Different numerical techniques, including differentiation, integration and the use of filters to solve common physics problems.

Felipe Giraldo Grueso
Physics and Mechanical Engineering Student
Universidad de Los Andes

Why coding?

- Learning how to use computers to our favor has saved wars over the history of humanity.
- Just remember Alan Turing, with his machine, he shortened the second world war and saved many lives.
- But before someone is ready to have such a great impact, the basis of coding must be learnt.
- Differentiation, integration and the implementation of filters in Python can be one of the most important things to learn when getting ready to save the world using numerical methods.

What is a numerical method?

- Its an approximation and a simplification to solve analytical problems.
- How do you solve the following integral?

$$\int_{-\infty}^{\infty} \frac{\sin x}{x} dx$$

- Complex variable or numerical methods. And which do you think is simpler?

Slicing

- a[start:stop] # items start through stop-1
- a[start:] # items start through the rest of the array
- a[:stop] # items from the beginning through stop-1
- a[:] # a copy of the whole array

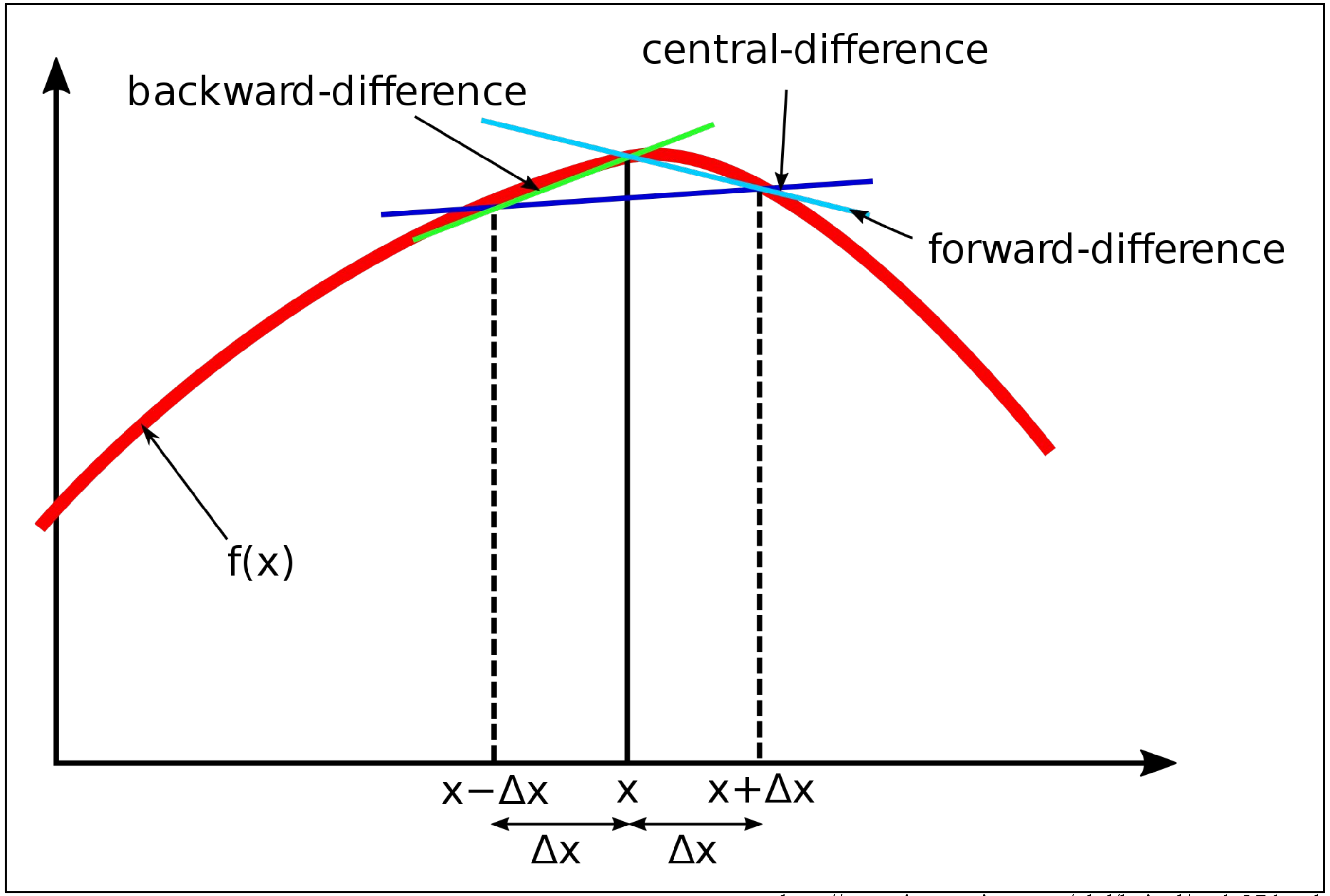
- a[start:stop:step] # start through, not past stop, by step
- a[-1] # last item in the array

- a[::-1] # all items in the array, reversed
- a[1::-1] # the first two items, reversed
- a[:-3:-1] # the last two items, reversed
- a[-3::-1] # everything except the last two items, reversed

Differentiation

- Forward difference
- Backward difference
- Central difference

- How do you implement these methods in Python?



http://www.iue.tuwien.ac.at/phd/heinzl/node27.html

$$f'(x) = \frac{f(x+h) - f(x)}{h}$$

$$f'(x) = \frac{f(x) - f(x-h)}{h}$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h}$$

How would you implement these?

-

- Let's try to implement them and calculate the derivative of:

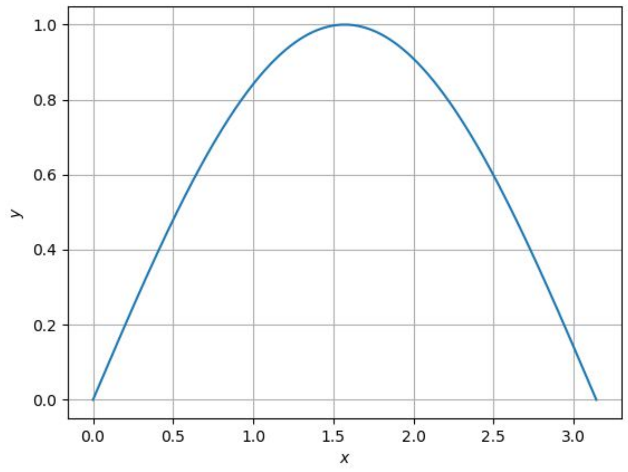
$$\sin x$$

- Take five minutes to think how would you calculate the derivative of this function using numerical methods.

- Hint: Think of slicing.

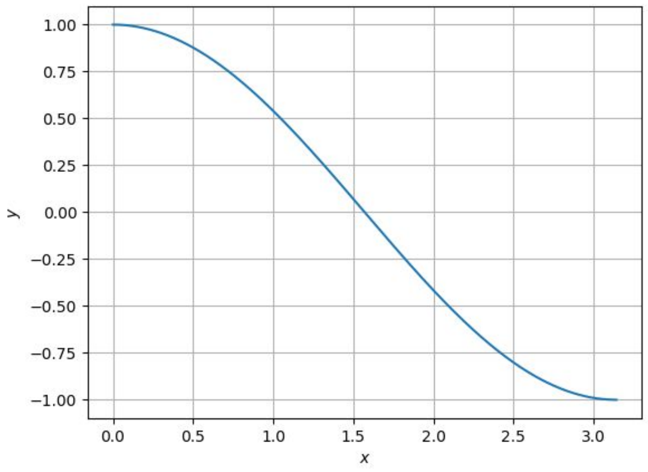
```python
def fun(x):
	return np.sin(x)

x = np.linspace(0,np.pi,1000)
y = fun(x)
```



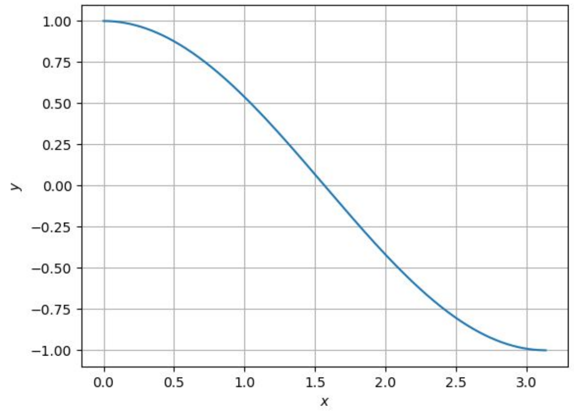

```python
def fun_prime(x):
	return np.cos(x)

y_prime = fun_prime(x)
```



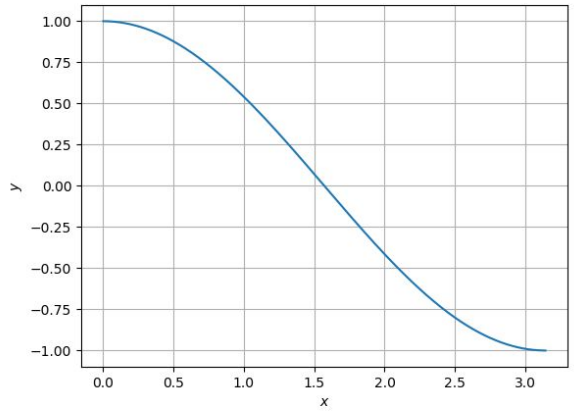
```
def forward_difference(x,y):
    h = (max(x)-min(x))/float(len(x)-1)
    prime = (y[1:]-y[0:-1])/float(h)
    return prime

y_prime_forward = forward_difference(x,y)
```




```
def backward_difference(x,y):
    h = (max(x)-min(x))/float(len(x)-1)
    prime = (y[1:]-y[0:-1])/float(h)
    return prime

y_prime_backward = backward_difference(x,y)
```



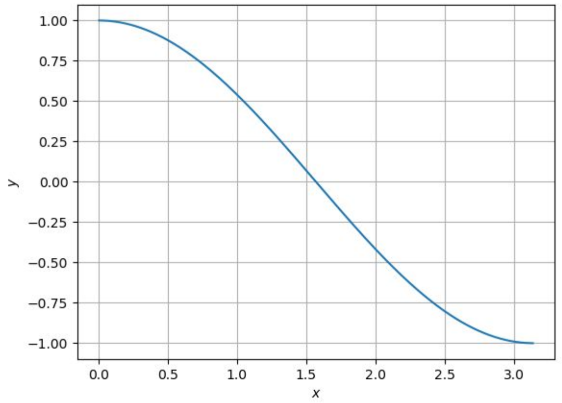
Notice any difference?

```
plt.plot(x[:-1],y_prime_forward)
```

```
plt.plot(x[1:],y_prime_backward)
```

```
def central_difference(x,y):
    h = (max(x)-min(x))/float(len(x)-1)
    prime = (y[2:]-y[0:-2])/float(2*h)
    return prime

y_prime_central = central_difference(x,y)
```



With forward and backward we lose 1 point and with central we lose 2 points.

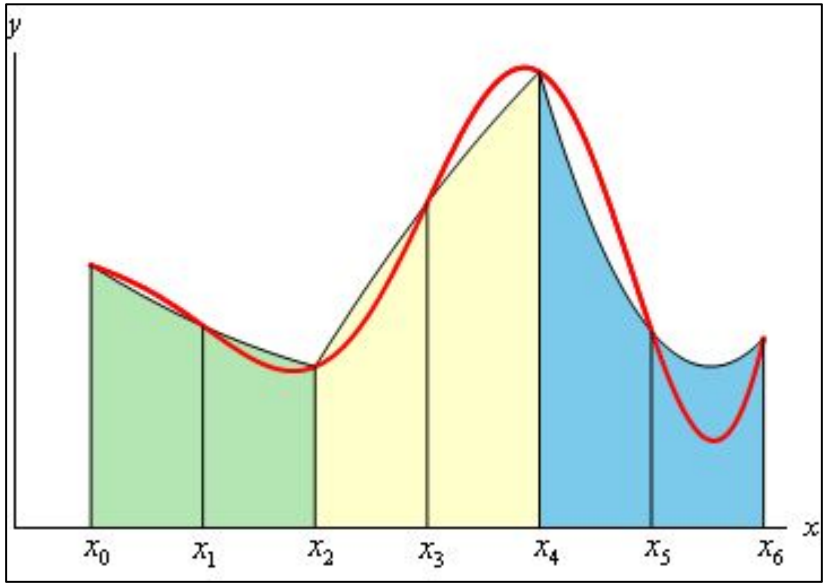
Meaning that for the first point we could use forward difference, for the last point we could use backward difference and for the rest, central difference.

```
def complete_prime(x,y):
    h = (max(x)-min(x))/float(len(x)-1)
    prime_0 = float(y[1]-y[0])/float(h)
    prime_last = float(y[-1]-y[-2])/float(h)
    prime = (y[2:]-y[0:-2])/float(2*h)
    complete_prime = np.concatenate([[prime_0],prime,[prime_last]])
    return complete_prime
```

Integration

- Trapezoids
- Simpson’s Method
- Monte Carlo

- How do you implement these methods of integration in Python?



http://tutorial.math.lamar.edu/Classes/CalcII/ApproximatingDefIntegrals.aspx

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{N} \frac{h}{2}\left(f(x_{i+1}) + f(x_i)\right) \rightarrow w_i = \frac{h}{2}, h, \dots, h, \frac{h}{2}$$

$$\int_a^b f(x)\,dx \approx \frac{a-b}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right) \rightarrow w_i = \frac{h}{3}, \frac{4h}{3}, \frac{2h}{3}, \frac{4h}{3}, \dots, \frac{4h}{3}, \frac{2h}{3}, \frac{4h}{3}, \frac{h}{3}$$

How would you implement these?

-

- Let’s try to implement them and calculate the integral our fun(x):

$$\sin x$$

- Take five minutes to think how would you calculate the integral of this function using numerical methods.
- Hint: Once again, think of slicing.

- Using our function, we know analytically that:

$$\int_0^{\pi} \sin x \, dx = 2$$

```
def int_trap(x,y):
    h = (max(x)-min(x))/float(len(x)-1)
    y *= h
    integral = np.sum(y[1:-1]) + ((y[0]+y[-1])/2.0)
    return integral
```

trapezoids = int_trap(x,fun(x)) $\longrightarrow$ trapezoids_scipy = integrate.trapz(fun(x),x)

Integral of sin(x)[0,π] using trapezoids:

- Using our implementation: 1.9999983517708517
- Using SciPy: 1.999998351770852

```
def int_simpson(x,y):
    h = (max(x)-min(x))/float(len(x)-1)
    y *= h
    integral = np.sum(y[1:-1:2]*4.0/3.0) + np.sum(y[2:-2:2]*2.0/3.0) + ((y[0]+y[-1])/3.0)
    return integral
```

simpson = int_simpson(x_simp,fun(x_simp)) → simpson_scipy = integrate.simps(fun(x), x)

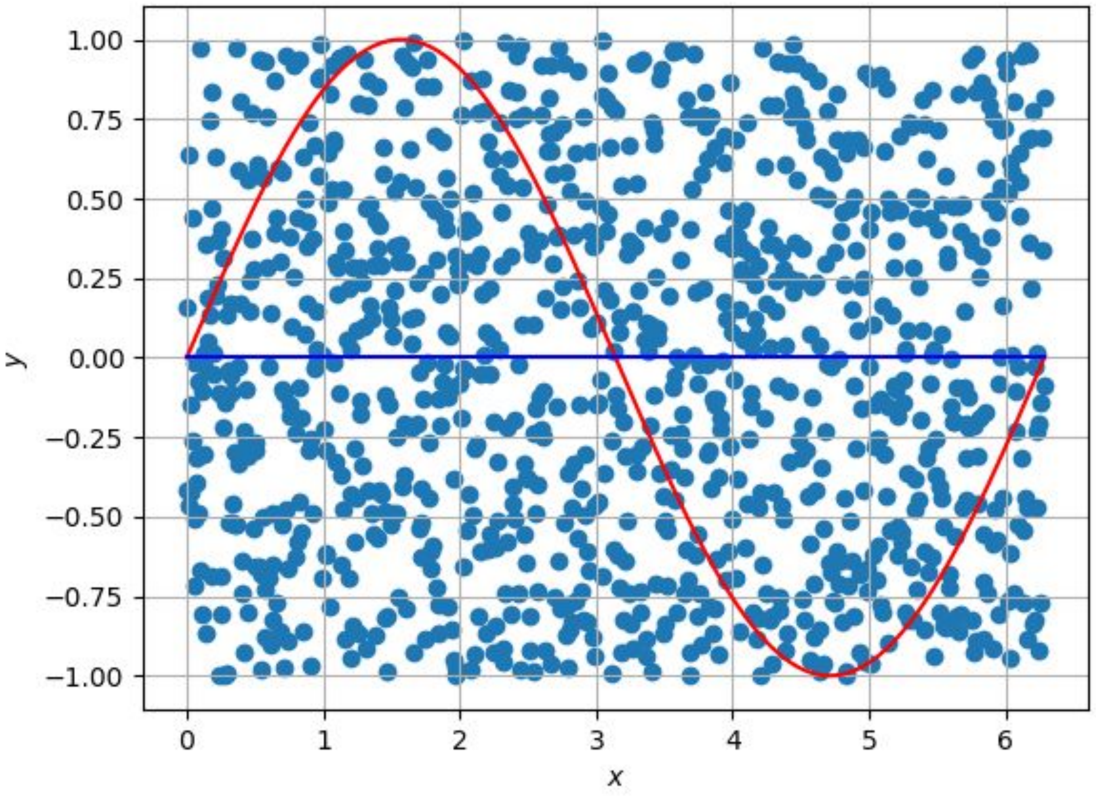
Integral of sin(x)[0,π] using Simpson's Method:

- Using our implementation: 2.0000000000010822
- Using SciPy: 2.0000000000010822

→ x_simp = np.linspace(0,np.pi,**1001**)

So Simpson, as it was suspected before, seems to have a more accurate result. And the use of SciPy is best for this type of problems.

Let’s take a look at Monte Carlo’s method



Take five minutes to think how would you implement this method.

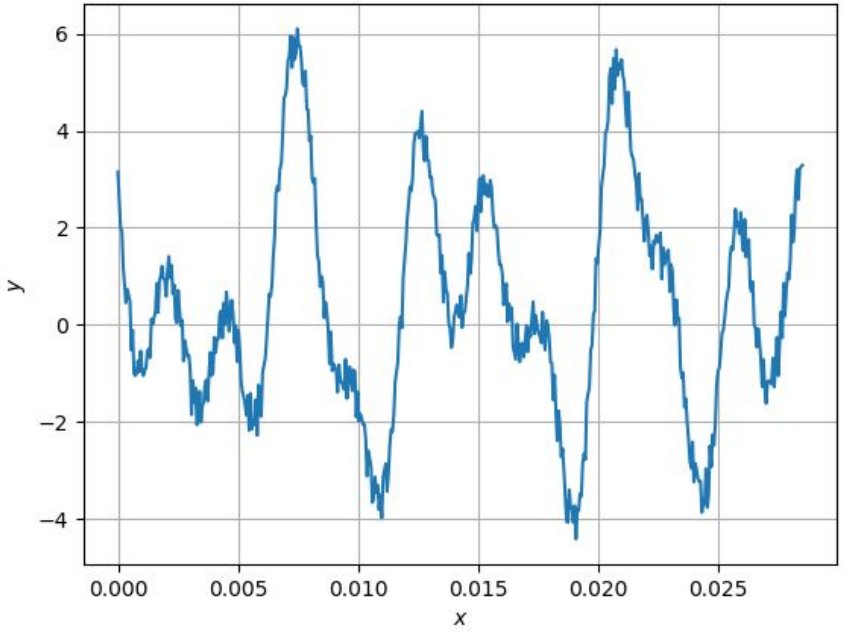
```python
def int_mc(x_min,x_max,y,N):
    counter = []
    y_max = max(y)
    y_min = min(y)
    area = (x_max-x_min)*(y_max-y_min)
    y_ran = np.random.uniform(y_min,y_max,N)
    for i in range(N):
        if(y_ran[i]>0 and y[i]>0 and abs(y_ran[i])<=abs(y[i])):
            counter.append(1)
        elif(y_ran[i]<0 and y[i]<0 and abs(y_ran[i])<=abs(y[i])):
            counter.append(-1)
        else:
            counter.append(0)
    return (np.mean(counter)*area)
```

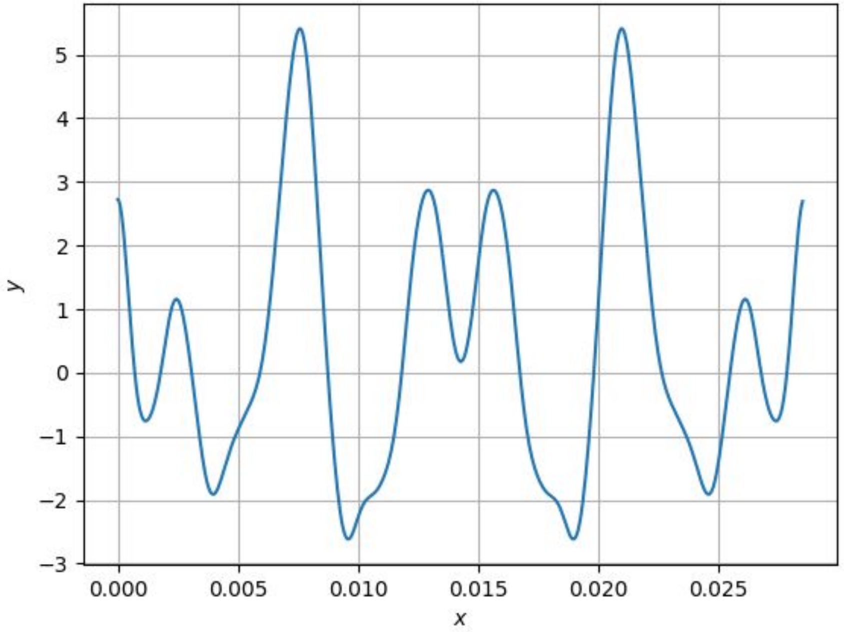
Integral of sin(x)[0,2pi] using Monte Carlo's Method:

- Using 1000 points: 0.006283177934168347
- Using 10000 points: 0.013823004828136034
- Using 100000 points: -0.005466371217186391

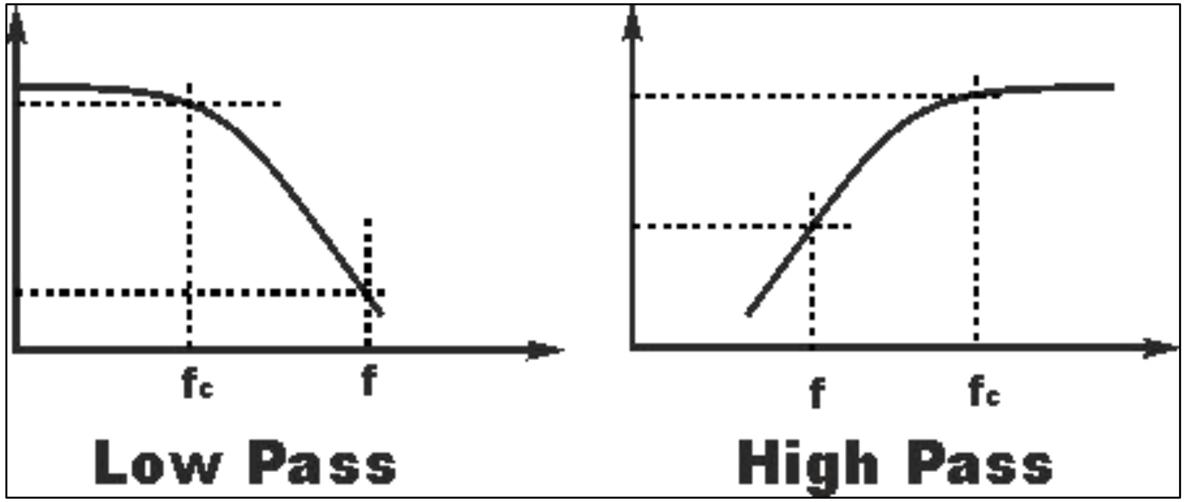
```python
monte_carlo_1000 = int_mc(0,np.pi,fun(np.random.uniform(0,np.pi,1000)),1000)
monte_carlo_10000 = int_mc(0,np.pi,fun(np.random.uniform(0,np.pi,10000)),10000)
monte_carlo_100000 = int_mc(0,np.pi,fun(np.random.uniform(0,np.pi,100000)),100000)
```

How do we get from the first graph to the second?





Filters!



https://electronics.stackexchange.com

But how do we get to the frequency domain from the time domain?

Discrete Fourier Transformation

Using SciPy, we can forget about the complex calculations and we only have to use the fftpack.

Common procedure

- Obtain experimental data that has noise you want to reduce.
- Use the DFT to enter the frequency domain.
- Cutoff the frequencies you want to eliminate.
- Recover your filtered signal using the inverse DFT.

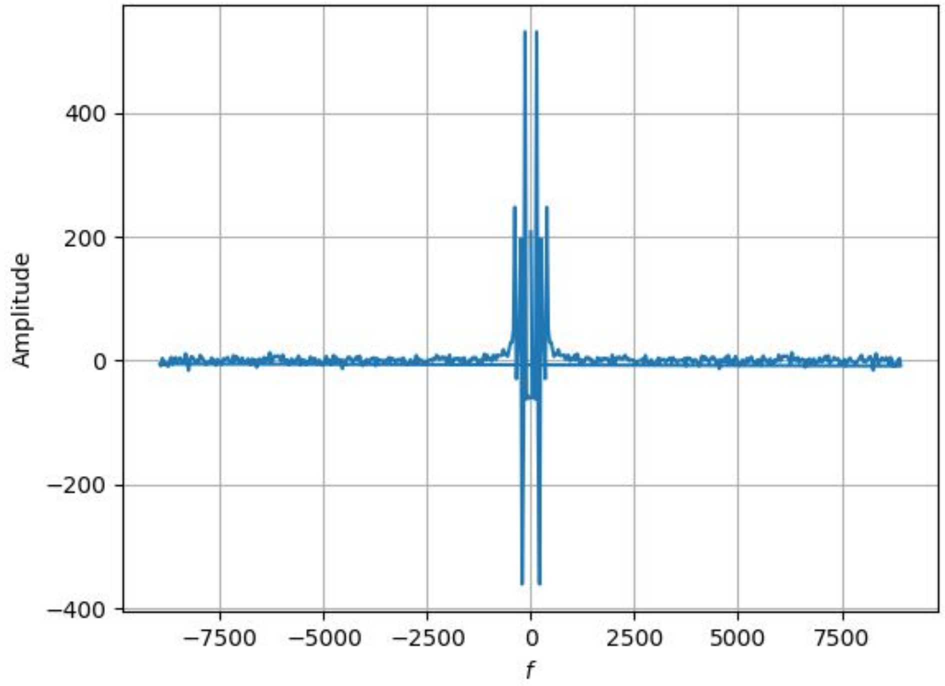
- For the next exercise go to https://1drv.ms/u/s!AgZJE-OGYI0NiFsq3i5NdC3jbYvP?e=cYk492 and download the file *signal.dat.* Data taken from the Physics Department at Universidad de Los Andes.

```
fourier_transform = np.real(fft(signal_y))
frequencies = fftfreq(len(signal_x),signal_x[1]-signal_x[0])
```

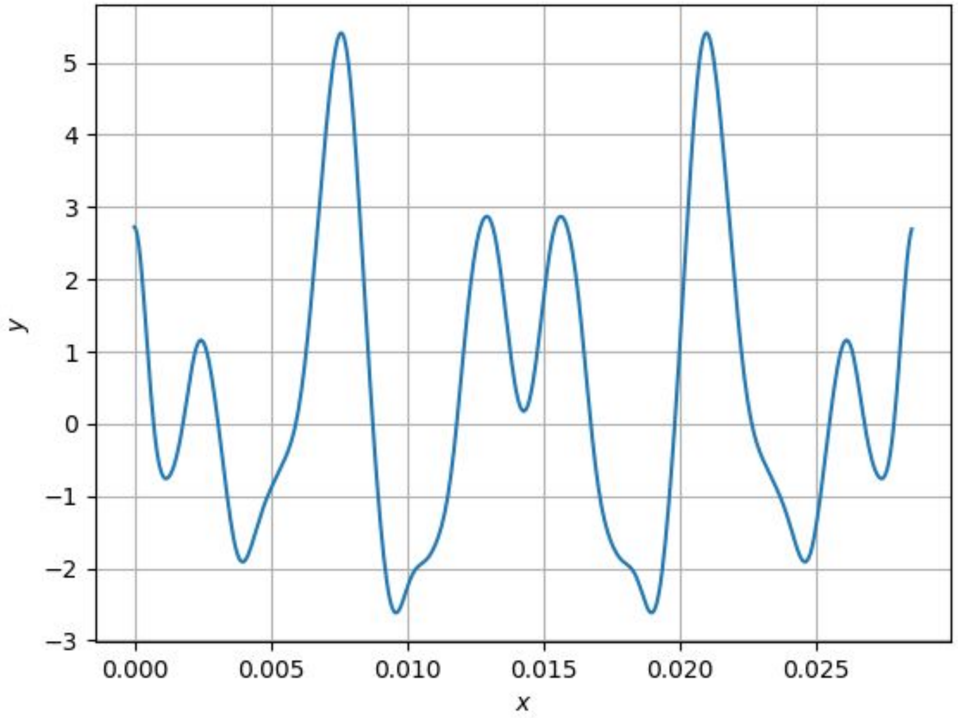
Which filter should we use? Let’s try a low-pass and a high-pass.

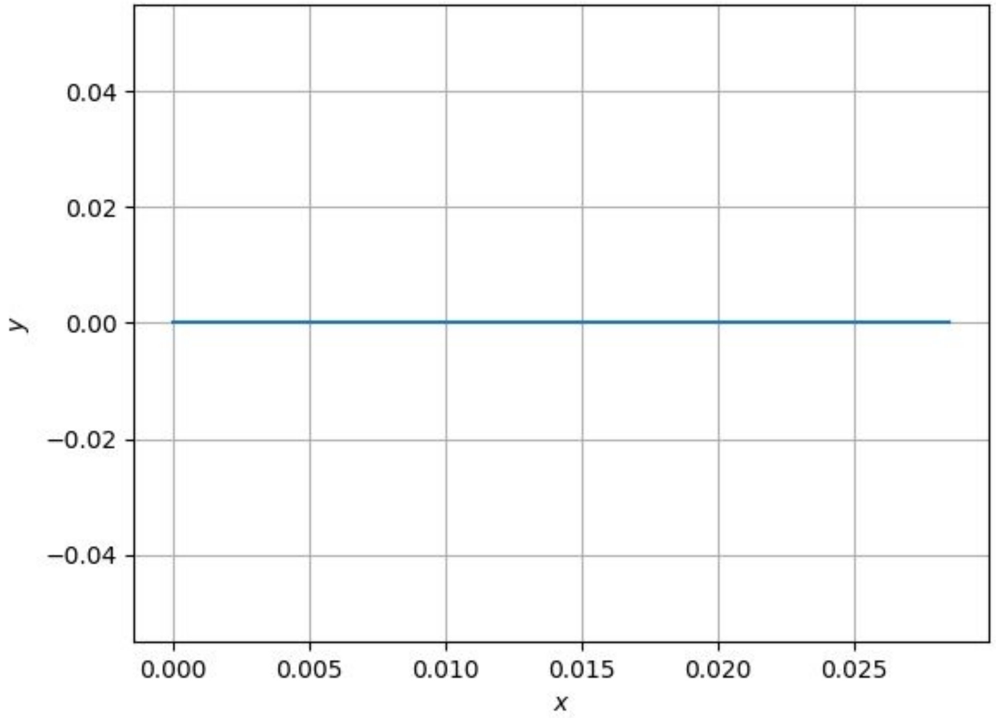
```
def filter_lowpass(frequencies,transform,n):
    for i in range(0,len(frequencies)):
        if abs(frequencies[i])>n:
        transform[i] = 0
    return transform

def filter_highpass(frequencies,transform,n):
    for i in range(0,len(frequencies)):
        if abs(frequencies[i])<n:
        transform[i] = 0
    return transform
```



```
signal_y_lowpass = ifft(filter_lowpass(frequencies, fourier_transform, 1000))
signal_y_highpass = ifft(filter_highpass(frequencies, fourier_transform, 1000))
```





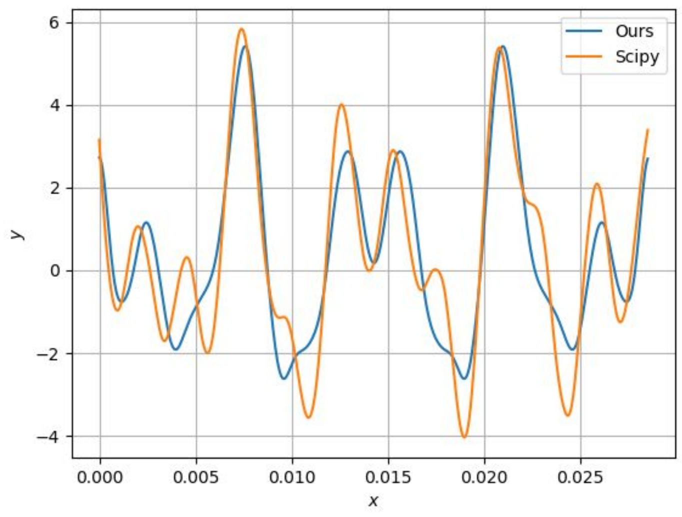
Which one do you think is the low-pass and the high-pass?

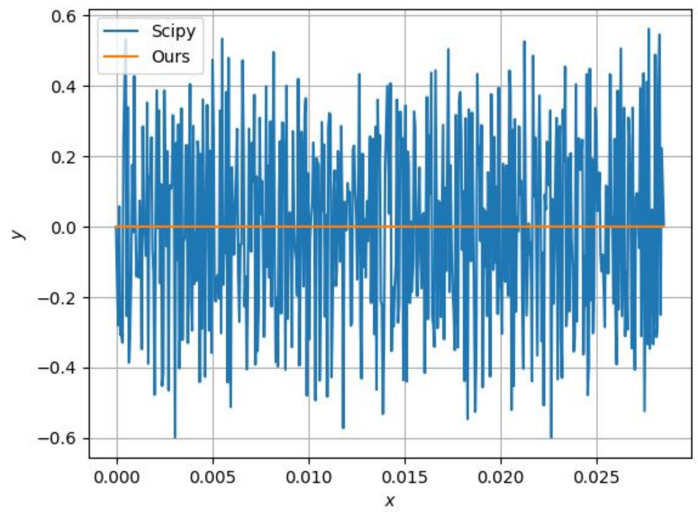
But SciPy can help us get a way better result!

```
b_low, a_low = signal.butter(3, 1000/((1/(signal_x[1]-signal_x[0]))/2), 'low')
scipy_y_lowpass = signal.filtfilt(b_low, a_low, signal_y)
```

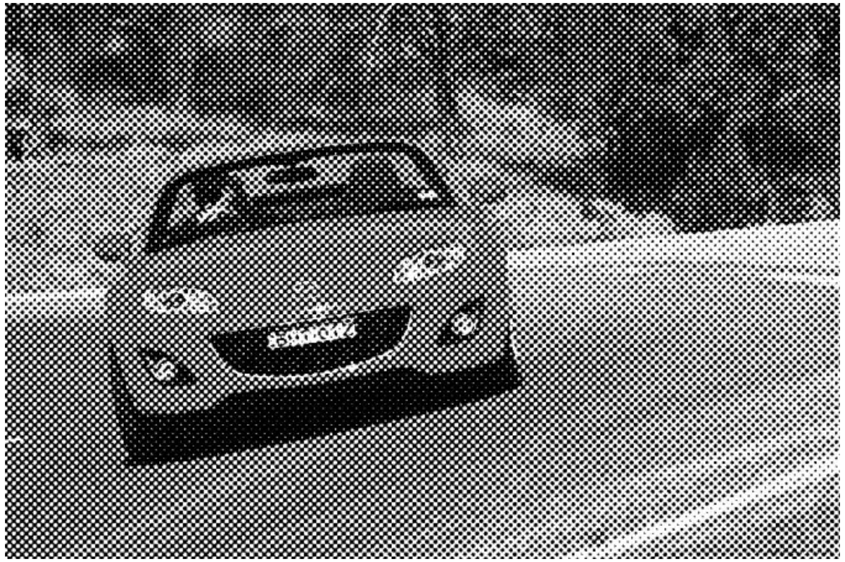
Polynomials of the infinite impulse response filter

```
b_high, a_high = signal.butter(3, 1000/((1/(signal_x[1]-signal_x[0]))/2), 'high')
scipy_y_highpass = signal.filtfilt(b_high, a_high, signal_y)
```





If you are interested, you can try to implement a filter in 2D using SciPy. And you can do things just like this.



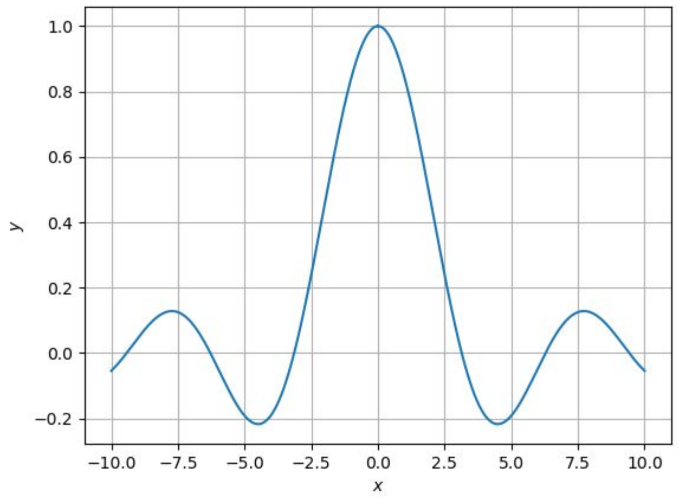
https://www.datasciencecentral.com/profiles/blogs/solving-some-image-processing-problems-with-python-libraries-part

Going back to our first problem

- Try to integrate numerically the following integral:

$$\int_{-\infty}^{\infty} \frac{\sin x}{x} dx$$

- Estimate infinity to 10E6.



```
def last_fun(x):
    fun = np.sin(x)/x
    fun[np.isnan(fun)] = 1
    return fun
```

```
integral = int_simpson(np.linspace(-10**6,10**6,10**6 +1),last_fun(np.linspace(-10**6,10**6,10**6+1)))
```

We know analytically that:

$$\int_{-\infty}^{\infty} \frac{\sin x}{x} dx = \pi$$

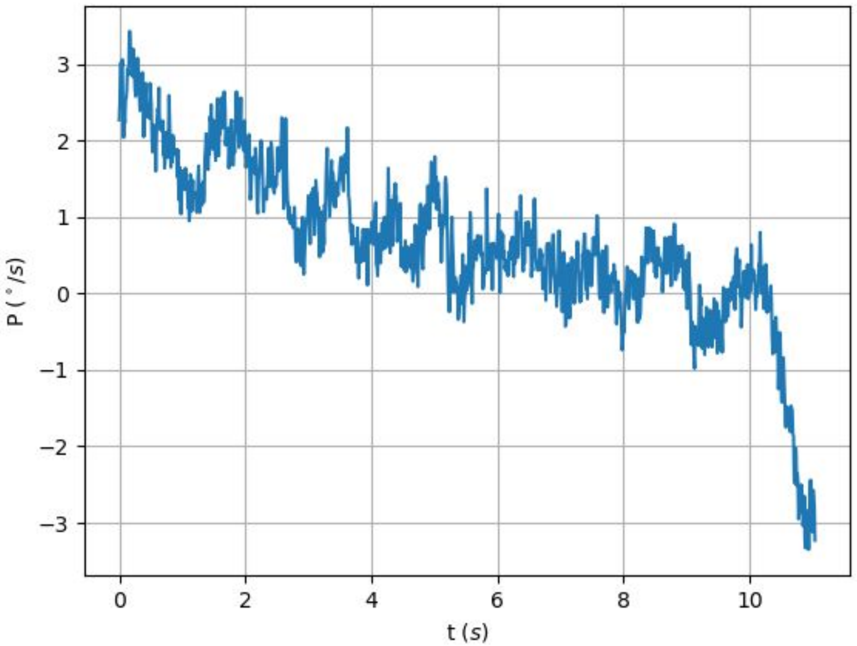
The integral of sin(x)/x $[-\infty,\infty]$ using numerical methods is: 3.1415904780234003

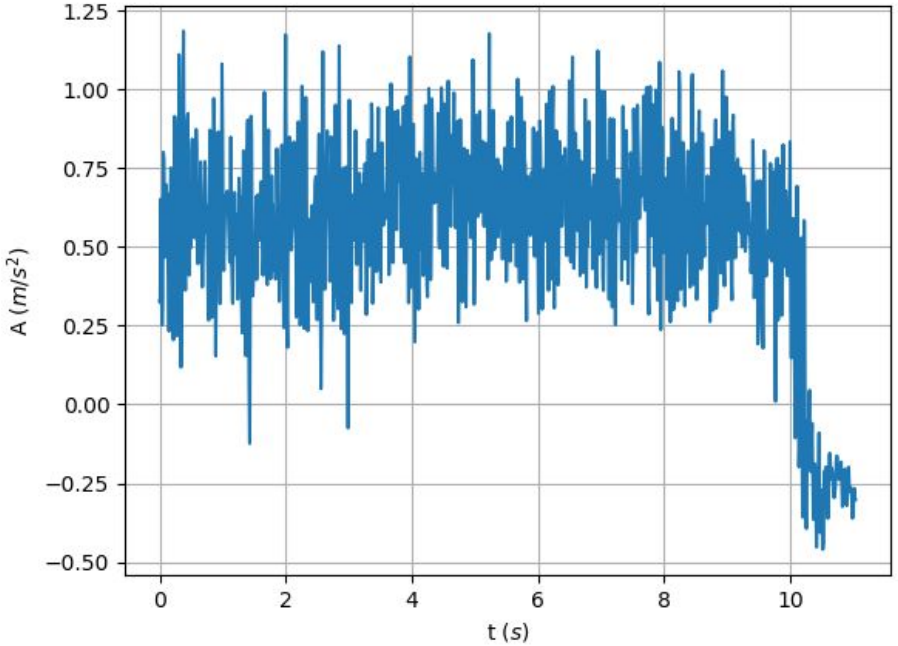
With an error of 6.925042908821231E-05%

Let’s solve a physics problem

- For the next exercise go to https://1drv.ms/t/s!AgZJE-OGYI0NiFzbK_z9bdBqvf_k?e=92YeBH and download the file *boat_data.txt.*
- This are real measurements of the acceleration and the pitch rate of a boat just like this:

Data taken from the Mechanical Engineering Department at Universidad de Los Andes.





Pitch Rate

Acceleration

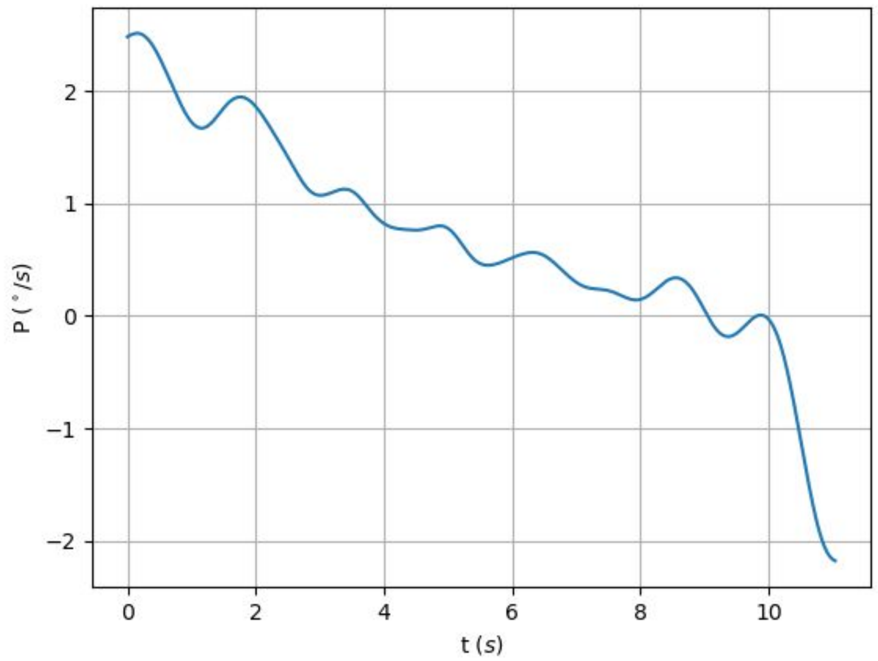
Try to find the final angle of elevation and the total displacement of the boat.

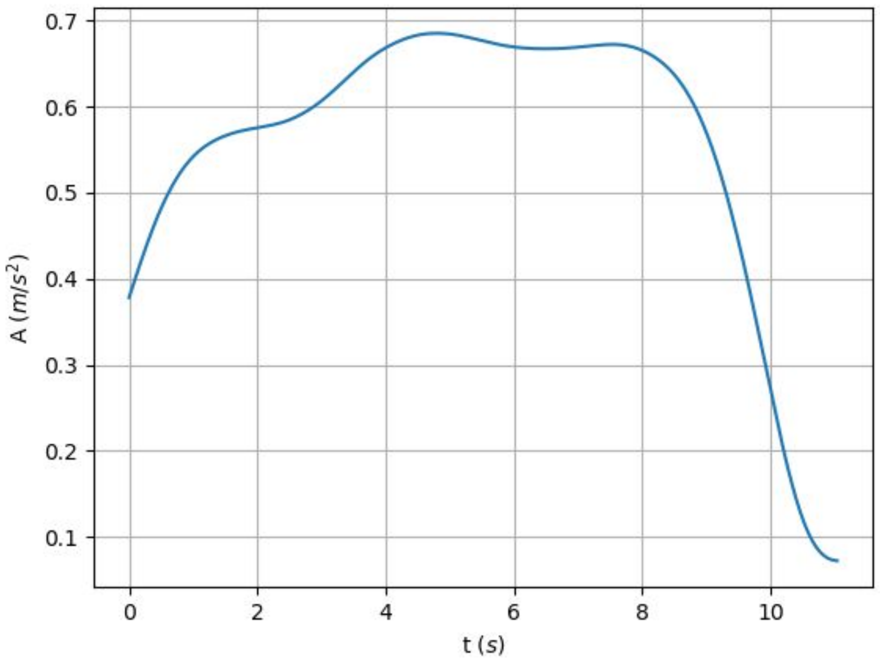
First, we filter the signals

```
a_acce, b_acce = signal.butter(2,(0.27/50), 'low')
a_pitch, b_pitch = signal.butter(2,(0.6341/50), 'low')

acce_filt = signal.filtfilt(a_acce,b_acce,boat_acce)
pitch_filt = signal.filtfilt(a_pitch,b_pitch,boat_pitch)
```

We would have to use the DFT to find the cutoff frequency.





Pitch rate filtered

Acceleration filtered

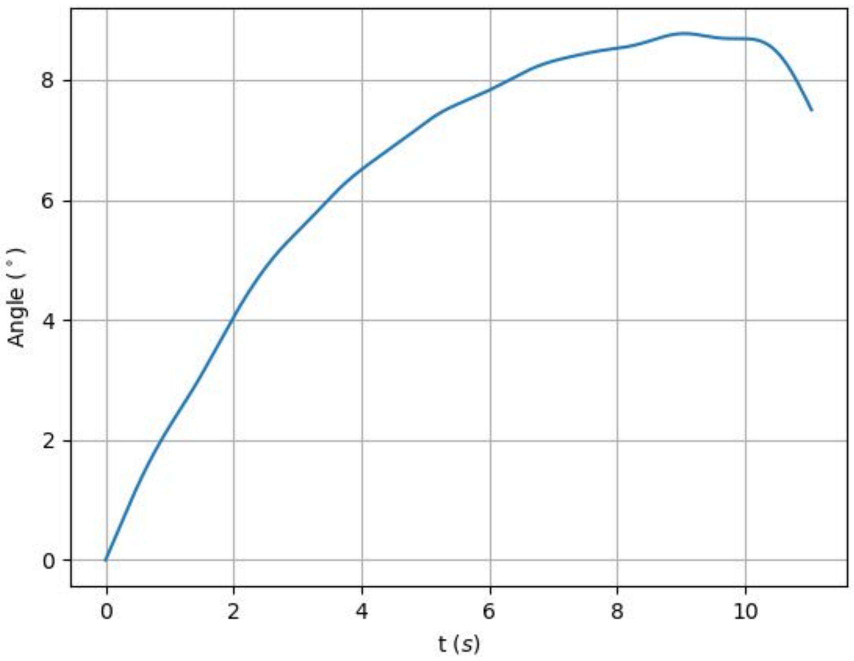
And then, we integrate

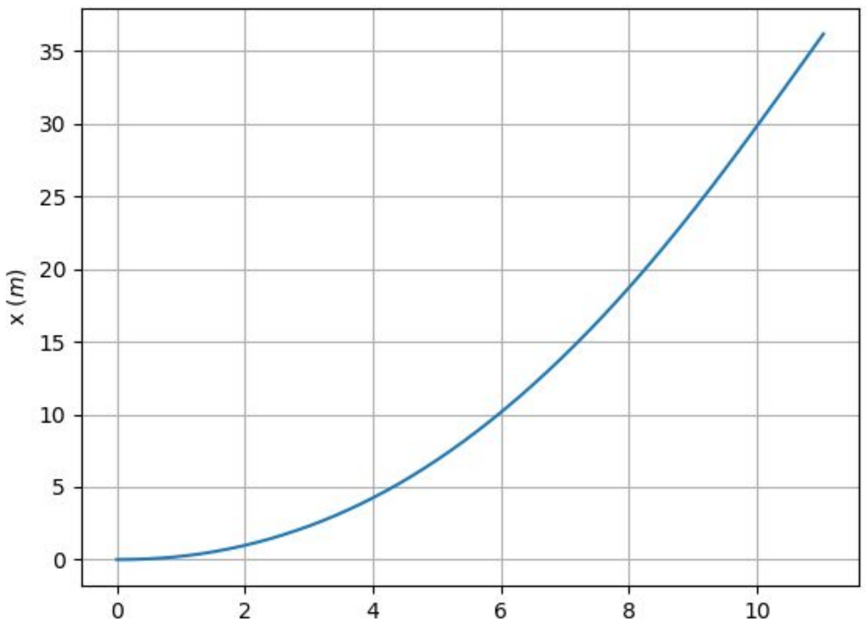
- We integrate once the pitch rate to find the final angle and we integrate twice to find the total displacement.

Having any problems?

```
x_boat = cum_trapz(boat_t[:-1], cum_trapz(boat_t,acce_filt))
angle_boat = cum_trapz(boat_t,pitch_filt)
```

```
def cum_trapz(x,d_y,y_0):
    y = np.empty(len(x))
    y[0] = y_0
    for i in range(0,len(x)-1):
        y[i+1] = (x[i+1]-x[i])*(((d_y[i+1]+d_y[i]))/2.0)+y[i]
    return y
```





Angle

Displacement

Thank you for your attention!
Hope you are now ready to save the world!