Dynamics

Intro from Foundation

PHYS 246 class 2

https://jnoronhahostler.github.io/IntroductionToComputationalPhysics/intro.html

Announcements/notes

```
from google.colab import drive
drive.mount('/content/drive')
!cp /content/drive/MyDrive/Colab\ Notebooks/Dynamics.ipynb ./
!jupyter nbconvert --to HTML "Dynamics.ipynb"
```

- First assignment is due tonight on gradescope
- Check new PDF and .ipynb guide on the website

(if you already submitted and the PDF and .ipynb are clear and legible, you are ok)

- Note that there is not much partial credit. Grade will be based on number of correct entries. A few correct things are better than a lot of incorrect things.
- You WILL be graded on good coding practices!
- Related to above, make sure that the first part of your assignment is correct before moving on -- the end depends on the beginning.

Emails

- Make sure Phys 246 is in the title
- Always add* Surkhab and Maxwell
- Nights/weekends I most likely won't be able to respond
- Grading changes on assignments will be always discussed amongst the 3 of us

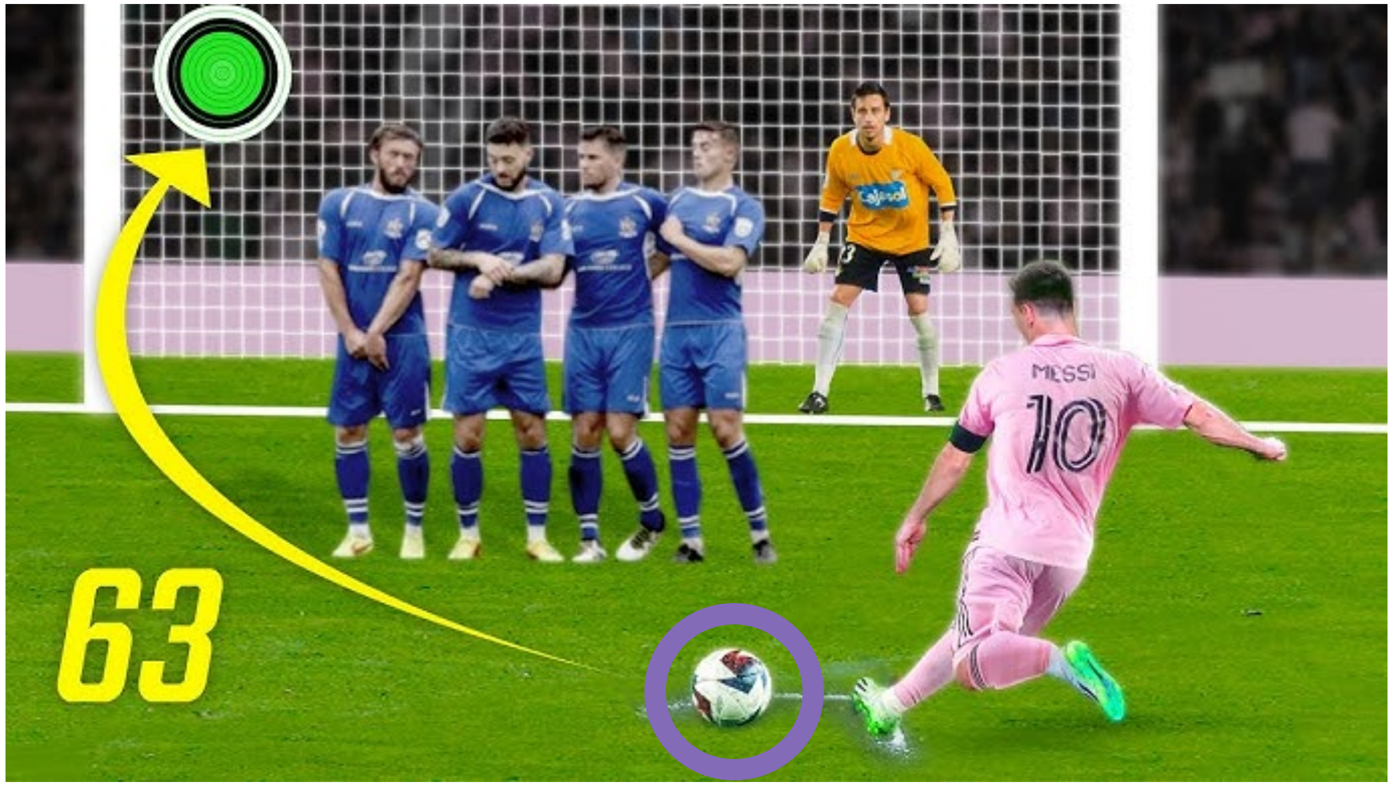
* Exceptions: letter of rec requests, research requests, or personal issues that may affect grades

Dynamics

How do objects move?

- Let's start with something easy like a "point-like particle".

- We will need the particle's initial conditions



- Equations to describe the particles movement in time
- A numerical method to solve these initial conditions+equations

Newtonian equations of motion

How do we solve them numerically?

$$x(t) = x_0 + \frac{1}{2}\left[v(t) + v_0\right] t$$

$$v(t) = v_0 + at$$

$$F = ma \rightarrow a = \frac{F}{m}$$

Getting the position over time

$$x(t_0) = 0, \quad v(t_0) = 55\, m/s \qquad t_{i+1} = t_i + \Delta t$$

Slope @ t_i

$$x(t_{i+1}) \sim x(t_i) + v(t_i)\Delta t$$

Next step

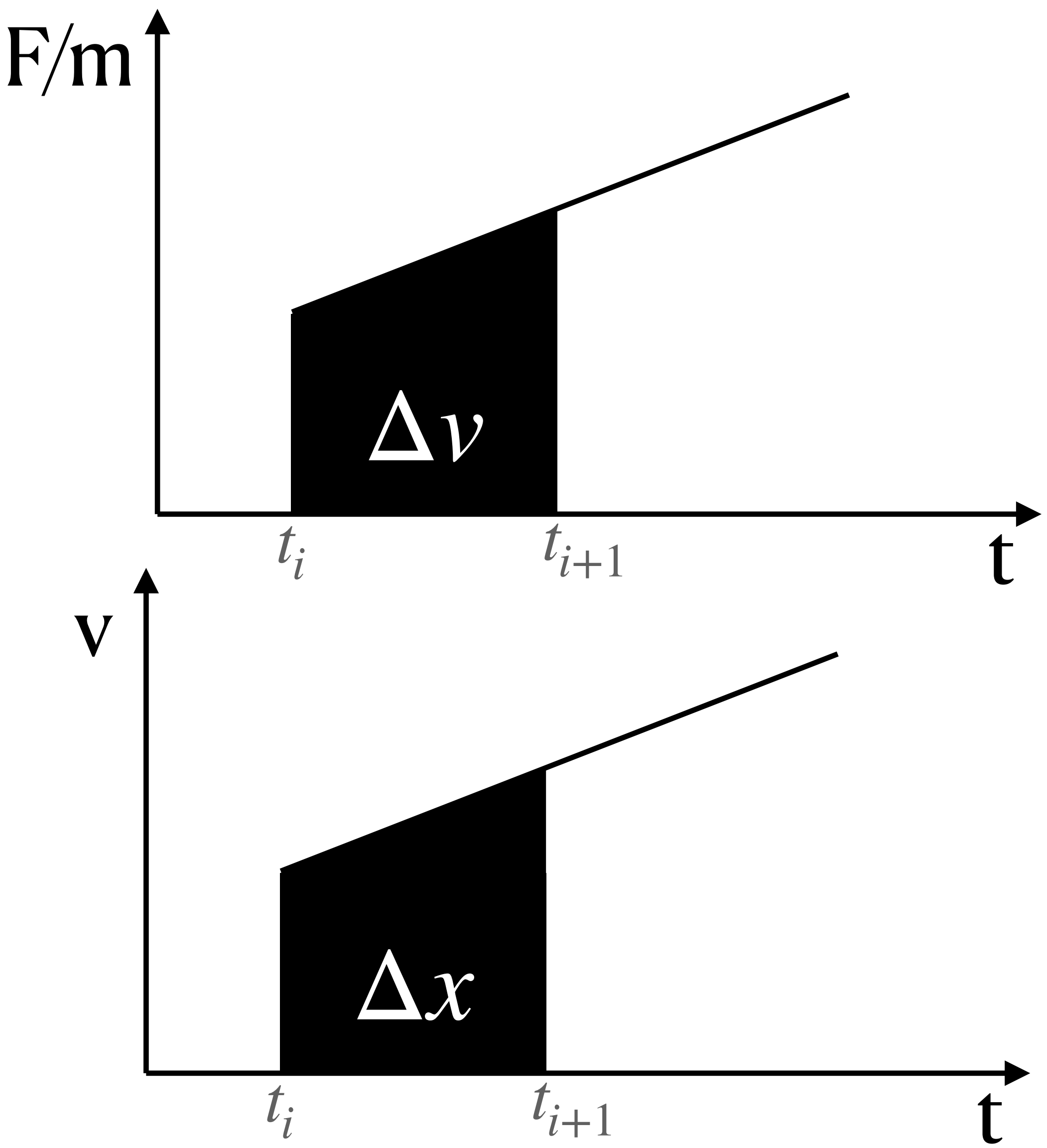
Step size

Current step

What should we use here? How do we do this numerically?

What is $v(t)$?

1D Dynamics by numerical integration



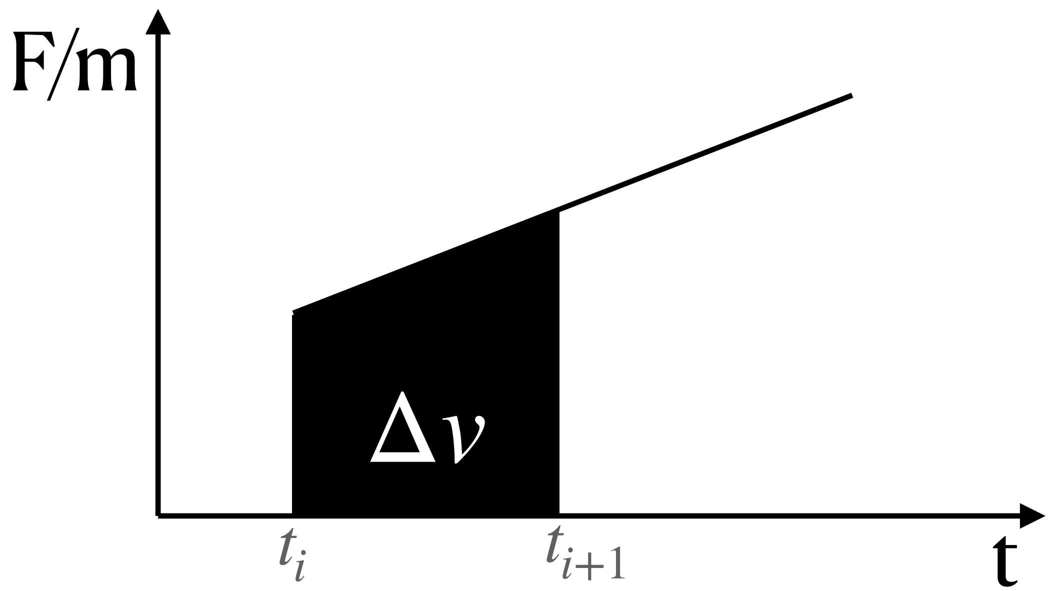
Suppose we know the position and velocity at some time t_i, and want to know these quantities at a future time t_{i+1}

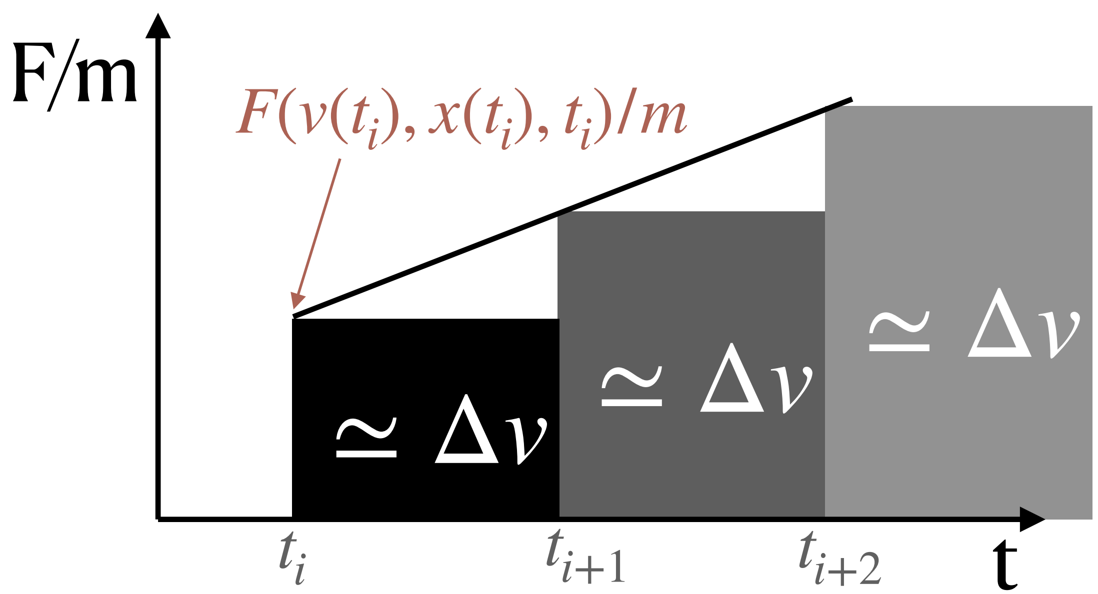
Newton says: $F = m\dfrac{dv}{dt}$

So $v(t_{i+1}) - v(t_i) = \displaystyle\int_{t_i}^{t_{i+1}} \frac{F}{m} dt$

If we can compute this integral we know what the velocity will be. Similar for position.

Euler integration





$\Delta t = t_{i+1} - t_i$

We know the position and velocity at some time t_i, and want to know these quantities at a future time t_{i+1}

If we can compute $F(v, x, t)$, then we can approximate this integral.

Error is proportional to Δt:

$$F = F(t_i) + \frac{dF(t_i)}{dt}\Delta t + \frac{1}{2}\frac{d^2F(t_i)}{dt^2}(\Delta t)^2 + \ldots$$

We're assuming a non-relativistic system, let $m = const$

Euler algorithm

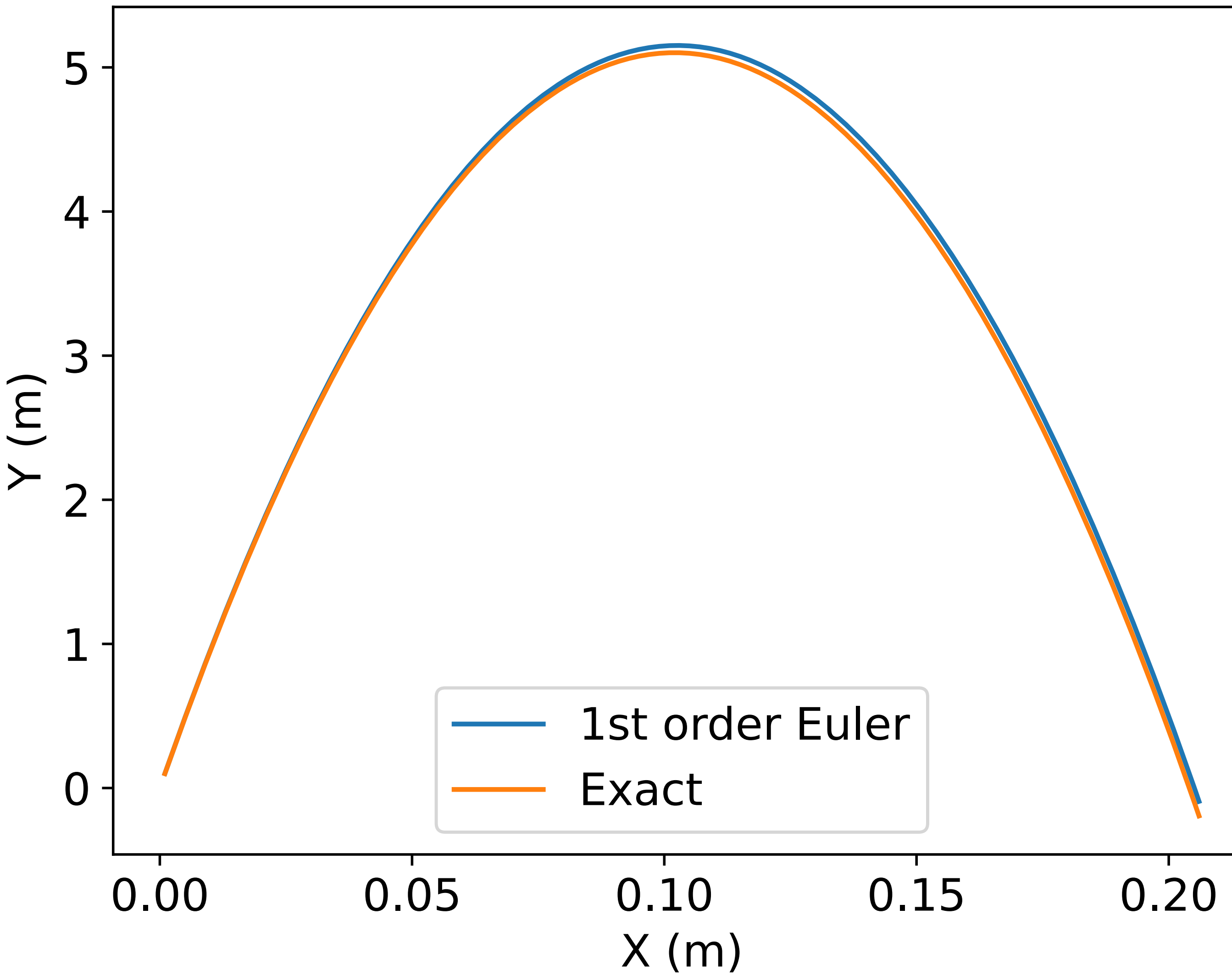
Start with $x(t_0)$, $v(t_0)$ (may be vectors)

First find velocity

$$v(t_{i+1}) = v(t_i) + \frac{F(v(t_i), x(t_i), t_i)}{m}\Delta t$$

Then position

$$x(t_{i+1}) = x(t_i) + v(t_i)\Delta t$$

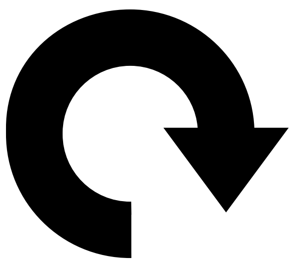


Iterate until you go for long enough..

How do we practically implement iteration?

loops, arrays, lists - oh my!

Loops



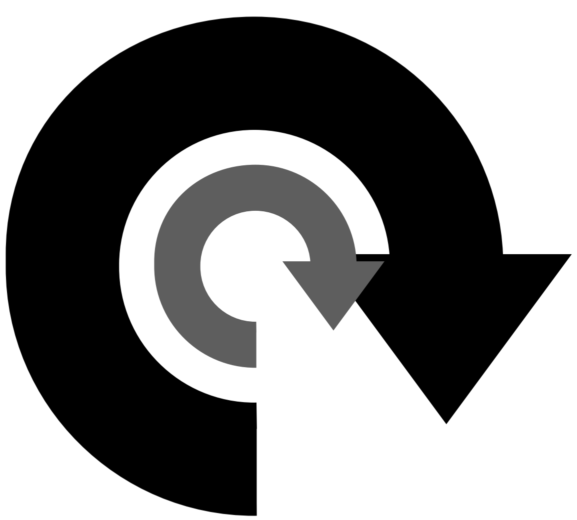
for, while

Arrays

Mydata = [3,”four”,5,6.78492]

Different data type, slower, more memory

Nested loops



Numpy lists

Import numpy as np
list1 = np.array([3,4,5,6])

Typical single data type, faster, less memory, element-wise operations

Notes on Python: lists, dictionaries, and numpy arrays

Python lists: [a,b,c]

- a,b, c can be anything
- "+" means append
- Built-in to python

Numpy arrays

- a,b, c must all be the same type
- "+" means element-wise add
- From the numpy library
- Can be quite fast

Python dictionaries: [a:x,b:y,c:z]

- look up tables
- a,b,c have to be immutable (strings, numbers, tuples)
- x,y,z can be anything
- No '+' operator

Similar to a structure in c++

Force model

Need to compute $F(\mathbf{v}, \mathbf{x}, t)$

Gravity plus air resistance:

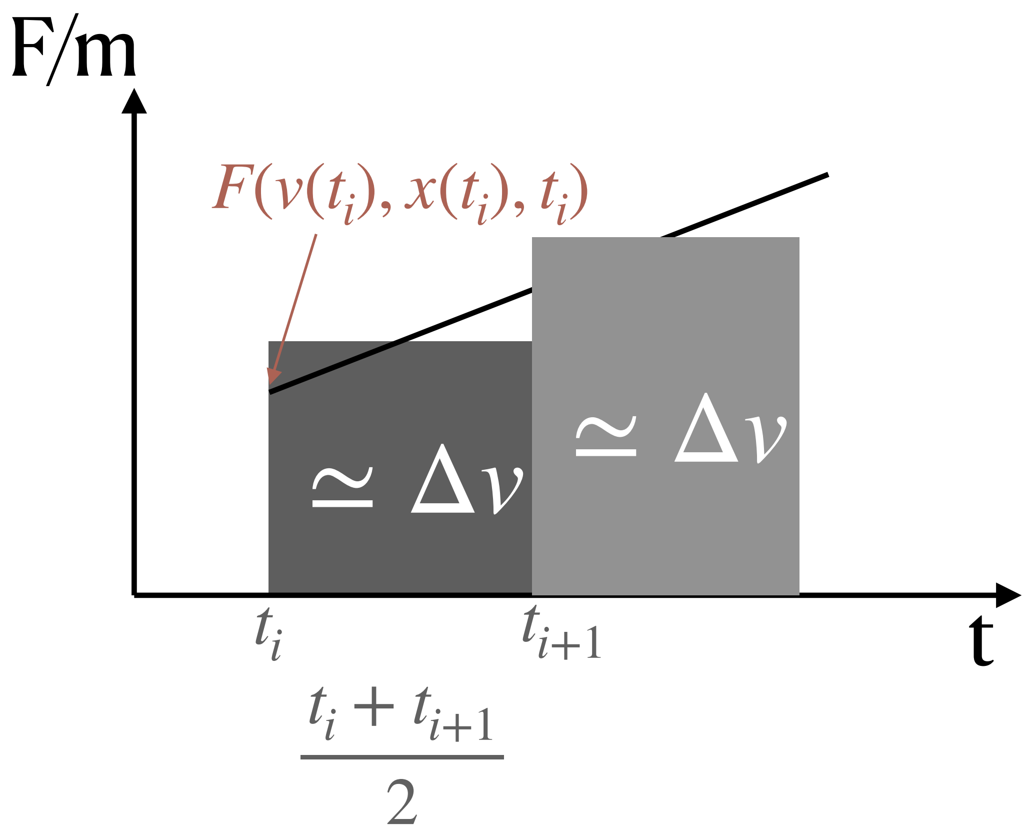
$$F(\mathbf{v}, \mathbf{x}, t) = -b\mathbf{v} - mg\hat{y}$$

This is why computational approaches are so useful--you can throw any force model at the simulator and it basically is the same effort.

Testing your code: discussion

How can we be confident that our math and implementation are correct?

Midpoint method (refined Euler)



We know the position and velocity at some time t_i, and want to know these quantities at a future time t_{i+1}

If we can compute $F(v, x, t)$, then we can approximate this integral.

Error is proportional to $(\Delta t)^2$:

$$F = F(t_{mid}) + \frac{dF(t_{mid})}{dt}\Delta t + \frac{1}{2}\frac{d^2F(t_{mid})}{dt^2}(\Delta t)^2 + \dots$$

Midpoint method

In practice

Calculate the halfway time: $t_{n+1/2} = t_n + \frac{\Delta t}{2}$

Use Taylor series for the midpoint (like the Euler method):

$$x_{n+1/2}\left(t_n + \frac{\Delta t}{2}\right) \sim x(t_n) + \frac{\Delta t}{2} v(t_n, x(t_n))$$

Then, the full step-size is:

$$x(t_{n+1}) = x_n + v\left(t_{n+1/2}, x_{n+1/2}\right) \Delta t$$

Modular code

Avoid Hardcoding Woes!

```
[3] def v(t):
      return 55


    def f(x,t):
      x1=x+v(t)*0.1
      t1=t+0.1
      return x1,t1

    v(0.4)
    f(1,2)
```

Hard coded time steps

```
(6.5, 2.1)
```

```
[4] vavg=55 #m/s
    deltat=0.1 #s

    #constant velocity
    def v(t):
      return vavg


    # calculate the position for  constant velocity
    def f(x,t):
      x1=x+v(t)*deltat
      t1=t+deltat
      return x1,t1

    f(1,2) # position at time=2.1 sec
```

```
(6.5, 2.1)
```

Constants defined up top, functions not hard coded

Functions passed as variables

```
vavgMessi=55 #m/s
vavgPele=60 #m/s
deltat=0.1 #s

#constant velocity function for Messi
def vMessi(t):
  return vavgMessi

#constant velocity function for Pele
def vPele(t):
  return vavgPele


# calculate the position for  constant velocity,
#but the velocity can be a function!
def f(v,x,t):
  x1=x+v(t)*deltat
  t1=t+deltat
  print("position=",x1)
  return x1,t1

f(vMessi,1,2) # position at time=2.1 sec

f(vPele,1,2) # position at time=2.1 sec
```

```
position= 6.5
position= 7.0
(7.0, 2.1)
```

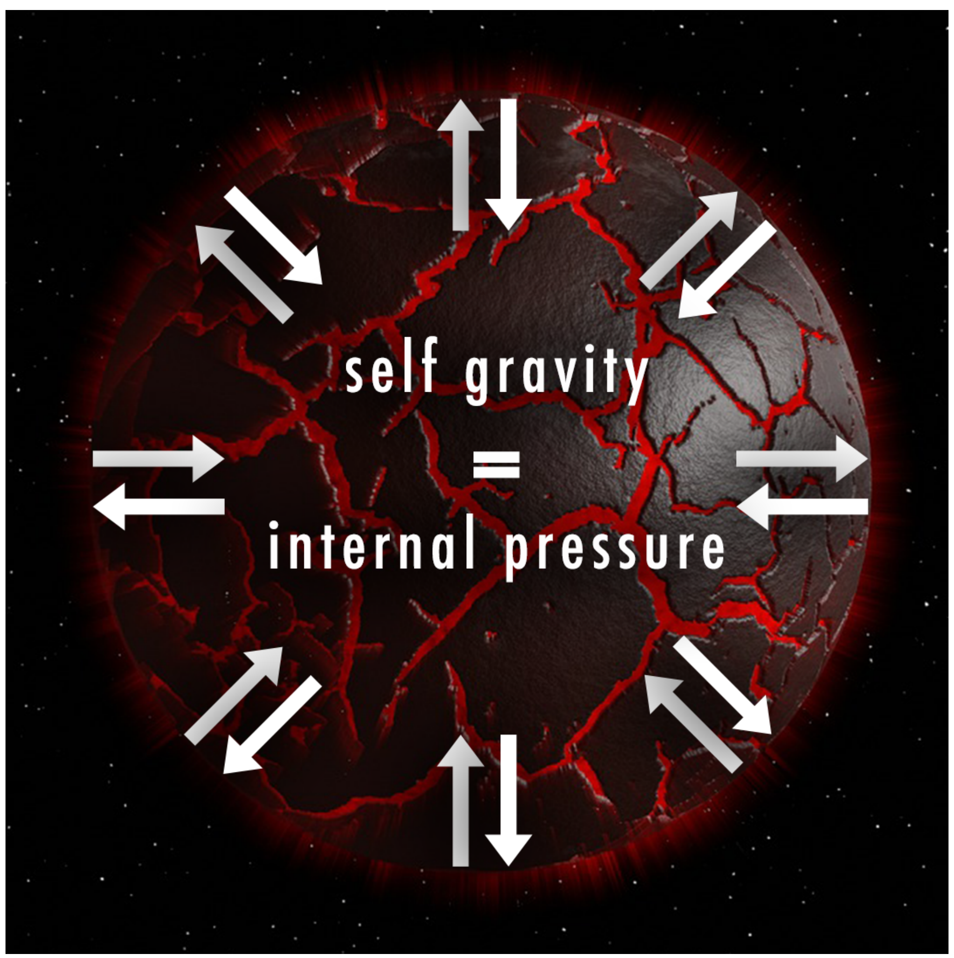
Quantifying error (log-log plots)

Typically we say that the error is proportional to Δt^n, but that's the 'leading-order' term, only valid for small enough Δt.

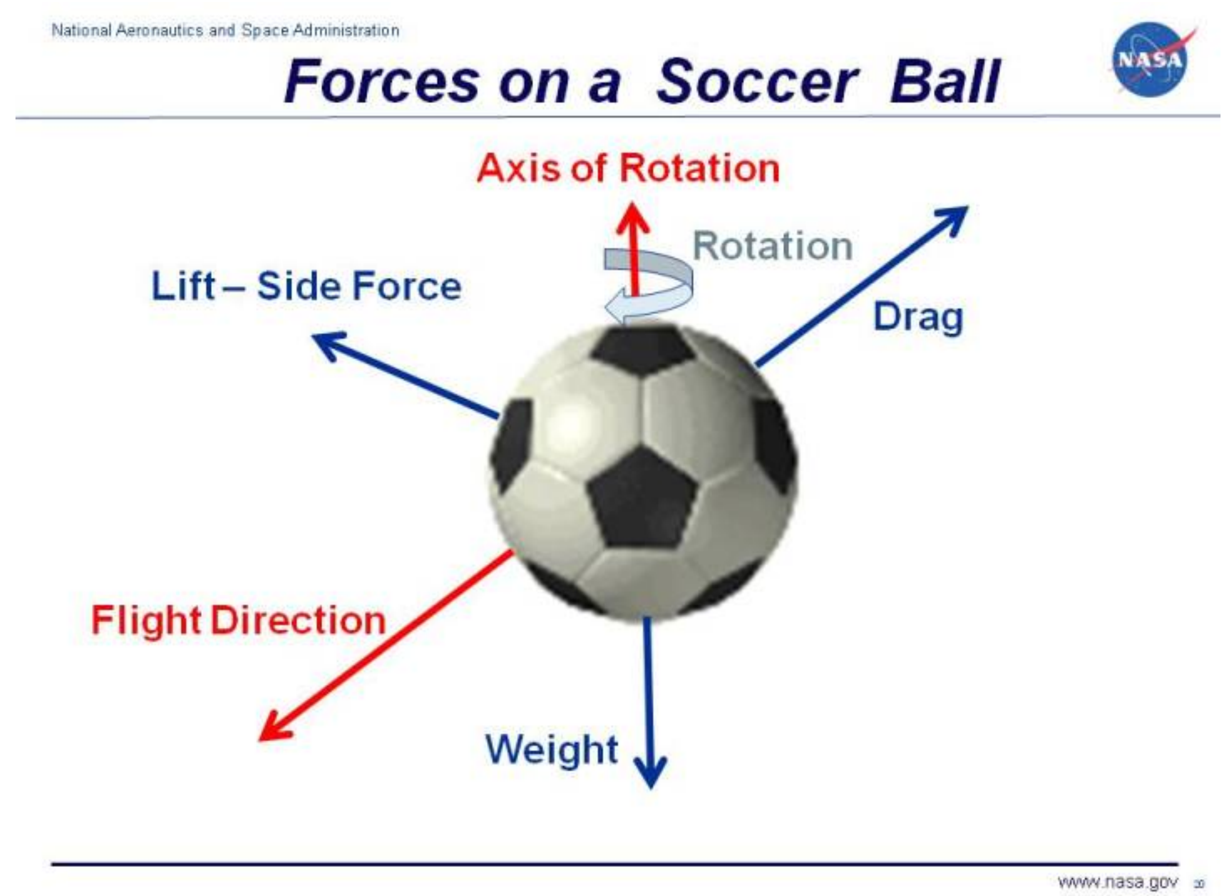
Where are these methods used?

Numerical integration is everywhere!

Hydrostatic equilibrium of stars



Physics of sports



Warning!!

2D dynamics

- Make sure your 2D dynamics code is working!
- You WILL need this for next week's assignment…