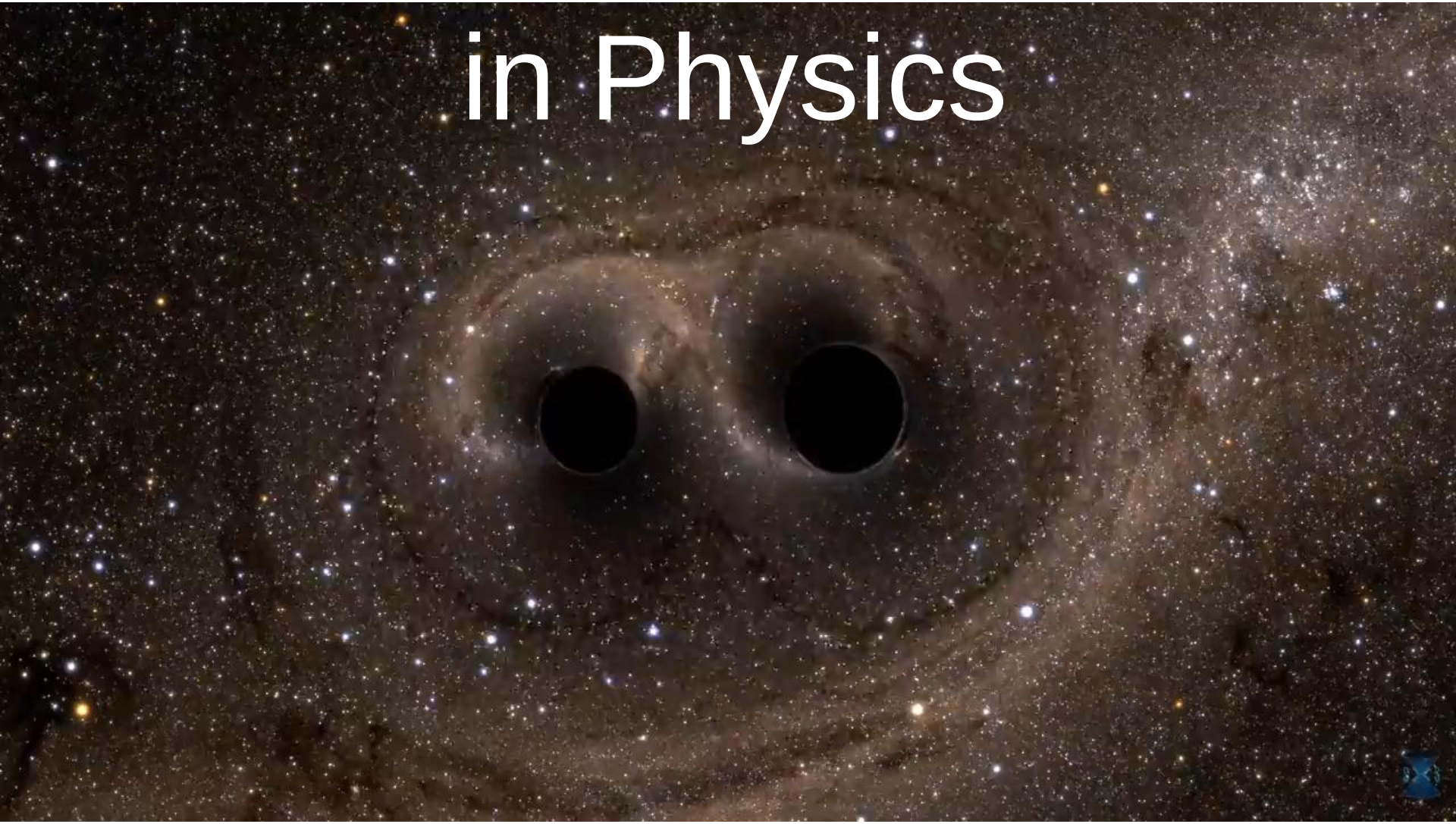
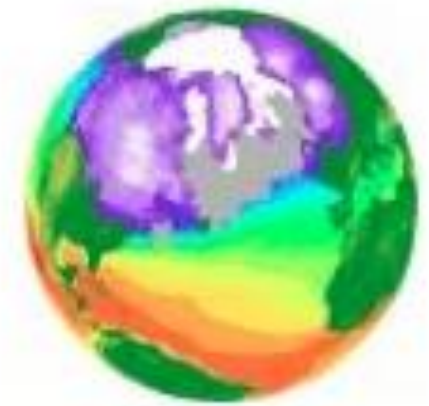
Scientific Computing in Physics

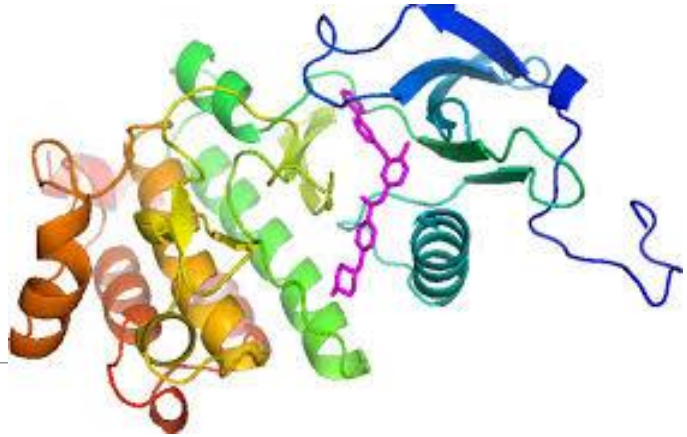


August 14th
SUSC, Physics Department

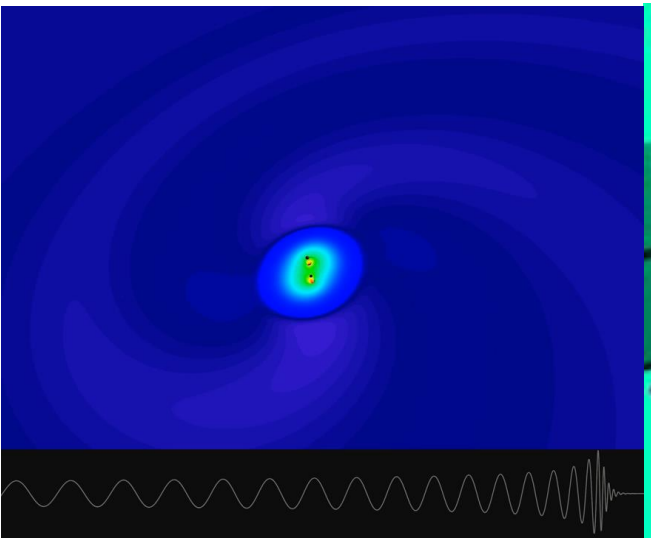
Charles Woodford
Canadian Institute of Theoretical Astrophysics
Physics UNIVERSITY OF TORONTO
CITA ICAT

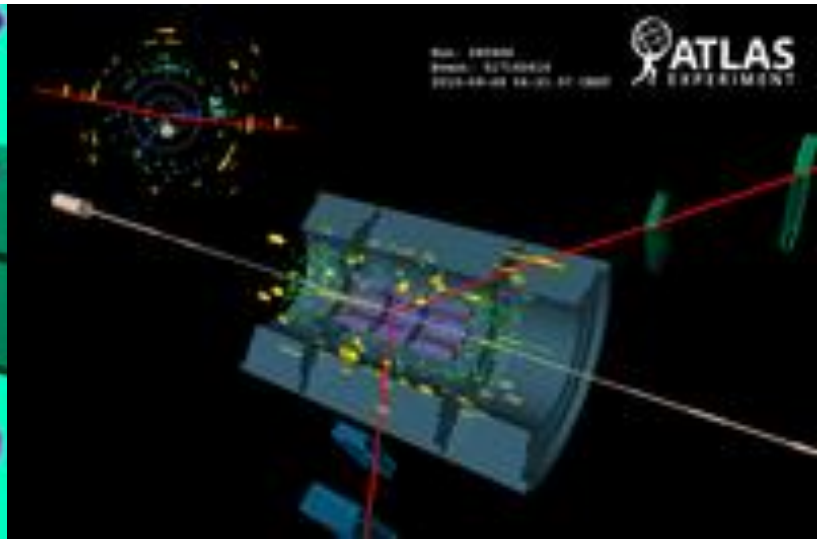


What is Scientific Computing?



Using computers to simulate real life scenarios, in order to forecast, estimate, or constrain real-world problems.

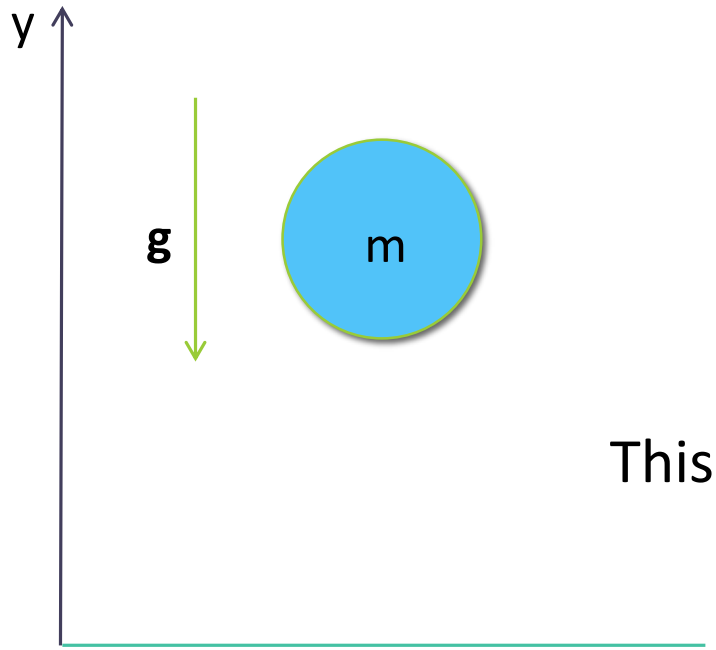




Why Physics Uses Computing

- Some physics problems have simple equations that can be solved "analytically" (i.e. with pen and paper). These usually involve simple forces and few objects being acted upon:

- e.g.: ball under the influence of gravity: $F = ma$
$= -mg$



Solution:

velocity: $v_f = v_i - g(t_f - t_i)$

position: $y_f = y_i + v_i(t_f - t_i) - \frac{1}{2}g(t_f - t_i)^2$

This is a very IDEALIZED problem: Why?

Why Physics Uses Computing

- Forces can be much more complicated:
  - e.g. fluid in Earth's core:

$$\vec{F} = -\nabla P + \rho \vec{g} + \mu \nabla^2 \vec{v} + \frac{1}{3}\nabla(\nabla \cdot \vec{v}) - 2\rho \vec{\Omega} \times \vec{v}$$

- System may include many components, each affected by different forces
  - e.g. Binary Neutron star merger, detected by LIGO and simulated at Goddard

- These problems can't be solved with pen & paper - Instead we need the rapid calculation power and large memory of the biggest computers available

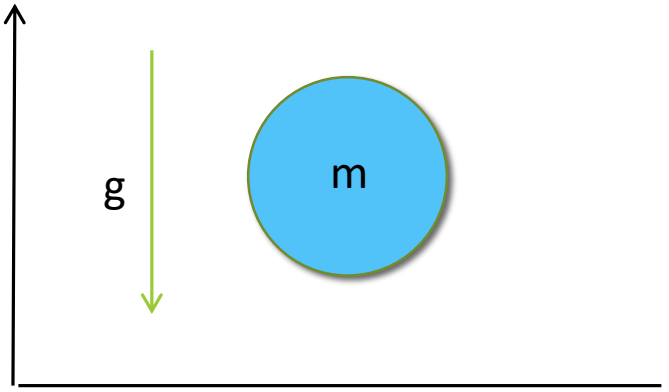
How Does a Computer Solve Physics Problems?

- Computers are dumb, but very good at doing basic math fast!
- What we need to do:
  1. Find the equations that describe the physics
  2. Rewrite them in terms of simple math operations
     i. There may be MANY of these operations but that's okay because computers are fast! The fastest supercomputer in the world (Top 500 list) is Summit (or OLCF-4) – can do 143.5 PFLOPS (11/18)
  3. Tell the computer to do the mathematical operations

How Does a Computer Solve Physics Problems?

- Example: Ball under the influence of gravity:

$$F = ma$$
$$= -mg$$

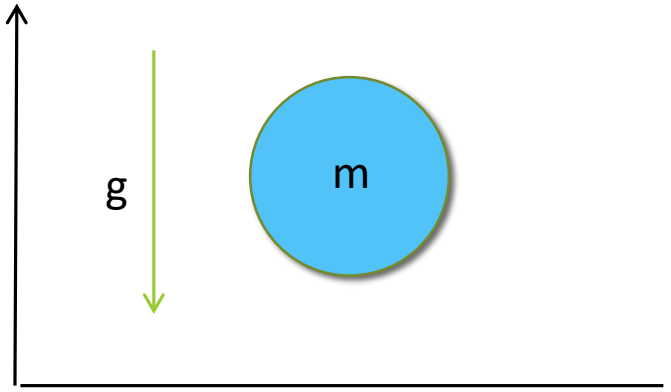


- Lets pretend we don't know the solution (since we don't for other problems). How do we get the computer to calculate the solution?

How Does a Computer Solve Physics Problems?

- Example: Ball under the influence of gravity:

$$F = ma$$
$$= -mg$$



- Lets pretend we don't know the solution (since we won't for other problems). How do we get the computer to calculate the solution?

1. Determine the equations governing the physics we are studying: (these equations are true as long as Δt is very very small)

Newton's 2nd Law:

$$F = ma \qquad a = \frac{\Delta v}{\Delta t}$$
$$= -mg \qquad v = \frac{\Delta y}{\Delta t}$$
$$a = \frac{F}{m}$$

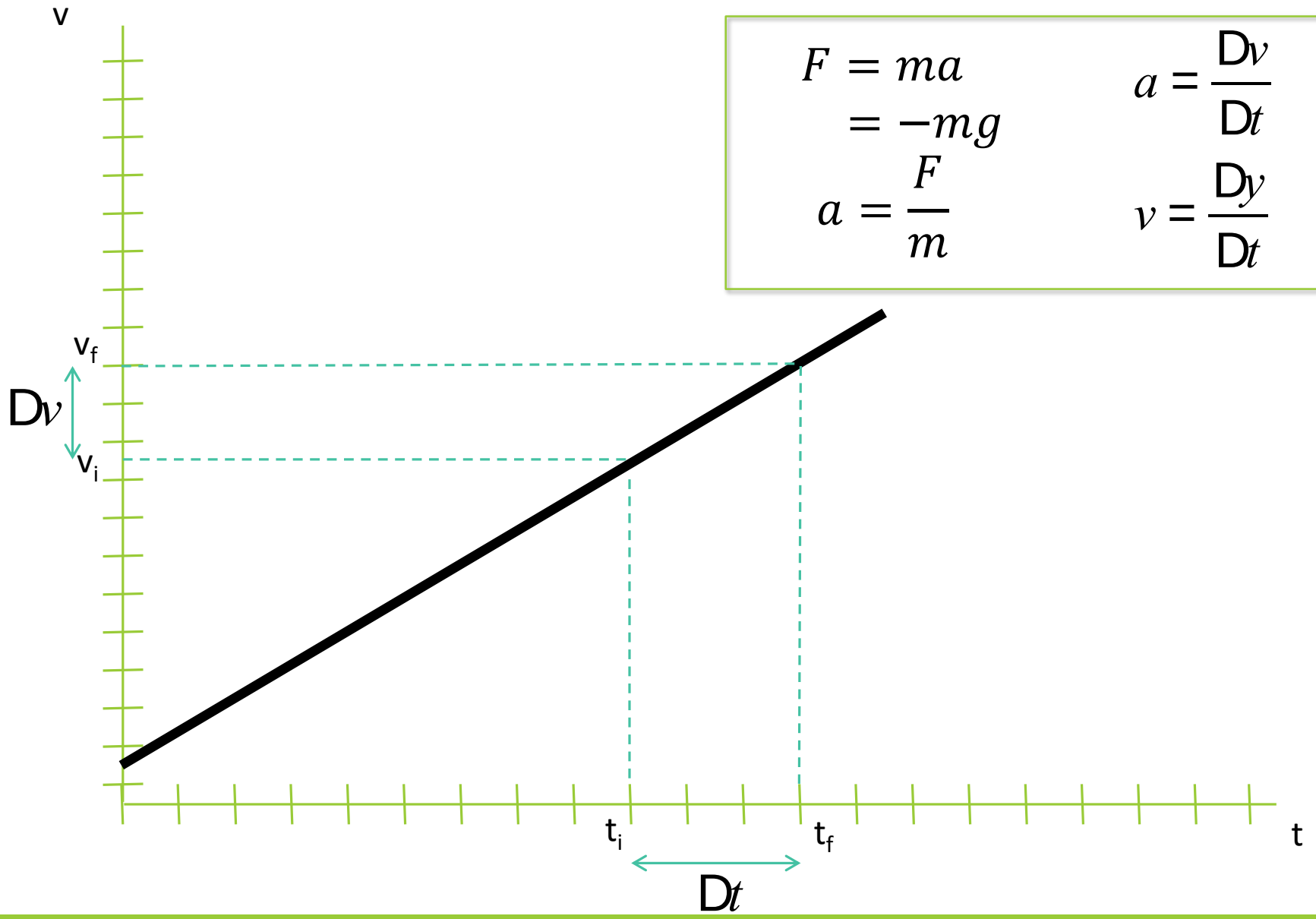
Δt : small interval of time

Δv: change in velocity over Δt

Δy: change in position over Δt

How Does a Computer Solve Physics Problems?

$$F = ma \qquad a = \frac{\Delta v}{\Delta t}$$
$$= -mg$$
$$a = \frac{F}{m} \qquad v = \frac{\Delta y}{\Delta t}$$



How Does a Computer Solve Physics Problems?

2. Rewrite them in terms of simple mathematical operations
   ii. Rewrite our equations for a(t) and v(t) using:

$$\Delta v = v_f - v_i$$

$$\Delta y = y_f - y_i$$

$$F = ma \qquad a = \frac{\Delta v}{\Delta t}$$
$$= -mg \qquad v = \frac{\Delta y}{\Delta t}$$
$$a = \frac{F}{m}$$

We get:

$$v_f = v_i - g\Delta t$$
$$y_f = y_i + v_f \Delta t$$

These are now equations using basic math operations on how to update the velocity and position after each time step.

Notice these aren't the analytic solutions:

$$v_f = v_i - g(t_f - t_i)$$
$$y_f = y_i + v_i(t_f - t_i) - \frac{1}{2}g(t_f - t_i)^2$$

How Does a Computer Solve Physics Problems?

3. Tell the computer to do the math!

$$v_f = v_i - g\Delta t$$
$$y_f = y_i + v_f \Delta t$$

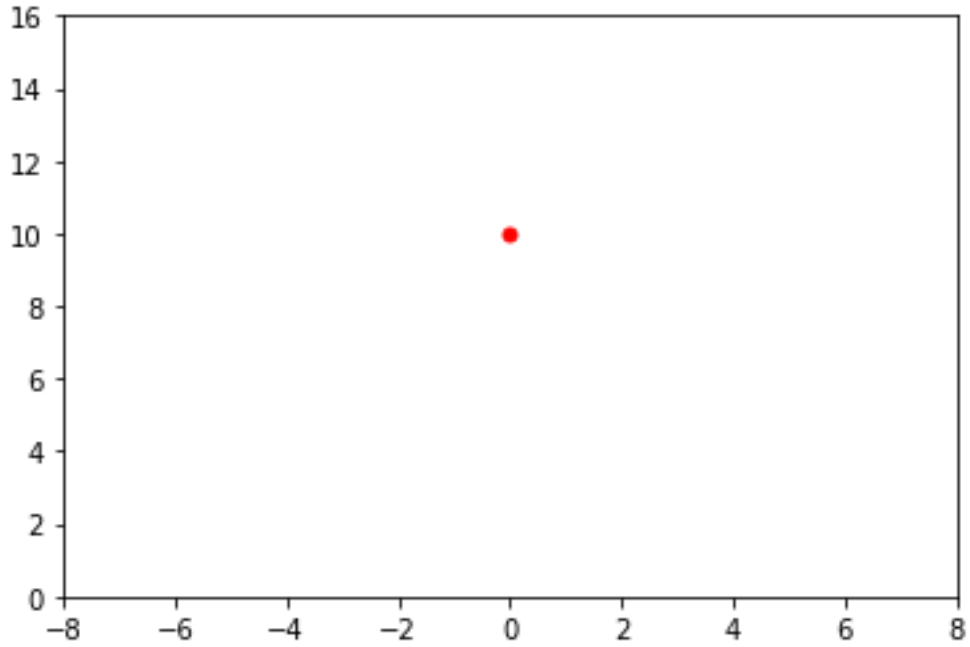
So our *algorithm* should be:

1. Start with t=0, v_i, y_i defined at t=0.
2. Define Δt (up to you!)
3. Take 1 step in time ($1\Delta t$)
4. Calculate new v_f and y_f from formulas above
5. Repeat steps 3 – 5 for however many time steps we need

Lets use the python programming language to do this!

Free Fall Code

- Click on the “Spyder” icon on the desktop
- Under the “File” menu open “freefall.py” on the desktop
- Save as “freefall_v1.py” (in case you need to start over)
- Close the help window, so you only have the code and iPython console screens.
- Run your program using “F5” or the big green arrow in the tool bar.
- You should get the following in the iPython console:



Free Fall Code

- Lets look at the program:
  - Any line starting with # is a comment, the code does not read it
  - This line defines the position of the ball in 2D:

```
ball = np.array([0.0,10.0])
```

x-position y-position

  - These lines define the ball's initial velocity and acceleration due to gravity:

```
ball_vel=np.array[(0.0,0.0)]
g=9.8
```

Since that is the only part of the code active right now, no time steps are taken so the ball never updates its velocity or position and stays at the same location (that's why nothing happens).

Free Fall Code

- Okay, so lets make something happen:
  1. Start with t=0
  2. Define Δt
     - In the code, remove the # sign in front of the following lines:

```
t=0.0
dt = 0.01
```

  3. Take 1 step in time (1 Δt)
  4. Calculate new v_f and y_f

- In the code, remove the # sign in front of the following lines (make sure to keep the indentation):

```
t=t+dt
ball_vel[1] = ball_vel[1] - g*dt
ball[1] = ball[1] + ball_vel[1]*dt
```

$\longleftrightarrow$

$$v_f = v_i - g\Delta t$$

$$y_f = y_i + v_f \Delta t$$

Free Fall Code

- Repeat steps 3 – 4 for however many time steps we need
  - The best way to do this with a computer is using a “loop”. A “while” loop repeats all the commands indented in the lines after the while statement as long as some condition is met. There are other kinds of loops you can try in your own codes, like “for” loops.
  - Remove the # sign in front of the line: `while ball[1] >= 0.0` what does this mean?
  - This starts our loop. The commands indented afterwards:

```
t=t+dt
ball_vel[1] = ball_vel[1] - g*dt
ball[1] = ball[1] + ball_vel[1]*dt
```

  will be calculated each time through the loop.

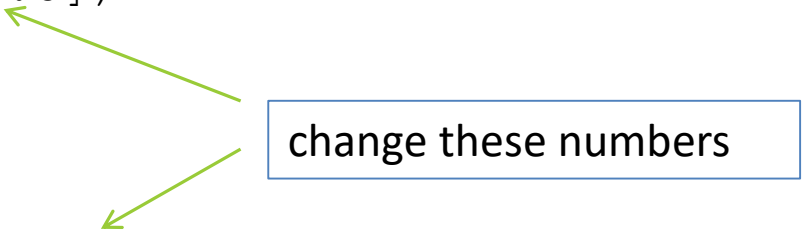
  - The loop will keep repeating until the position of the ball < 0 (the ball hits the ground).
  - Uncomment the rest of the loop and the last line to include graphic output. Run your program.

Free Fall Code

- Try running your program with different initial y velocities and different initial y positions by changing the following lines:

```
ball = np.array([0.0,10.0])
```

```
ball_vel = np.array([0.0,0.0])
```



- What changes for your runs? Is there a difference for positive and negative initial velocity values?

Projectile Motion Code

- Open the program “projectile.py”. Save it as “projectile_v1.py”.
- Notice that currently it is almost identical to the free fall code. If you run it you will see it is the same.
- Projectile motion also allows initial velocities in the “x” (horizontal) direction so instead of dropping the ball we will throw the ball.
- Your mission: Modify some of the lines and add lines in this code to give the ball an initial velocity in the x direction and to update the x components of the position and velocity in time.
- ***If you cannot see the ball on the plots, it means that it’s position is outside of the frame boundary. Change the values in the lines to larger/smaller values.

```
plt.axis([-16,16,0,16])
```

x-axis min and max y-axis min and max

Projectile Motion Code

- Is there any acceleration in the x direction?

  No, gravity only works in the y direction. So in the x direction:

$$v_f = v_i$$
$$x_f = x_i + v_f \Delta t$$

- Do we ever need to update the velocity in the x direction?

  No, its always the same. It's a constant = initial x velocity.

- Do we need to update the position in the x direction?

  Yes! According to the above formula.

Projectile Motion Code

- Here is the line in the code for the y position.

```
ball[1] = ball[1] + ball_vel[1]*dt
```

- Create a very similar line for the "x" position (replace all the 1's with 0's) and add it to the code right after the above line. Don't forget to keep the line indented!
- Run your code. Did you get projectile motion?

- Try varying your initial x and y velocities to see what you get.

- You have now created a projectile motion code!

Recap:

- We defined the laws of physics for our object
- We rewrote the laws into simple mathematical operations
- We made the computer do the operations over and over again to update the motion of the object.

Projectile Motion Code

- Our code works, but is it accurate?
- In this case, we know the analytical solution, so we can compare the results! How can we compare our code against the analytical solution?

$$v_{f,y} = v_{i,y} - g(t_f - t_i) \qquad v_{f,x} = v_{i,x} = v_x$$

$$y_f = y_i + v_i(t_f - t_i) - \frac{1}{2}g(t_f - t_i)^2 \qquad x_f = x_i + v_x(t_f - t_i)$$

Brainstorm:

- Print results from code to screen (how?)
- Calculate analytical solution in the code
- Calculate differences and statistical measures in the code
- Do the differences change with different parameters (vi, yi, t, etc.)?

Is this accurate to real life?

Let’s test it out!

We’ll go outside and test using real projectiles!

Let’s compare our analytical solution to what we see in real life. Remember that this is all part of science: theory (analytical solutions), computation, and experiment!

What do we need to consider when testing our projectiles? What’s realistic?

Resources:

1. Lynda online coding courses (www.lynda.com)
2. Ladies Learning Code (www.ladieslearningcode.com)
3. Enthought Canopy (FREE platform for using, creating, and running python code! – www.enthought.com/products/canopy/)
4. Python tutorials and walkthroughs (www.python.org)
5. Download and use Spyder yourself!
   i. Follow instructions on our wiki: http://compwiki.physics.utoronto.ca/ , in the Python 3 section
   ii. Download Anaconda 3 directly: https://www.continuum.io/downloads
6. Hatch (www.hatchcanada.com)

Feel free to contact me if you have any questions about this session!

cwoodford@black-holes.org

You can also find all these resources plus the code on my website: www.cita.utoronto.ca/~woodford