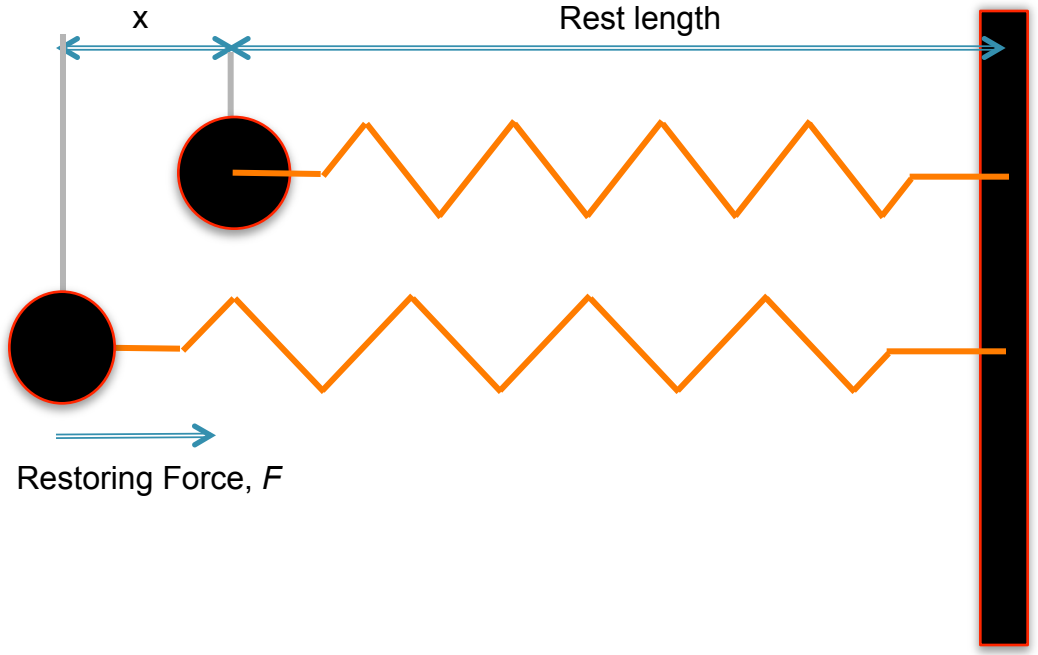
Mass on a Spring

- Mass m
- Spring constant k

- Extension x

$$F = -kx$$

$$\frac{d^2x}{dt^2} = -\frac{k}{m}x$$



Solve with Euler

- Euler, Heun, RK4 et al are for 1st order ODEs
- Rewrite our 2nd order DEQ in terms of 1st order equations – Introduce linear velocity, *v*
- Now we have a pair of first order DEQs

$$\frac{d^2x}{dt^2} = -kx$$

$$\frac{dx}{dt} = v$$

$$\frac{dv}{dt} = -\frac{k}{m}x$$

Solve with Euler

- Euler, Heun, RK4 et al are for 1st order ODEs
- Rewrite our 2nd order DEQ in terms of 1st order equations
  – Introduce linear velocity, *v*
- Now we have a pair of first order DEQs
  – The equations are 'coupled'

$$\frac{d^2x}{dt^2} = -kx$$

$$\frac{dx}{dt} = v$$

$$\frac{dv}{dt} = -\frac{k}{m}x$$

Solve with Euler

- Recap Euler:

$$x_{t+\Delta t} \approx x_t + f(x_t, t)\Delta t$$

$$f(x_t, t) \equiv \left.\frac{dx}{dt}\right|_{x_t, t}$$

Solve with Euler

- Recap Euler: $x_{t+\Delta t} \approx x_t + f(x_t, t)\Delta t$

- Harmonic Oscillator: $\frac{dx}{dt} = v, \frac{dv}{dt} = -\frac{k}{m}x$

Solve with Euler

- Recap Euler: $x_{t+\Delta t} \approx x_t + f(x_t, t)\Delta t$
- Harmonic Oscillator: $\frac{dx}{dt} = v, \frac{dv}{dt} = -\frac{k}{m}x$
- Ta-da:

$$x_{t1} = x_{t0} + v\Delta t$$

$$v_{t1} = v_{t0} - \frac{k}{m}x\Delta t$$

```
from __future__ import division
import numpy
import matplotlib.pyplot as pyplot

m=0.1
k= 0.4

x0=0.3
v0=0.0
t0=0

t1=35
n_panels=500
dt=(t1-t0)/n_panels

xs = numpy.zeros((n_panels),)
vs = numpy.zeros((n_panels),)
ts = numpy.zeros((n_panels),)

def dx_dt(x,v):
    return v

def dv_dt(x,v):
    return -(k/m)*x
```

Model Parameters

Initial conditions

timestep

arrays

Differential equations!

```
for i in range(n_panels-1):
    k0x=dx_dt(x0,v0)
    k0v=dv_dt(x0,v0)
    x1=x0+k0x(x0,v0)*dt
    v1=v0+k0v(x0,v0)*dt

    xs[i+1]=x1
    vs[i+1]=v1
    ts[i+1]=t0+i*dt
    x0=x1
    v0=v1
```

Integrate via euler

```
pyplot.figure(figsize=(8,8))
pyplot.subplot(211)
pyplot.plot(ts,xs,color='red')
pyplot.xlabel('time (s)')
pyplot.ylabel('position (m)')
pyplot.subplot(212)
pyplot.plot(xs,vs,color='red')
pyplot.xlabel('position (m)')
pyplot.ylabel('velocity (m/s)')

pyplot.show()
```

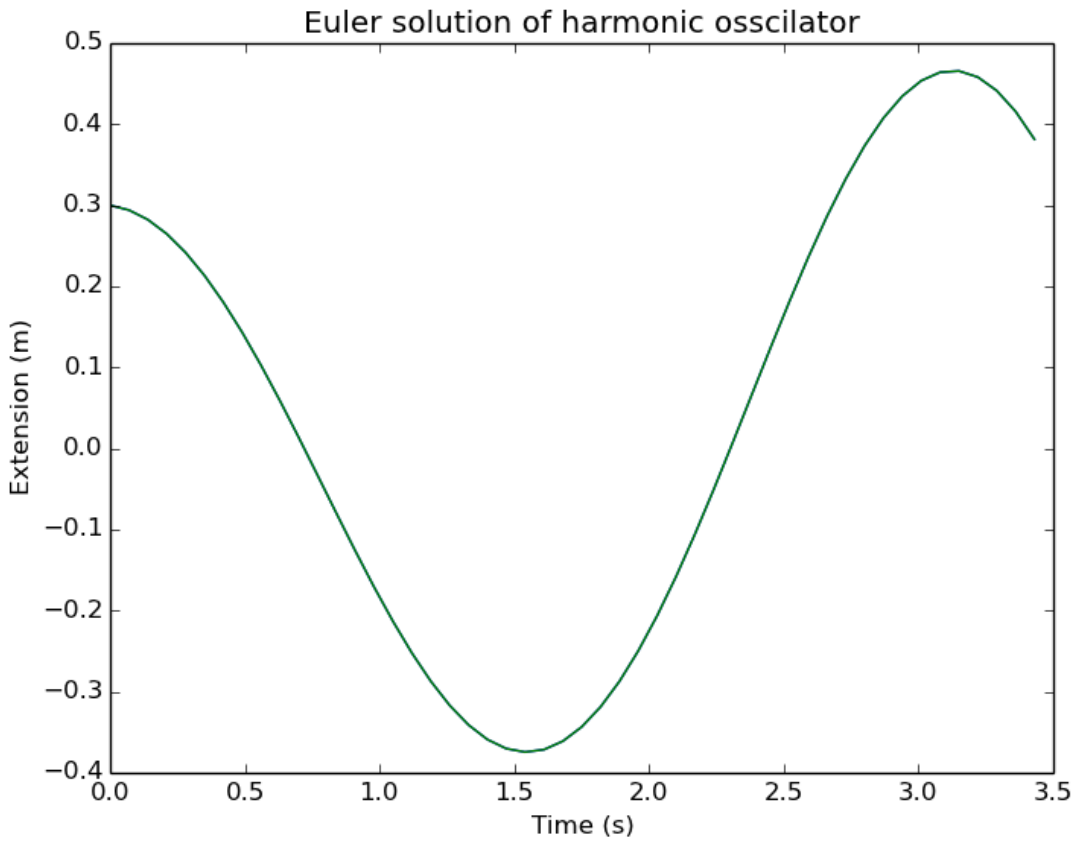
Plot

Euler Timestep

- Within our code we have two states
  - Before the timestep
    - `x0, v0`
  - After the timestep
    - `x1, v1`
- Any simulation code (not just DEQs) has to carefully separate the present and the future
- 'Tick Tock'

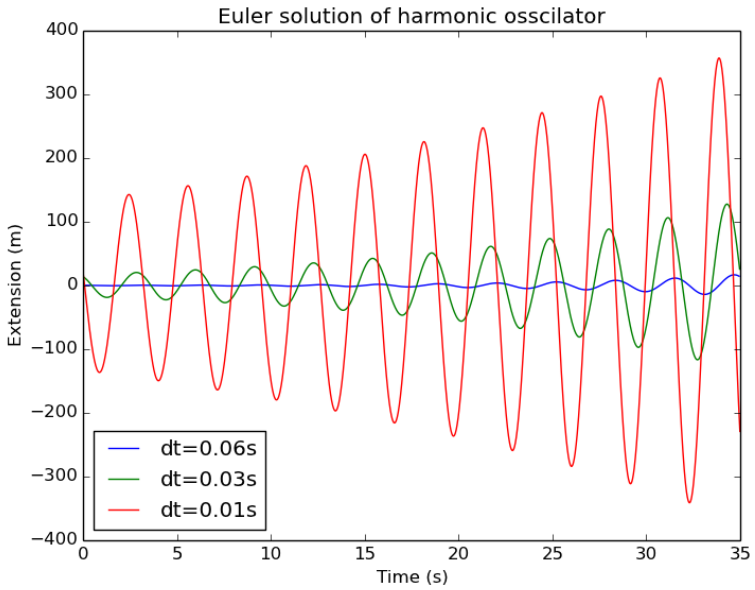
```
for i in range(n_panels-1):
    k0x=dx_dt(x0,v0)
    k0v=dv_dt(x0,v0)
    x1=x0+k0x(x0,v0)*dt
    v1=v0+k0v(x0,v0)*dt

    xs[i+1]=x1
    vs[i+1]=v1
    ts[i+1]=t0+i*dt
    x0=x1
    v0=v1
```

Let's run it!

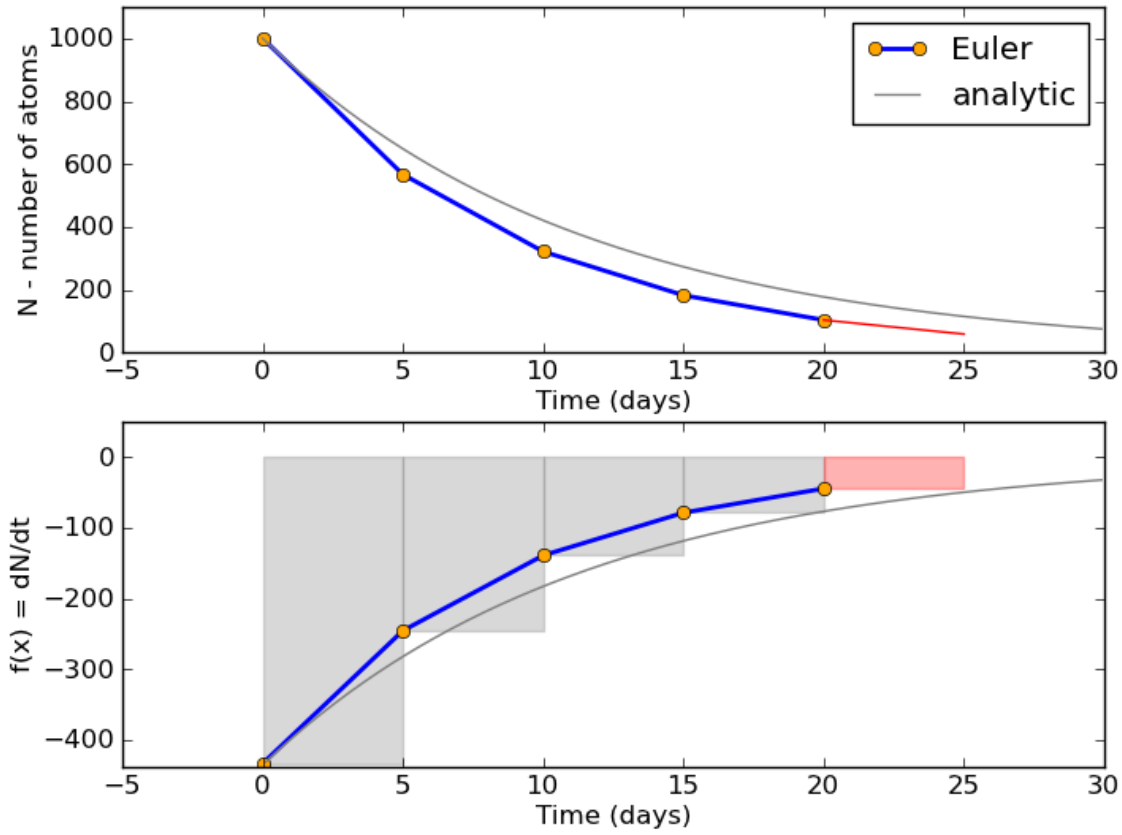

Euler solution of harmonic osscilator
Extension (m)
Time (s)
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

Euler is failing us

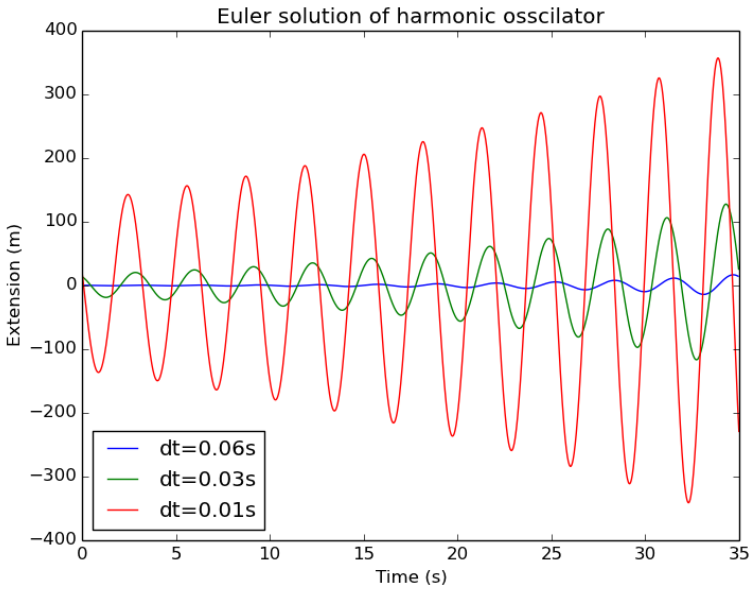
- Euler method has an error in it



Flashback – nuclear decay



Euler is failing us



- Euler method has an error in it
- The error accumulates over time
  - Problematic for decay
  - Catastrophic for periodic systems!

Energy Conservation

- **Physics 101 – Energy is conserved**
- The error in the Euler timestep violates conservation of energy
- There's an error in total energy in each timestep:

$$0.5\frac{k}{m}\left(kx^2 + mv^2\right)\left(\Delta t\right)^2$$

- We need a better method!

Energy Conservation

There's an error total energy in each timestep:

$$0.5\frac{k}{m}\left(kx^2 + mv^2\right)\left(\Delta t\right)^2$$

- The error itself is always positive and is proportional to KE (v^2) and PE (x^2)

- Over one full cycle of the pendulum:

$$E(t+\Delta t) = E(t) + \alpha E(t)$$

$$E(t+\Delta t) = (1+\alpha)E(t)$$

Energy Conservation

There's an error total energy in each timestep:

$$0.5\frac{k}{m}\left(kx^2 + mv^2\right)\left(\Delta t\right)^2$$

- The error itself is always positive and is proportional to KE (v^2) and PE (x^2)

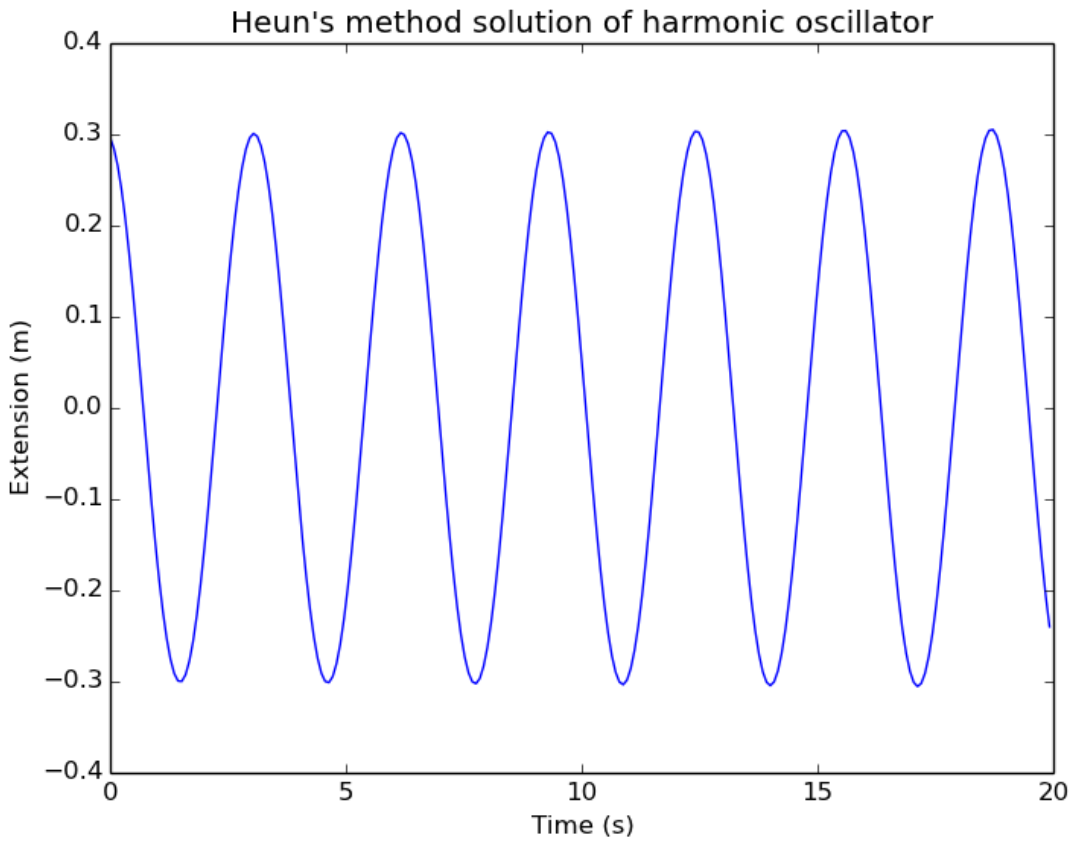
- Over one full cycle of the pendulum:

$$E(t+\Delta t) = E(t) + \alpha E(t)$$

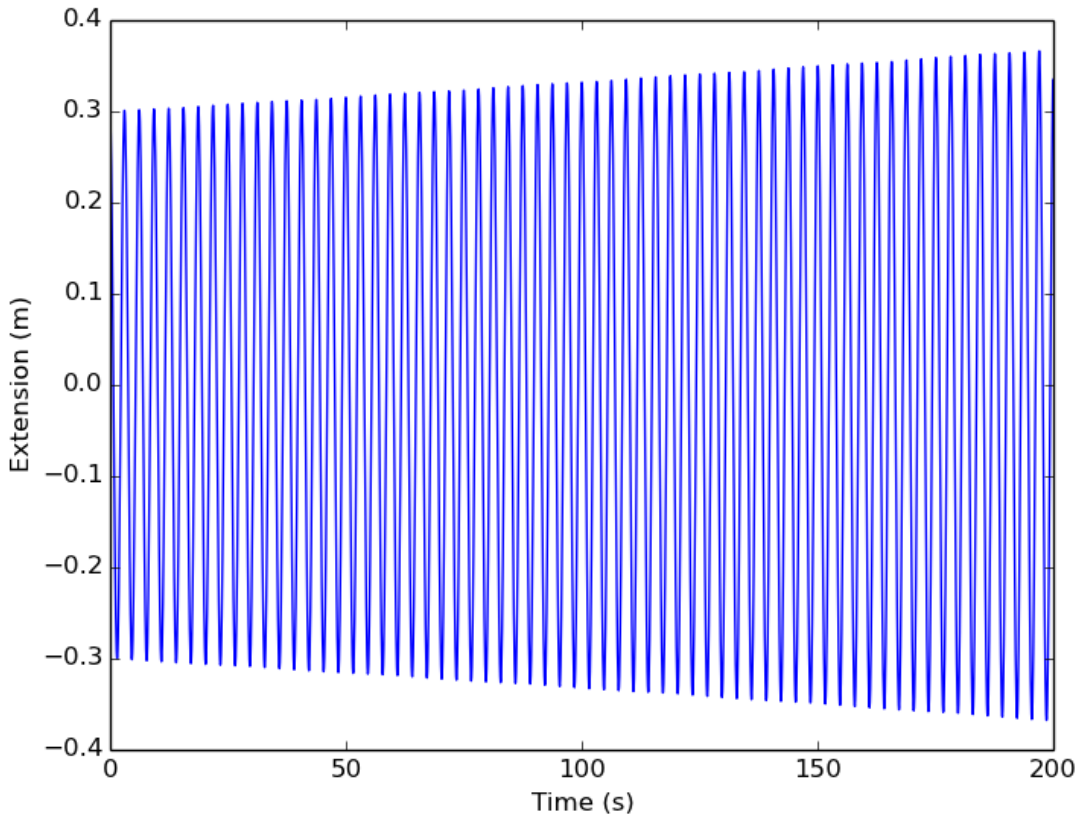
$$E(t+\Delta t) = (1+\alpha)E(t)$$

Exponential growth!

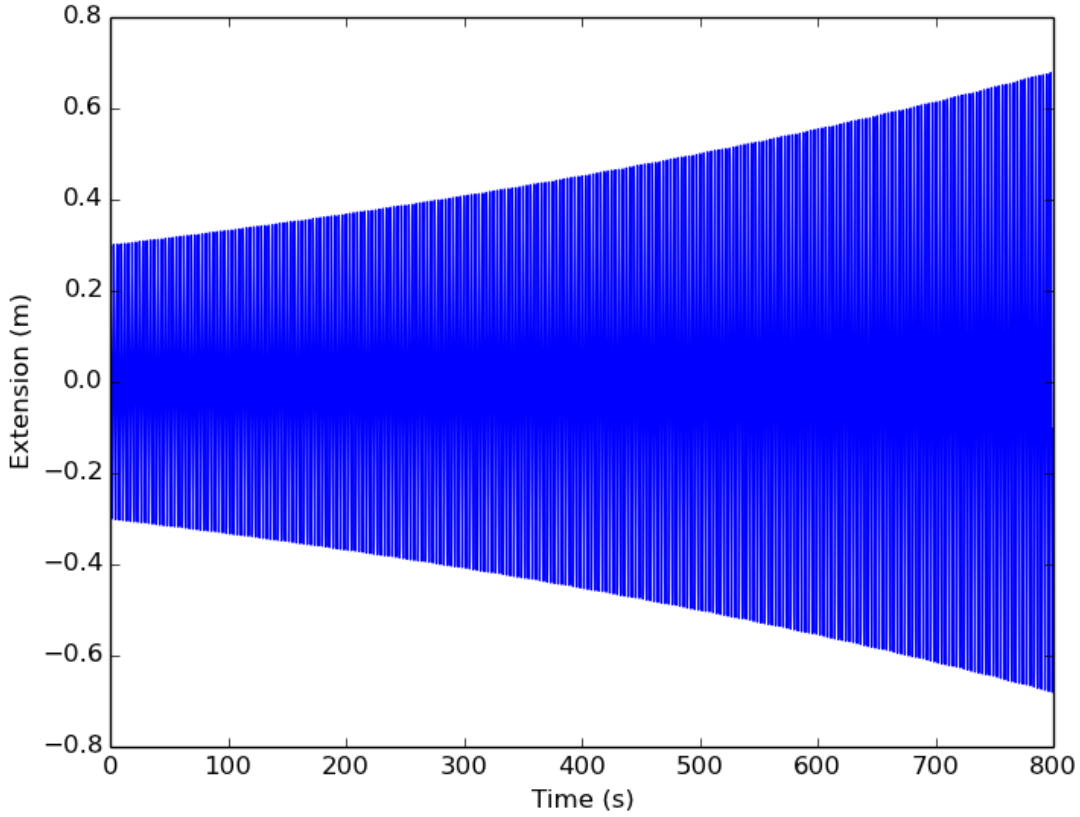
Let’s try Heun



Close...



... but no cigar


0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
Extension (m)
0
100
200
300
400
500
600
700
800
Time (s)

EnergyCons_Euler

IPy localhost:8889/notebooks/EnergyCons_Euler. Reader

IP[y]: Notebook EnergyCons_Euler

File Edit View Insert Cell Kernel Help

Code Cell Toolbar: None

In [1]:
```
from __future__ import division
import sympy
from IPython.display import display
sympy.init_printing()
```

In [2]:
```
v0 = sympy.Symbol('v_0') # Velocity at start of timeste
x0 = sympy.Symbol('x_0') # Extension at start of timestep
dt = sympy.Symbol('dt')  # Timestep
k  = sympy.Symbol('k')   # Spring constant
m  = sympy.Symbol('m')   # Mass
```

In [3]:
```
v1 = v0 - (k/m) * x0 * dt # Velocity after timestep
x1 = x0 + v0         * dt # Extension after timestep
```

In [4]:
```
def spring_energy(vel, extension):
    return 0.5*m*vel**2 + 0.5*k*extension**2
```

In [5]:
```
e0 = spring_energy(v0, x0)
e1 = spring_energy(v1, x1)
display(e0)
display(e1)
```

$$0.5kx_0^2 + 0.5mv_0^2$$

$$0.5k\left(dtv_0 + x_0\right)^2 + 0.5m\left(-\frac{dtk}{m}x_0 + v_0\right)^2$$

In [7]:
```
(e1-e0)
```

Out[7]:
$$-0.5kx_0^2 + 0.5k\left(dtv_0 + x_0\right)^2 - 0.5mv_0^2 + 0.5m\left(-\frac{dtk}{m}x_0 + v_0\right)^2$$

In [8]:
```
(e1-e0).expand().simplify()
```

Out[8]:
$$\frac{0.5k}{m}dt^2\left(kx_0^2 + mv_0^2\right)$$

EnergyCons_Heun

IPy localhost:8889/notebooks/EnergyCons_Heun Reader

IP[y]: Notebook EnergyCons_Heun

File Edit View Insert Cell Kernel Help

Code Cell Toolbar: None

In [10]:
```
# k0 = dx/dt at t0=
k0_v = -(k/m) * x0
k0_x =        v0
# x' = euler prediction
vp = v0 + k0_v * dt
xp = x0 + k0_x * dt
# k1 = dx/dt at euler prediction
k1_v = -(k/m) * xp
k1_x =        vp
# Averaged rate of change
k_v = (k0_v + k1_v)/2
k_x = (k0_x + k1_x)/2
# x1
v1 = v0 + k_v * dt
x1 = x0 + k_x * dt
#
v1 = v1.simplify()
x1 = x1.simplify()
```

In [11]:
```
display(v1)
display(x1)
```

$$\frac{1}{m}\left(-\frac{dtk}{2}\left(dtv_0 + 2x_0\right) + mv_0\right)$$

$$-\frac{dt^2kx_0}{2m} + dtv_0 + x_0$$

In [12]:
```
e0 = spring_energy(v0, x0)
e1 = spring_energy(v1, x1)
(e1-e0).expand().simplify()
```

Out[12]:
$$\frac{0.125dt^4}{m^2}k^2\left(kx_0^2 + mv_0^2\right)$$

Euler-Cromer

Euler

$$v_{t1} = v_{t0} - \frac{k}{m} x_0 \Delta t$$

$$x_{t1} = x_{t0} + v_0 \Delta t$$

Euler-Cromer

$$v_{t1} = v_{t0} - \frac{k}{m} x_0 \Delta t$$

$$x_{t1} = x_{t0} + v_1 \Delta t$$

Euler

$$v_{t1} = v_{t0} - \frac{k}{m} x_0 \Delta t$$

$$x_{t1} = x_{t0} + v_0 \Delta t$$

Euler-Cromer

$$v_{t1} = v_{t0} - \frac{k}{m} x_0 \Delta t$$

$$x_{t1} = x_{t0} + v_1 \Delta t$$

Euler-Cromer reorders the time-step slightly.

This change leads to energy conservation

There is still an error, but it cancels over one full cycle

Very useful for studying periodic systems

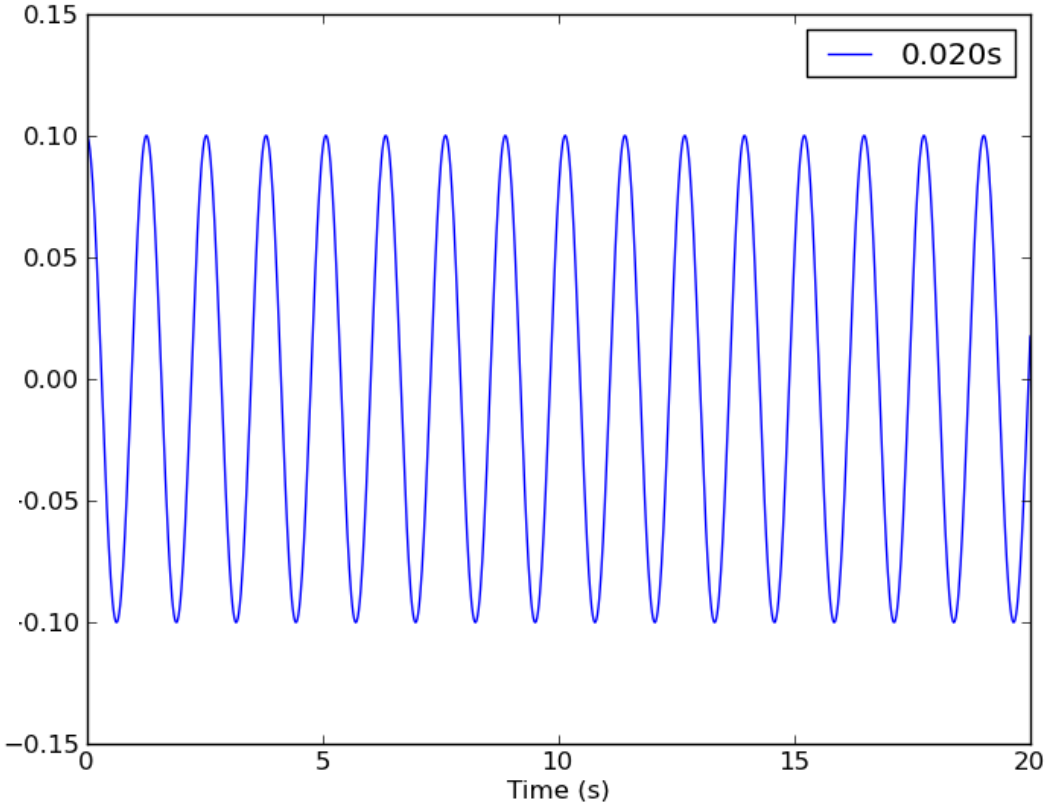
```
for i in range(n_panels-1):
    k0x=dx_dt(x0,v0)
    k0v=dv_dt(x0,v0)
    v1=v0+k0v(x0,v0)*dt
    x1=x0+k0x(x0,v1)*dt

    xs[i+1]=x1
    vs[i+1]=v1
    ts[i+1]=t0+i*dt
    x0=x1
    v0=v1
```

Integrate via euler – cromer!!

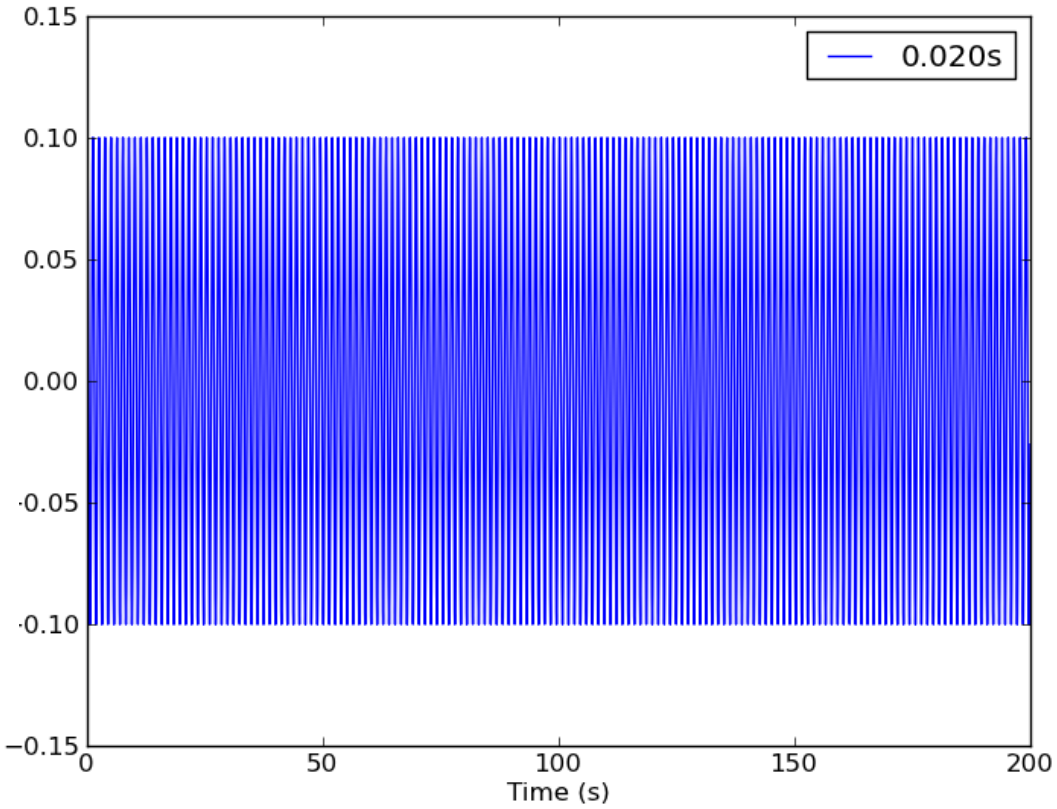
```
pyplot.figure(figsize=(8,8))
pyplot.subplot(211)
pyplot.plot(ts,xs,color='red')
pyplot.xlabel('time (s)')
pyplot.ylabel('position (m)’)
pyplot.subplot(212)
pyplot.plot(xs,vs,color='red')
pyplot.xlabel('position (m)')
pyplot.ylabel('velocity (m/s)')

pyplot.show()
```

Plot

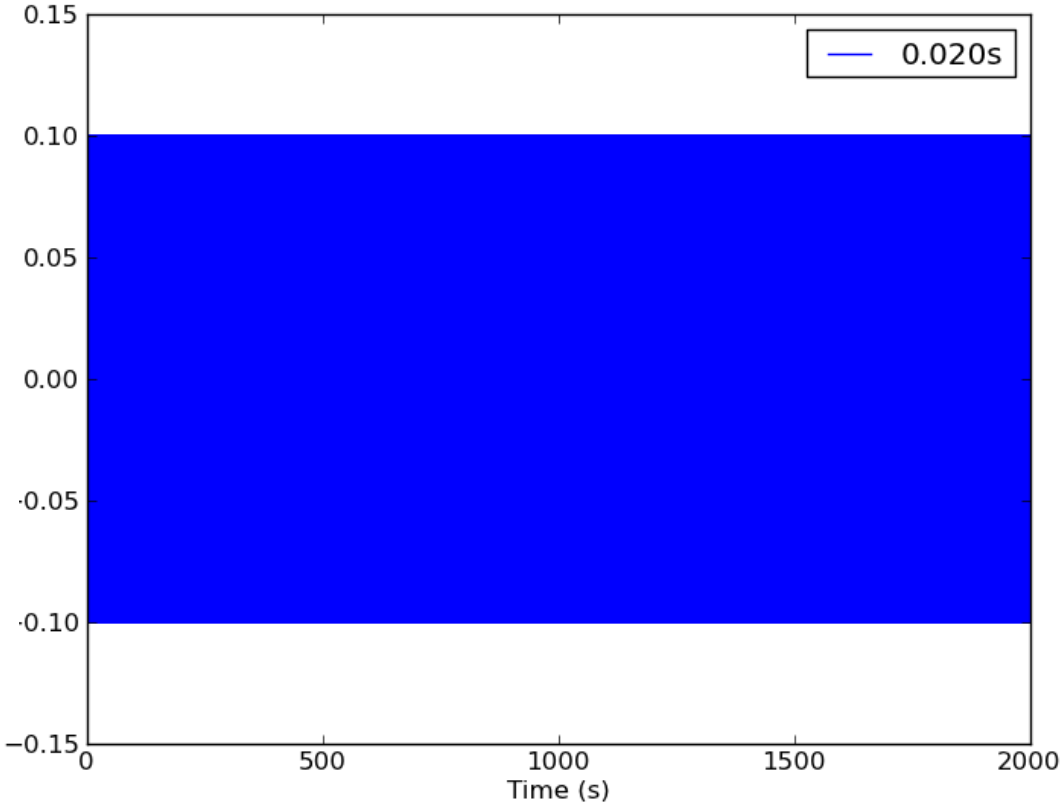
Euler-Cromer



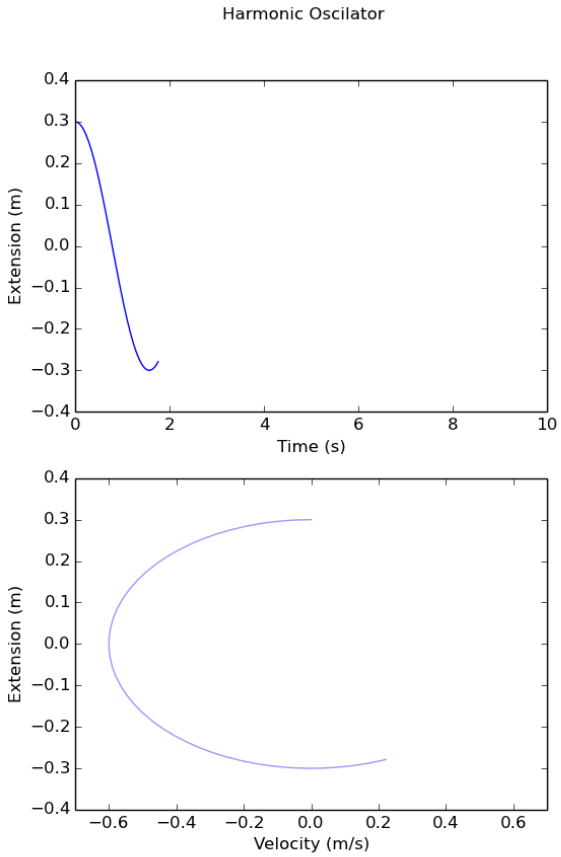
Euler-Cromer



Euler-Cromer



Phase space and orbits...



(movie on DUO)

Moral of the story

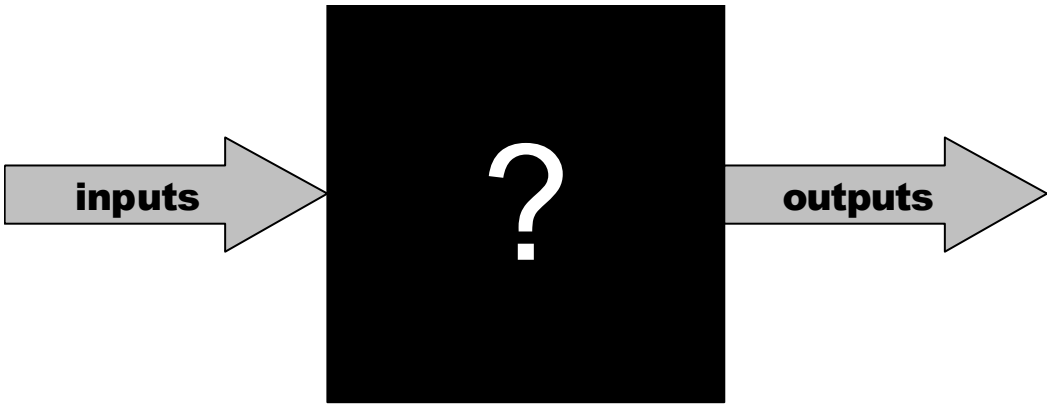
- Numerical methods are imprecise
  - Errors introduced
- Can distort results for a damped or bounded system
- Can be catastrophic for a periodic system

Black Box ODE solver

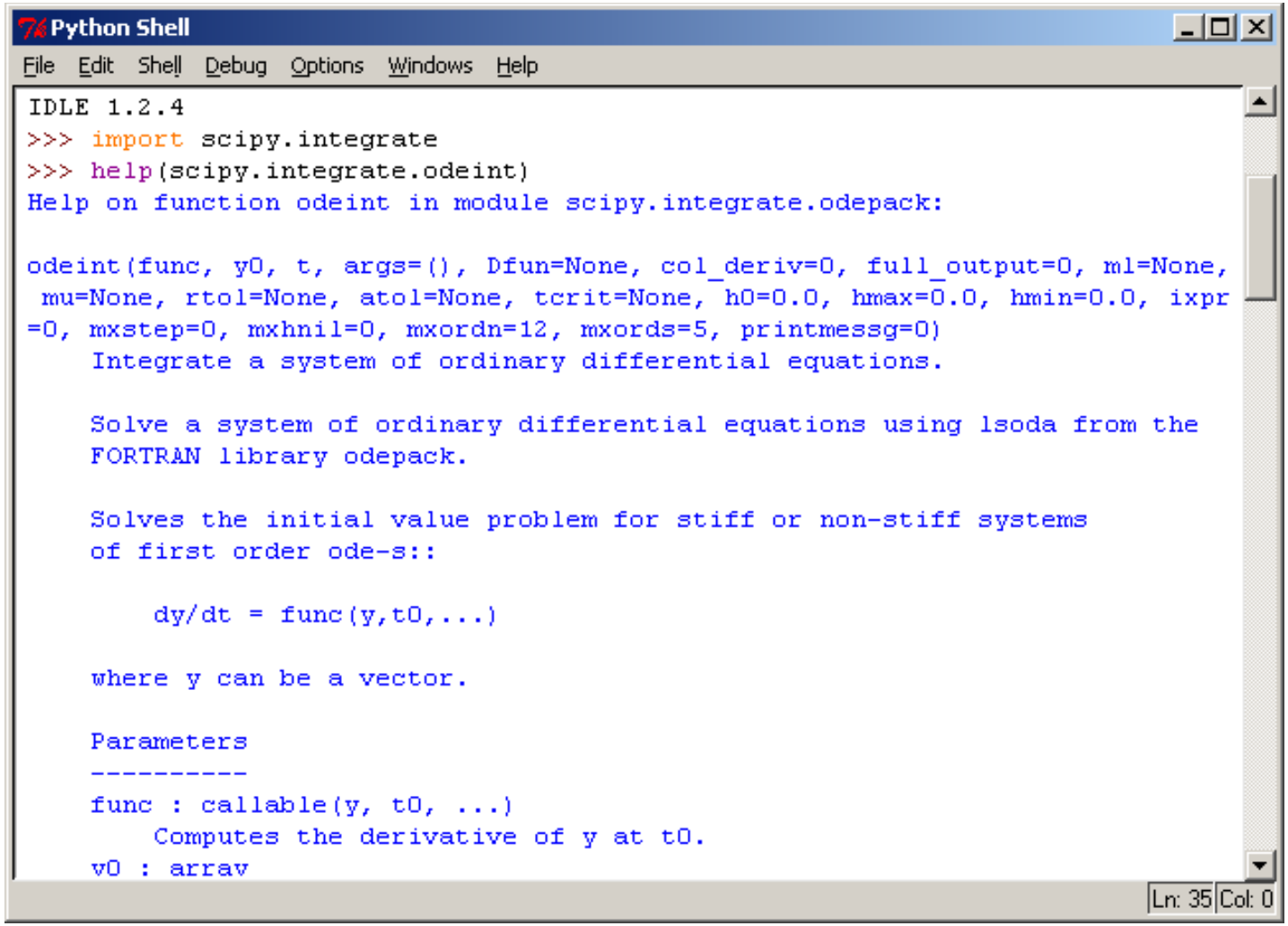
Black Box

- As with plain integration, lots of black box solvers for DEQs exist

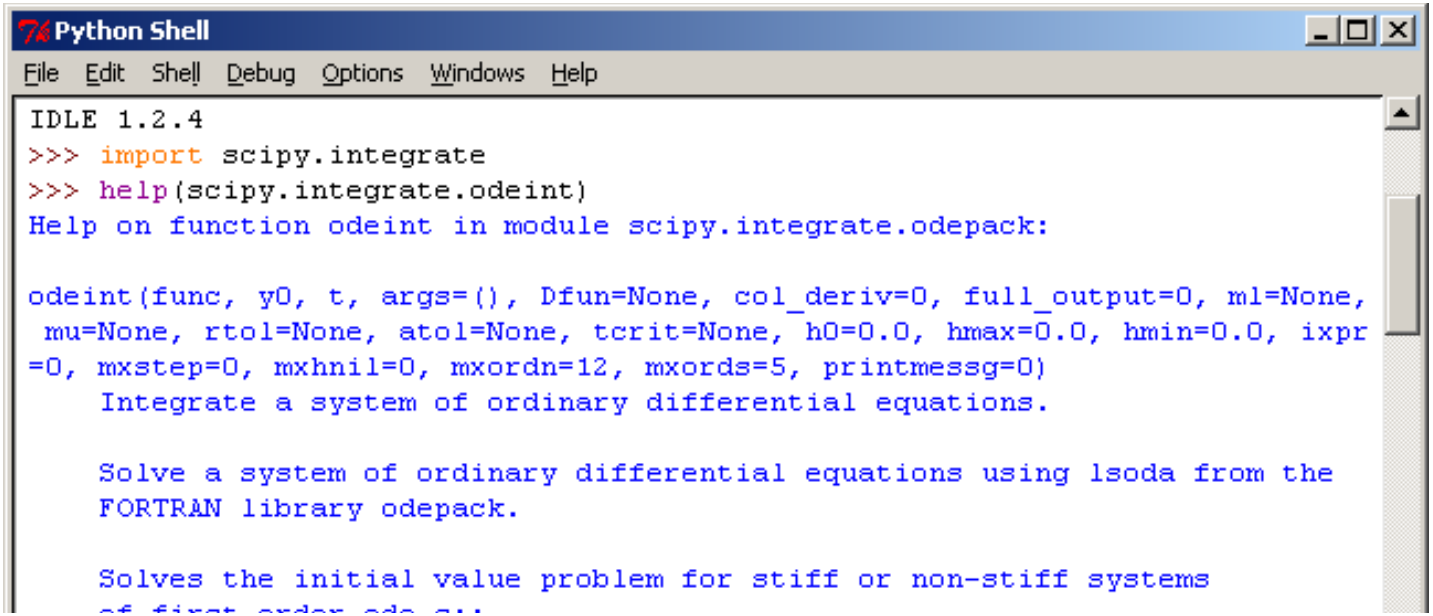
- `scipy.integrate.odeint`



scipy.integrate.odeint



scipy.integrate.odeint



Wow, that's a lot of options

It's a lot more than for scipy.integrate.quadrature (week 2)

Numerically solving ODEs is a very complex field!

Using odeint for the harmonic oscillator

- We need to follow a series of steps
  1. Represent the system state (x,v) as a vector
  2. Make a function that takes a system and returns a vector of (dx/dt, dv/dt)
  3. Make a series of timepoints for odeint to solve
  4. Invoke odeint

- Let's look at points 1-2 first

Code tidy-up

Using arrays

```
xs = numpy.zeros((n_panels),)
vs = numpy.zeros((n_panels),)

ts =  numpy.linspace(t0,t1,n_panels)

xs_ode=numpy.zeros((n_panels),)
vs_ode=numpy.zeros((n_panels),)
```

arrays

```
# required format for ODEINT
def f((x,v),time):
        dx_dt=v
        dv_dt=-(k/m)*x
       return numpy.array((dx_dt,dv_dt))
```

Call with a pair of x,v values – pass in as array for a single time value

Differential equations - combined

```
# state is a single pair of x,v values
state=numpy.array((x0,v0))

values=scipy.integrate.odeint(f, state, ts)

xs_ode=values[:,0]
vs_ode=values[:,1]


#compare to euler
for i in range(n_panels-1):
    k0x,k0v=f( (x0,v0),ts[i])
    xs_eul[i+1]=x0+k0x*dt
    vs_eul[i+1]=v0+k0v*dt
    x0=xs_eul[i+1]
    v0=vs_eul[i+1]
```

Integrate

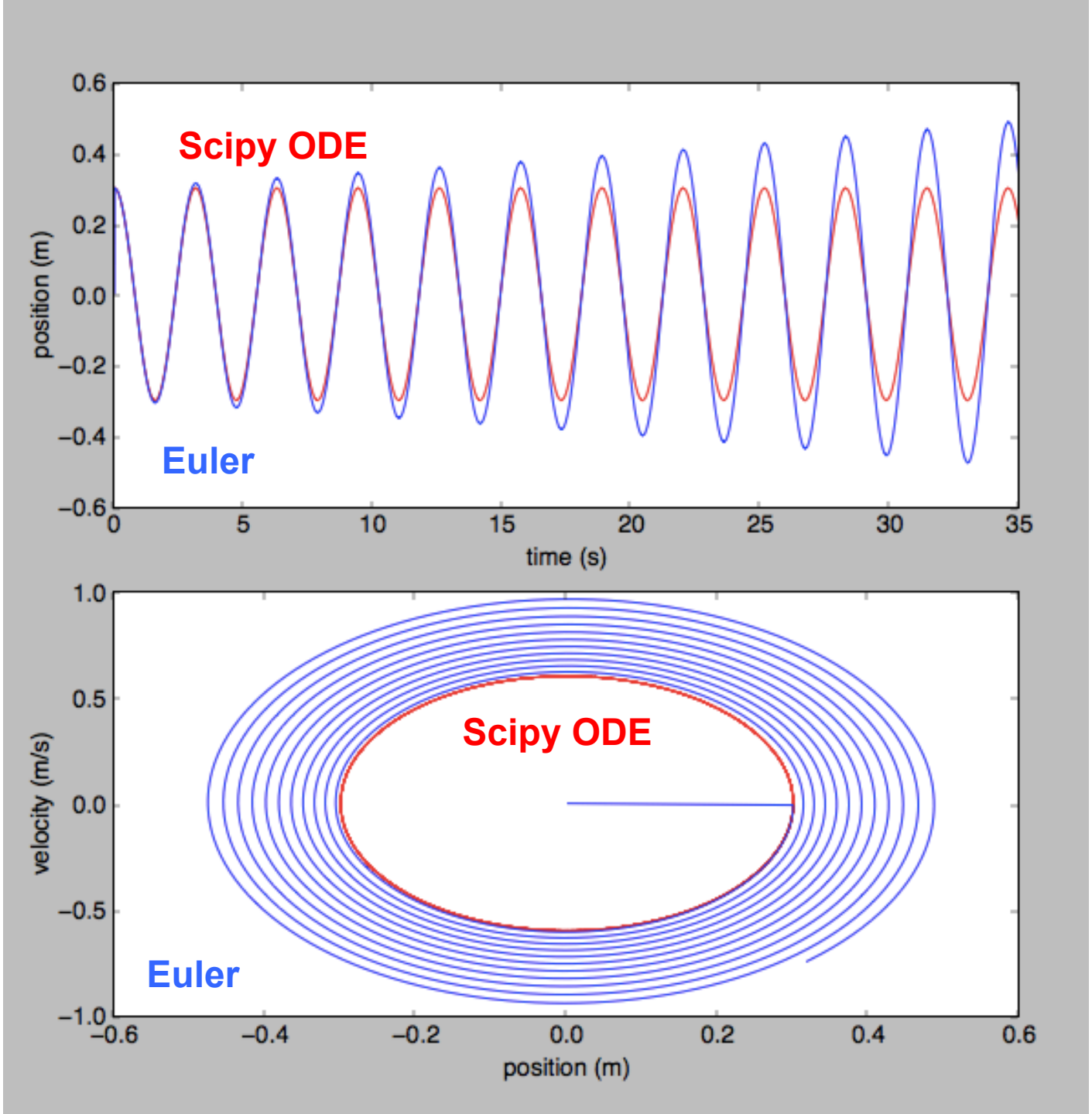
Via python

Returns 2D array

So split into x and v

```
pyplot.figure(figsize=(8,8))
pyplot.subplot(211)
pyplot.plot(ts,xs_ode,color='red')
pyplot.plot(ts,xs_eul,color='blue')
pyplot.xlabel('time (s)')
pyplot.ylabel('position (m)')

pyplot.subplot(212)
pyplot.plot(xs_ode,vs_ode,color='red')
pyplot.plot(xs_eul,vs_eul,color='blue')
pyplot.xlabel('position (m)')
pyplot.ylabel('velocity (m/s)')

pyplot.show()
```

Compare with plot

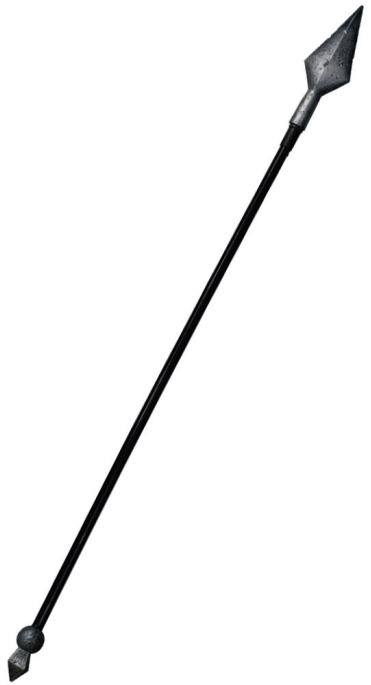

Scipy ODE
Euler
position (m)
time (s)
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
5
10
15
20
25
30
35
Scipy ODE
Euler
velocity (m/s)
position (m)
1.0
0.5
0.0
−0.5
−1.0
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6

Ballistics assessment

Ballistics throughout history

Pre-historic Physics

- “I’ve got a spear”
- “How fast do I have to throw it, and at what angle, for it to hit my prey?”



Ballistics throughout history

Modern Physics

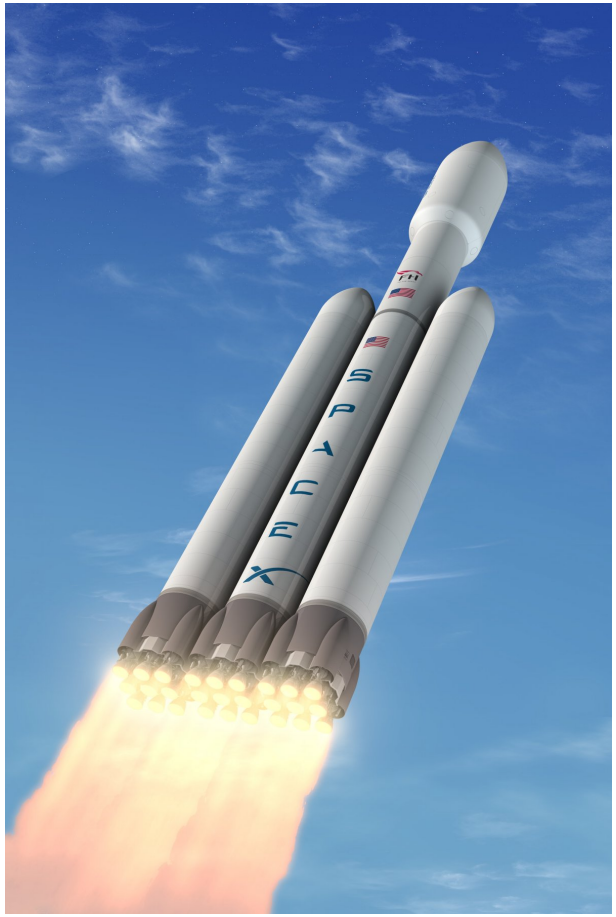
- “I’ve got Vickers and Armstrong 38.1cm gun. The wind speed is such, the air pressure is such and the temperature is such”

- “What azimuth elevation should I fire at to hit that ship 22 miles away?”

Bateria de Cenzias, Cartagena, Spain

Ballistics throughout history

- Exciting Physics!
  - I've got a rocket ship
  - How do I get to the Moon?



How to answer the question...

- Write a differential equation for the forces on the projectile

- Solve it – numerically for realistic problems

Make some assumptions

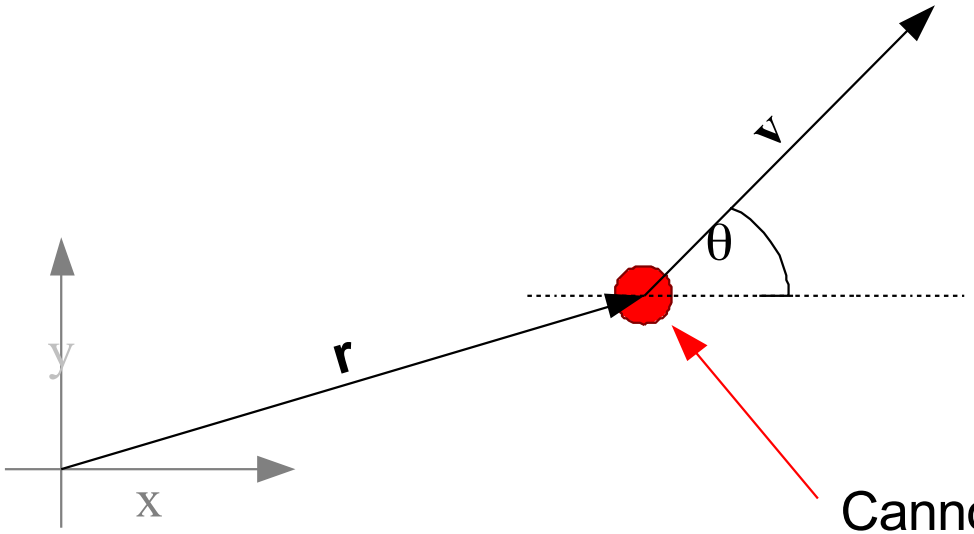
- Height of projectile small compared to radius of earth (mg not GMm/r^2)

- Air pressure invariant of projectile height

Assumptions

- It is standard to make simplifying assumptions when making a model
- Remember *GIGO – Garbage In, Garbage Out*
- Check your assumptions
  - especially that they remain valid as your model grows in complexity
- Document your assumptions

Formulate the problem



Cannonball:

m	mass
r	Position = (x, y)
v	Velocity = (v_x, v_y)

Forces

$$F_{grav} = \begin{pmatrix} F_{gx} \\ F_{gy} \end{pmatrix} = -mg \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$F_{drag} = -\kappa \rho A v \begin{pmatrix} v_x \\ v_y \end{pmatrix}$$

$$F = F_{grav} + F_{drag}$$

$$v^2 = \left(v_x^2 + v_y^2 \right)$$

- κ drag coefficient
- ρ air density
- A cross section area
- v $(v_x^2+v_y^2)^{.5}$

Differential Equations

2nd order

$$\frac{d^2x}{dt^2} = F_x / m$$

$$\frac{d^2y}{dt^2} = F_y / m$$

Turn into first order DEQs by introducing velocity

Differential Equations

2nd order

$$\frac{d^2x}{dt^2} = F_x / m$$

$$\frac{d^2y}{dt^2} = F_y / m$$

1st order

$$\frac{dv_x}{dt} = F_x / m$$

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

$$\frac{dx}{dt} = v_x$$

$$\frac{dy}{dt} = v_y$$

Substitute in the forces

$$\frac{dv_x}{dt} = \left(-\kappa\rho A v v_x\right)/m$$

$$\frac{dv_y}{dt} = \left(-\kappa\rho A v v_y - mg\right)/m$$

$$\frac{dx}{dt} = v_x$$

$$\frac{dy}{dt} = v_y$$