Búsqueda de picos y ceros

Vamos a usar la función `find_peaks` del paquete `scipy`

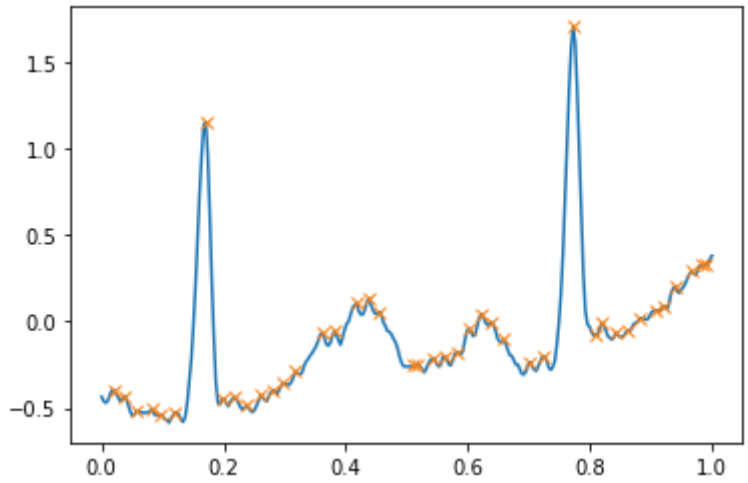
La documentación completa la encuentran acá:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html

```
#cargo las librerías necesarias
import matplotlib.pyplot as plt #para dibujar
from scipy.signal import find_peaks #la funcion find_peaks
from scipy.signal import unique_roots #la funcion find_peaks
from scipy.misc import electrocardiogram #para cargar datos de prueba
import numpy as np #para operaciones matemáticas
```

La función `find_peaks` detecta máximos locales, es decir, puntos que son más altos que los dos puntos a sus lados (¿podríamos usarla para detectar mínimos en lugar de máximos?).

La función nos devuelve `peaks`, que tiene los índices (las posiciones) de los máximos. Podemos usar esos índices para extraer la posición (`x`) y altura (`y`) de los picos.

```
y = electrocardiogram()[2200:2500]
t = np.linspace(0,1,len(y) )
peaks, _ = find_peaks(y)
plt.plot(t,y)
plt.plot(t[peaks], y[peaks], "x")
plt.show()

print(peaks)
```

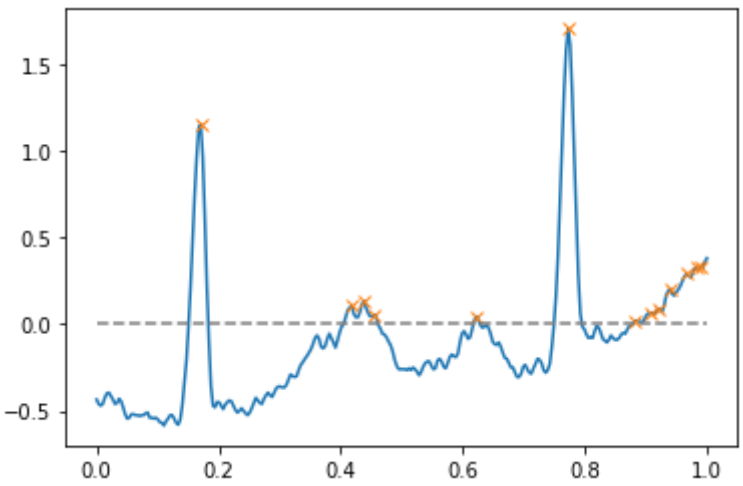



```
[  6  11  17  25  29  36  51  59  65  71  78  84  89  95 108 114 125 131
 136 153 155 162 168 174 180 186 191 197 210 216 231 242 245 252 258 264
 271 275 281 289 294 296]
```

Tiene muchos parámetros que nos permiten elegir cuáles de todos los máximos detectar. En este documento mencionamos los más comunes.

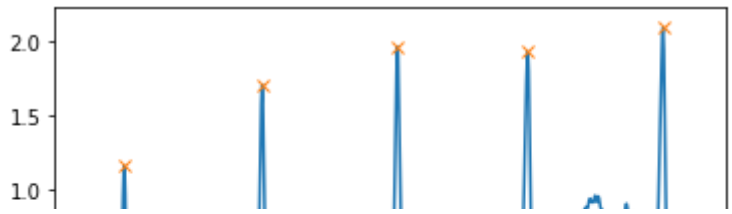
En este primer ejemplo usamos el parámetro `height`, para detectar sólo los picos que están por encima de cero:

```
y = electrocardiogram()[2200:2500]
t = np.linspace(0,1,len(y) )

peaks, _ = find_peaks(y, height=0)
plt.plot(t,y)
plt.plot(t[peaks], y[peaks], "x")
plt.plot(t,np.zeros_like(t), "--", color="gray")
plt.show()
```



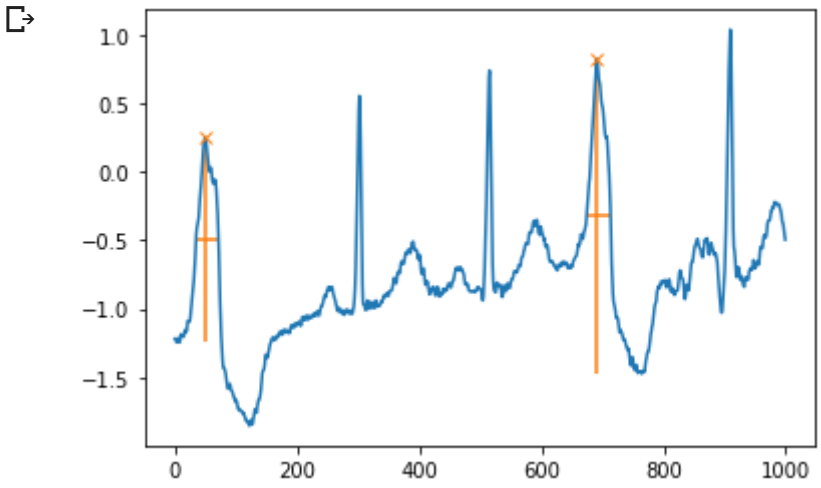
El parámetro `distance` limita la distancia mínima entre máximos sucesivos. Está en unidades de samples, no de tiempo.

```
y = electrocardiogram()[2200:3000]
t = np.linspace(0,1,len(y) )

peaks, _ = find_peaks(y, distance=150)
plt.plot(t,y)
plt.plot(t[peaks], y[peaks], "x")
plt.show()


print(np.diff(peaks))
print(np.diff(t[peaks]))
```



Tiene otros parámetros, como `width` y `prominence`. Y la segunda respuesta, `properties`, nos da información de cada uno de los picos detectados.

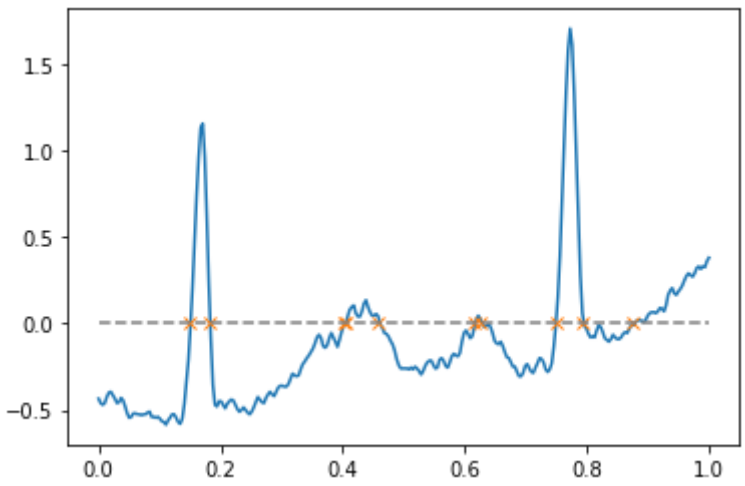
```
x = electrocardiogram()[17000:18000]
peaks, properties = find_peaks(x, prominence=1, width=20)
properties["prominences"], properties["widths"]
plt.plot(x)
plt.plot(peaks, x[peaks], "x")
plt.vlines(x=peaks, ymin=x[peaks] - properties["prominences"],
           ymax = x[peaks], color = "C1")
plt.hlines(y=properties["width_heights"], xmin=properties["left_ips"],
           xmax=properties["right_ips"], color = "C1")
plt.show()
```



Para estimar las raíces (ceros) de la señal podés utilizar el siguiente código. En la definición de `i0` se calculan todos los casos en los que un elemento multiplicado por su anterior es negativo (es decir, cambio de signo). Se infiere de esto que hay ahí adentro al menos una raíz. La otra línea comentada permite seleccionar solo las raíces de pendiente positiva (útil para verificar periodos por ejemplo).

```
x = electrocardiogram()[2200:2500]
t = np.linspace(0,1,len(x))

i0 = np.nonzero(x[0:-1]*x[1:]<=0)[0]

# La siguiente línea la podés descomentar (sacando #) si querés que se ejecute
# i0 = np.nonzero(np.logical_and(x[0:-1]*x[1:]<=0,x[0:-1]<0))[0] # Elige solo las raíce

plt.plot(t,x)
plt.plot(t[i0], np.zeros_like(t[i0]), "x")
plt.plot(t,np.zeros_like(t), "--", color="gray")
```

```
11 plt.plot(t,np.zeros_like(t),  '--' , color= 'gray' )
12
13 plt.show()
```




```
1 # Otra forma de calcular ceros
2 # x0 = np.pad(x, (0, 1), 'wrap') # Se organizan los valores de la señal
3 # tolerancia = 0.001 # Acá podés ingresar la tolerencia al distinguir ceros
4 # i0 = np.delete( np.where(np.diff(np.sign(x0)) != 0)[0] , -1) # Se extraen los índices
5 # i0_alt = np.delete(   np.where(np.logical_and(np.abs(np.diff(x0)) >= tolerancia, np.d
```