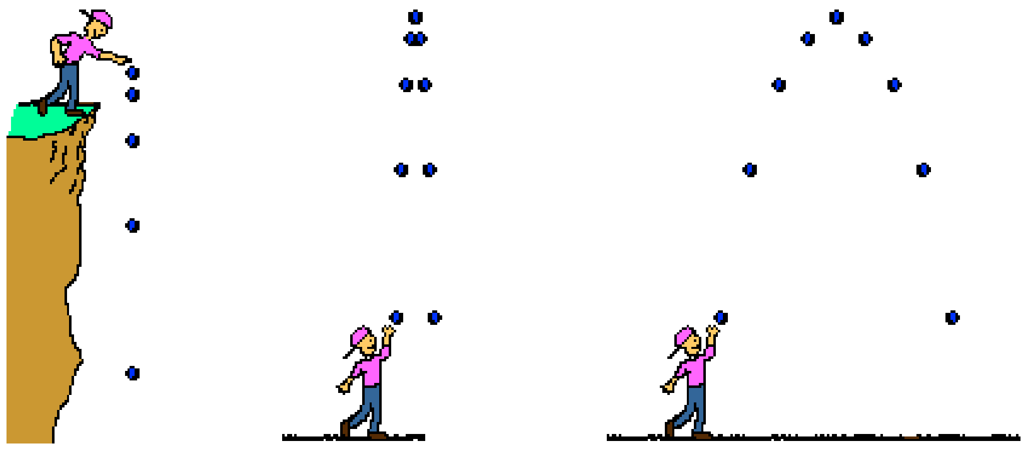
5

Doing Physics with Python

Introduction

In this worksheet we are going to do something a little different — we are going to try and solve some real-world problems in Physics! In particular, we are going to look as **projectile motion**. This is a part of Physics, which is interested in the movement of simple objects under the effect of gravity. Don't worry if you have not covered these in school yet, most of this is just common sense and please ask us if we cover anything that is confusing.



As before we have broken up the task in separate stages:

① **What are projectiles and what can I do with them?**

② **Our First Experiment: Dropping a ball**

③ **Our Second Experiment: Throwing Up (without the mess!)**

④ **Visualising our projectiles**

And in the next worksheet we will look at

① **One Dimension is Boring, Lets go 2D**

② **Generating graphs of the projectile movement.**

CoderDojo, Tramore, Waterford. (`kmurphy@wit.ie`)

This work is licensed under a Creative Commons "Attribution-NonCommercial-ShareAlike 3.0 Unported" license.

1 Who asked for Physics?

Projectile motion is the motion of an object thrown or projected into the air, subject to only effect of gravity. The object is called a **projectile**[1], and its path is called its **trajectory**. We are interested in
What do we know about moving objects? … we know that we can calculate how fast we are moving by dividing the distance we have travelled by the length of time taken. So we have

$$\text{velocity} = \frac{\text{distance}}{\text{time}} \qquad \Longleftrightarrow \qquad \text{distance} = \text{velocity} \times \text{time}$$

Lets think about this equation a bit to see if it make sense. It says:

- If velocity is larger (we are moving faster) then we will travel a larger distance. …… ✓
- If time taken is larger then the distance travelled is larger …………………………… ✓

Now since

$$\text{distance} = (\text{new position}) - (\text{old position})$$

we now have a formula that allows us to calculate the new position when given old position, velocity and time taken:

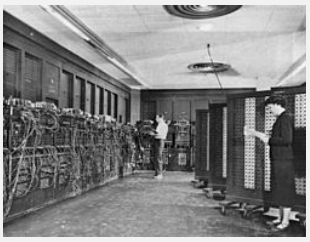
$$(\text{new position}) = (\text{old position}) + \text{velocity} \times \text{time} \tag{1}$$

OK, we got the above formula from using the fact that velocity is defined as the rate of change of position. Now think about **acceleration**. Acceleration is defined as the rate of change of velocity so using the same arguments as above we get formula

$$(\text{new velocity}) = (\text{old velocity}) + \text{acceleration} \times \text{time} \tag{2}$$

So now we have formulas, … computers love formulas, We are going to use them now to calculate the behaviour of objects.

A Little Bit of History …



ENIAC (Electronic Numerical Integrator And Computer) was one of the first general purpose computers. It was built during WW2 and completed in 1946. It was funded by the U.S. Army and was used to calculate ballistic tables for artillery shells, taking into account the effects of drag, wind and other factors influencing projectiles in flight.
ENIAC, unlike the computers of today was a colossal machine, weighing 30 tons, consuming 150 kilowatt of power and taking up 1800 square feet of floor space.
At the time it was proclaimed in the media as `a human brain'. It was very fragile and components needed frequent replacing. The first primary programmers of ENIAC were:

Kay McNulty

Jean Bartik

Betty Holberton

Marlyn Meltzer

Frances Spence

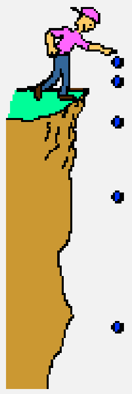
Ruth Teitelbaum

[1] If you can kick it, throw it or drop it then is is a projectile.

2 Throwing Up, or Dropping Down

2.1 Our First Experiment: Dropping a ball

Experiment One: Dropping Down



You are standing at the top of a tall cliff (height is 25 m) and you drop a ball. What we want to know is

- How long will the ball take till it hits the ground?
- How fast will the ball be moving when it hits the ground?
- How are you going to get your ball back?

We can't just say to our computer *"Hey solve this problem for me!"* we need to break this down and tell it how to solve this problem. Like most computer problems this come down to asking "What information do we need to store?" , "What calculations do we need to do?", and finally "What output do we wsnt to see?"

Step 1: What information do we need to store?

- **Position/Height of the ball.** We will use variable `y` for this. Yes, we could use `h` for height and it would make more sense here, but in general we tend to use `y` for vertical positions and `x` for horizontal positions. Initially the ball is 25 metres above the ground so we write:

```
y = 25
```

- **Velocity of the ball.** We will use variable `v` for this. Initially the ball is not moving so we write:

```
v = 0
```

- **Acceleration of the ball.** We will use variable `a` for this. When we release the ball, it will start moving downwards, due to the effect of gravity. This 'pull' effect feel like an acceleration and is approximately -9.81 m/s^2. The minus sign is because it points downwards towards the ground. So we write:

```
a = -9.81
```

- **Time.** We will use variable `t` for this. And we are going to start measuring time from the instant we drop the ball. so we write:

```
t = 0
```

- **Time Step.** What is this? Think of the ball falling and we taking picture after picture of it as it falls. The computer will calculate the position and velocity at each picture. The more pictures we take the more calculations the computer will make but the more accurate our results will be. We want to take enough picture to get results that are 'accurate enough' but not so much pictures the computer takes too long to do the calculations. Lets take ten pictures every second, so the time between pictures is 0.1, and we write

```python
dt = 0.1
```

Step 2: What calculations do we need to do?

We are going to keep taking picture after picture as the ball falls. To get the computer to repeat something for evert we write

```python
while True:
```

At every 'picture' we need to calculate the ball's position — compare the code to equation (1)

```python
    y = y + v*dt
```

and the ball's velocity — compare the code to equation (2)

```python
    v = v + a*dt
```

and update time. To update time we just add to our current value for time the time step. So we write

```python
    t = t + dt
```

Putting all of the above line together we get

```python
y = 25
v = 0
a = -9.81
t = 0

dt = 0.1

while True:

    y = y + v*dt
    v = v + a*dt
    t = t + dt
```

Create a new program with the above code and save it as `Drop_1.py`.
Run the program, what happens?

In our current program the computer appears to be busy doing work but we see nothing! We[2] forgot to do step 3 and tell the computer what information to output. Lets fix that now.

[2]OK, OK, I forgot, but you should have reminded me that I forgot …

Step 3: What information do we want to output?

At every 'picture' we take we are interested in the time the picture was taken and the position and the velocity of the ball. To do this we append the following line to our program:

```
    print(t, y, v)
```

Note the indentation in the above line. It is part of the **while** loop. Otherwise it will only output the final position and velocity.

Append the above line to your `Drop_1.py` program and run it. What happens?

It is a feast or a famine — now we are drowning in data! There are two problems here:

- The computer does not stop when the ball hits the ground — think about this, why would the computer stop? how does it know about the ground?

  Remember, computers are fast but stupid — we need to tell the computer when to stop. In this experiment we want to break out of our **while** loop when the ball hits the ground, that is, when the height of the ball is zero, so we append line

  ```
      if y<0: break
  ```

  Why did we use "`y<0`" and not "`y==0`"? In other words, why did we test if the height is less than zero instead of testing if the height is exactly zero?

  We did this because, when we take our pictures we are very, very unlikely to take the picture at the exact point in time at which the ball hits the ground so our test checks if the ball has hit the ground instead.

- What is the story with all the digits after the decimal point? The computer does not know how accurate you want your output so it outputs as much accuracy as it can compute. To fix this, change the print line from

  ```
      print(t, y, v)
  ```

  to

  ```
      print("%12.2f %12.2f %12.2f" % (t, y, v))
  ```

  This command is looks complicated but if we break it down it should become clearer:

  - The code `"%12.2f %12.2f %12.2f"` is called a **format string**. Think of it as a message with three empty boxes in it. The data after the string is used to fill in each box.
  - The letter "`f`" is used to say that the data will be a float (a number with a decimal point).
  - The "12.2" says to use at least 12 spaces to output the number but only output two digits after the decimal point.

Run your updated program and answer the following questions, in the space provided.

Question: How long is the ball falling? s

Question: At what speed is the ball moving when it hits the ground? m/s

Our program works but there are some issues:

- We don't see the ball position and velocity AT the instant we drop the ball.

  This is easy — just change the order of the compute lines with the print line.

- We could improve our output by adding a message to label each column of data.

  This is also easy — we just insert a print command before the **`while`** loop.

- We want to take more pictures but that will mean more output — how can tell the computer compute more values but only output some of these?

  This will require an **`if`** statement.

The following program fixes the above three issues.

Create a new program with the following code and save it as `Drop_2.py`.

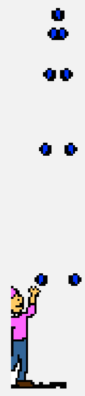
Drop_2.py

```
 1  # setup physics problem
 2  y = 25
 3  v = 0
 4  a = -9.81
 5  t = 0
 6
 7  dt = 0.1
 8  output_dt = 0.1
 9
10  # print header
11  print("%12s %12s %12s" % ("time", "height", "velocity"))
12  print("="*39)
13
14  # do computation
15  while True:
16
17      if  abs(int(t/output_dt)-t/output_dt)<output_dt/2:
18          print("%12.2f %12.2f %12.2f" % (t, y, v))
19
20      y = y + v*dt
21      v = v + a*dt
22      t = t + dt
23
24      if y<0: break
25
26  # output final data
27  print("The ball was falling for %.2f seconds." % t)
28  print("At impact ball was moving at %.2f metres per second." % -v)
```

- Notice the new variable `output_dt` and the **`if`** condition on line 17. These control how often we output results. We can now make `dt` as small as we like (depending on how fast our computer is) to get more accurate results, but still only outputting the same amount of data.

2.2 Our Second Expexriment: Throwing Up (without the mess!)

Experiment One: Dropping Down



You throw a ball up into the air with an initial velocity of 25 m/s. What we want to know is

- How long will the ball take till it hits the ground?
- How fast will the ball be moving when it hits the ground?
- What is the maximum height that the ball reaches?

To keep everybody's results the same we are going to assume that your hand is 1.3 metres above the ground..

Save your `Drop_2.py` program as `Throw_1.py` and modify it to look like the following.

Throw_1.py

```
# setup physics problem
y = 1.3
v = 25
a = -9.81
t = 0

dt = 0.001
output_dt = 0.01

max_height = 0

# print header
print("%12s %12s %12s" % ("time", "height", "velocity"))
print("="*39)

# do computation
while True:

    if abs(int(t/output_dt)-t/output_dt)<dt:
        print("%12.3f %12.2f %12.2f" % (t, y, v))

    y = y + v*dt
    v = v + a*dt
    t = t + dt

    if y<0: break

# output final data
print("The ball was in the air for %.2f seconds." % t)
print("At impact ball was moving at %.2f metres per second." % -v)
print("Maximum height of ball was %.2f metres." % max_height)
```

The program runs but the maximum height is incorrect. You need to insert some computation lines to update the maximum height, and then answer the following questions.

Question: How long is the ball in the air? s

Question: At what speed is the ball moving when it hits the ground? m/s

Question: What is the maximum height of the ball? m

This task is a little harder. Try it and we are here to help.

You need to modify the code so that it also stores when the ball reaches maximum height and then answer the following questions.

Question: How long is the ball moving upwards? s

Question: How long is the ball moving downwards? s

Question: What is the total distance the ball moves? m

This task is a little harder again. Try it and we are here to help.

Question: At what velocity is the ball moving when it passes your hand on the way down? m/s

2.3 Can I see it?

Numbers are great and all that … but a picture would be nice. Lets use some turtle graphics to visualise our ball. To do this we

- Lets call our turtle `ball` instead of `bob`, (but feel free to keep `bob` if you like) so we write

```python
import turtle
ball = turtle.Turtle()
```

- Since throwing a turtle is not nice, lets change it to a ball, so we write

```python
ball.shape("circle")
ball.shapesize(0.5)
```

- Next we setup the ball's initial position using

```python
ball.penup()
ball.goto(0,y)
ball.pendown()
```

- And trance the ball movement using

```
    ball.goto(0,y)
```

Save your `Throw_1.py` program as `Throw_2.py` and modify it to look like the following.

Throw_2.py

```
# setup physics problem
y = 1.3
v = 25
a = -9.81
t = 0

dt = 0.01
output_dt = 0.1

max_height = 0

import turtle
ball = turtle.Turtle()
ball.shape("circle")
ball.shapesize(0.5)

ball.penup()
ball.goto(0,y)
ball.pendown()

# print header
print("%12s %12s %12s" % ("time", "height", "velocity"))
print("="*39)

# do computation
while True:

    if abs(int(t/output_dt)-t/output_dt)<dt:
        print("%12.3f %12.2f %12.2f" % (t, y, v))

    ball.goto(0,y)

    y = y + v*dt
    v = v + a*dt
    t = t + dt

    if y<0: break

# output final data
print("The ball was in the air for %.2f seconds." % t)
print("At impact ball was moving at %.2f metres per second." % -v)
print("Maximum height of ball was %.2f metres." % max_height)
```

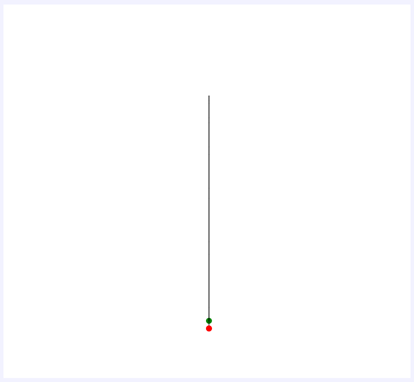
To be fair, that output was kinda sad. We need to zoom in a bit to see it properly. To do this is we insert two lines to create a screen where the x- and y- coordinates are more suited to our needs.

Save your `Throw_2.py` program as `Throw_3.py` and insert the following two lines.

`Throw_3.py` **(only showing new lines)**

```
import turtle

screen = turtle.Screen()
screen.setworldcoordinates(-5,-5,5,45)
```



OK, this is a little better. I have added some other to mark the start point as green and the end point as red. Modify your `Throw_3.py` code so that we see the start and the end point using the command `ball.stamp()` and `ball.color()`.

3 Where are we now?

OK, we have done enough for today, next week we are going to:

- Move to two-dimensional projectile movement.
- Generate graphs of the projectile movement.