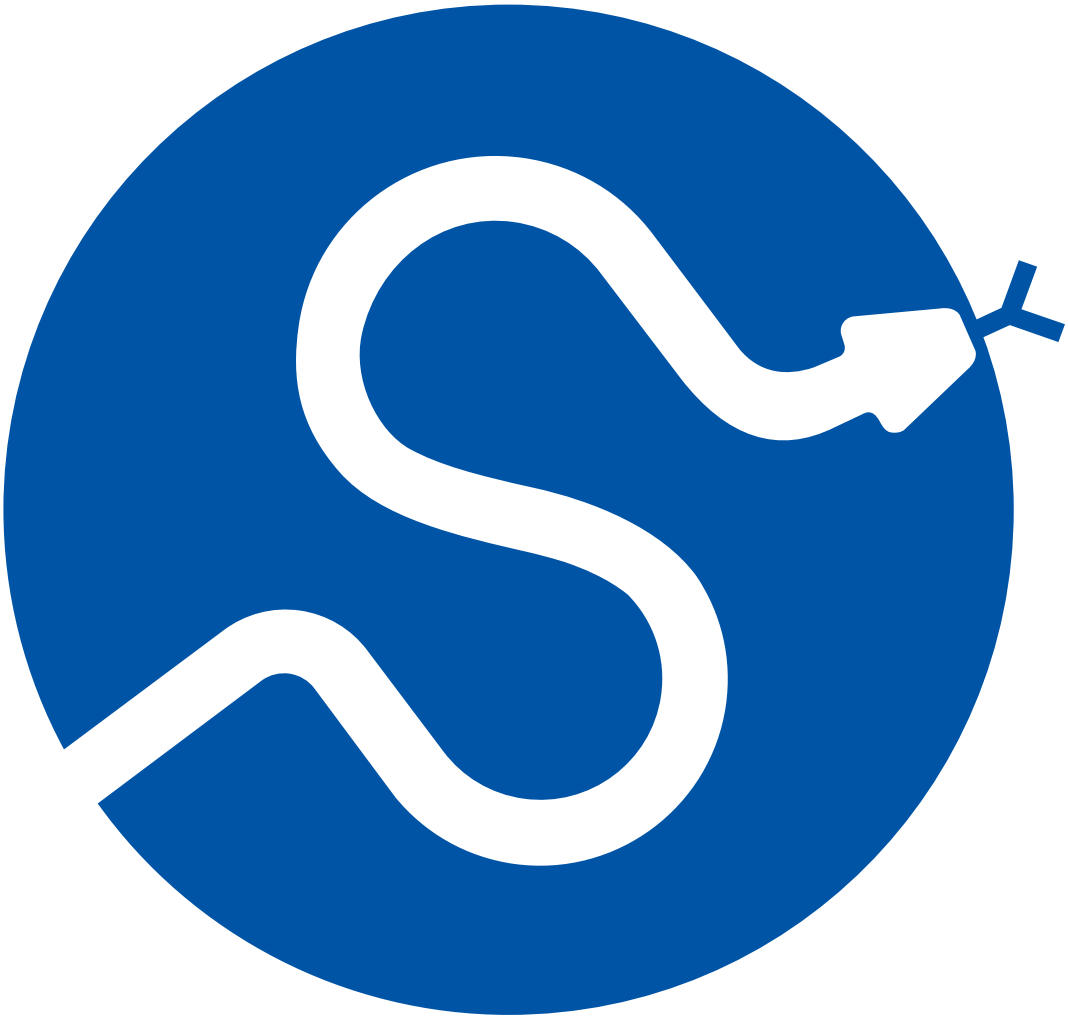
Introduction to Python Day 5

Verjinia Metodieva and Daniel Parthier

2025-03-18

SciPy

- A python library for scientific questions
- NumPy in the background
- Many useful functions for signals (filtering, finding peaks, FFT, deconvolution)
- Some statistical tests (t-test, wilcoxon, correlation, etc.)



Signal processing

- Module `signal`
- Important part of analysis
- Getting information of interest

Detect peaks

- Function to detect peaks in data: `find_peaks`

- You can specify criteria
  - Width (minimum, maximum)
  - Distance to next peak
  - Threshold
  - etc.

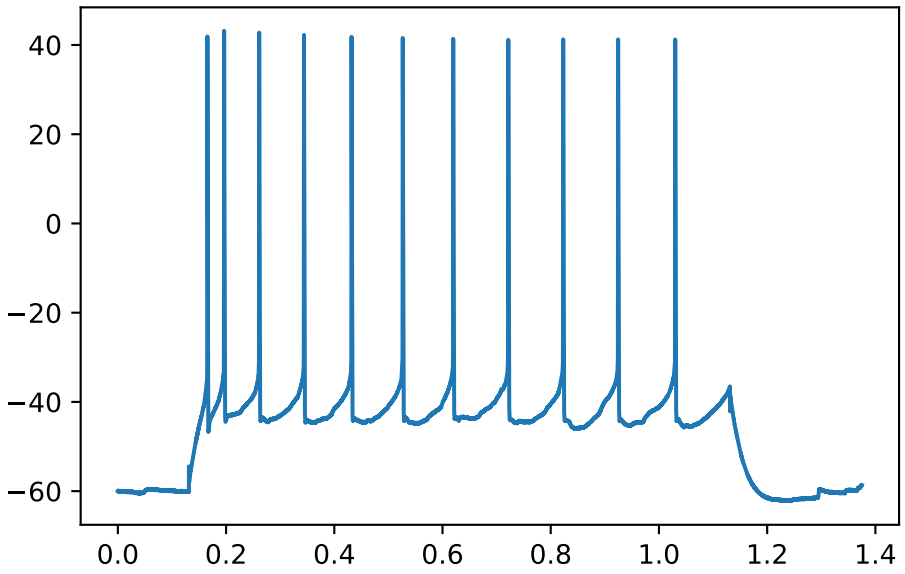
Detect peaks

```
import json
import numpy as np
import matplotlib.pyplot as plt

sampling_rate = 2e4
with open('data/charact_data.json') as f:
    data = json.load(f)
# print the keys
data = {key: np.array(data[key]) for key in data.keys()}
time_s = np.linspace(0, data['D1'].shape[1]/sampling_rate, data['D1'].shape[1])
```

Detect peaks

```
plt.plot(time_s, data['D1'][10])
plt.show()
```



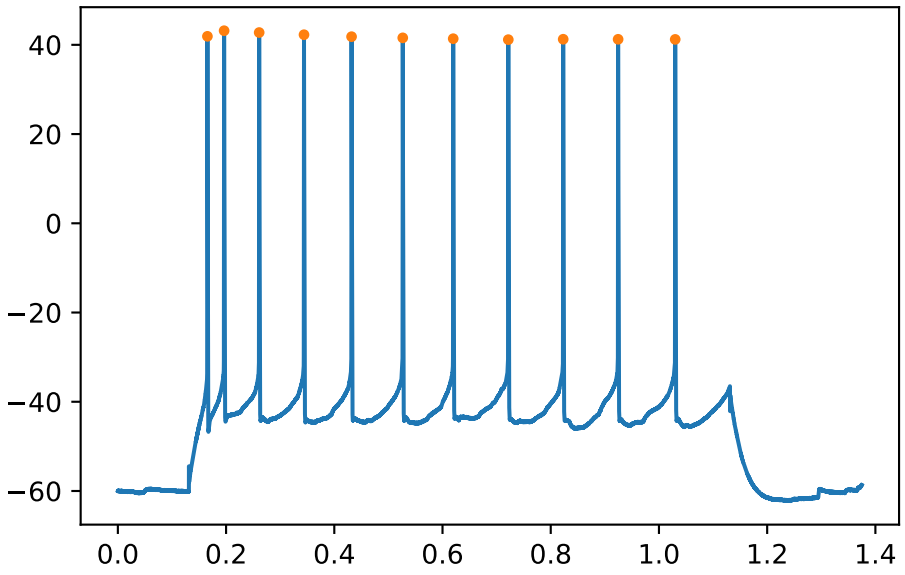
Detect peaks

```
from scipy import signal
peak_location, peak_properties = signal.find_peaks(data['D1'][10], height=0, width=[10, 50],
print(peak_location)
```

```
[ 3310  3927  5223  6880  8638 10530 12395 14431 16460 18490 20600]
```

Detect peaks

```
plt.plot(time_s, data['D1'][10])
plt.plot(time_s[peak_location], data['D1'][10][peak_location], '.')
plt.show()
```



Detect peaks

- Peak properties:
  - Peak height, widths, prominences, thresholds
  - Depends on function input
  - More details in documentation

Practice

- Find the peaks of all traces
- Append the location and height to separate lists
- Plot the number of spikes against the current
  - `[-300, -200, -150, -100, -50, 0, 50, 100, 150, 200, 250, 300]`
- Think of a good way to make use of functions

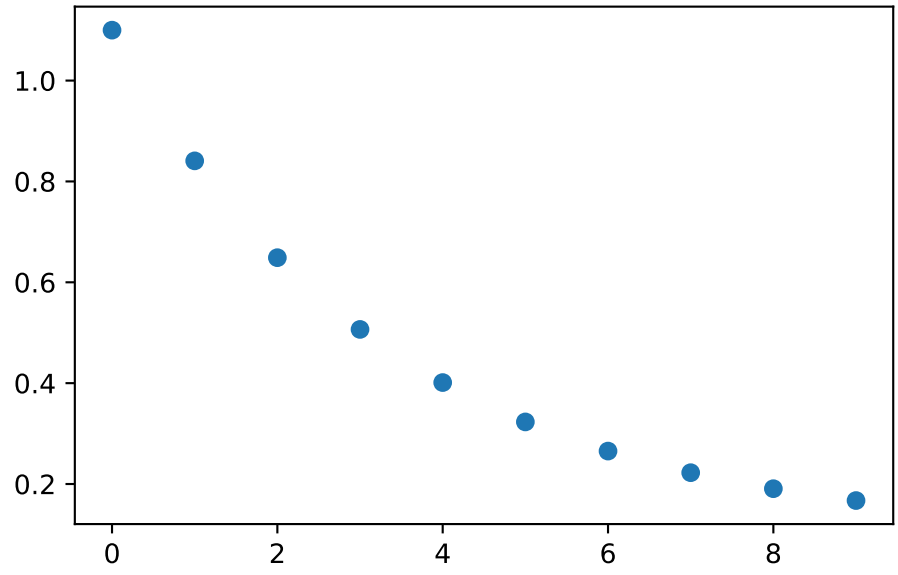
Fit data

- Modul `optimize`
- Different functions to fit lines/functions
- `curve_fit` allows to specify a function which is fit to data

Specify a function

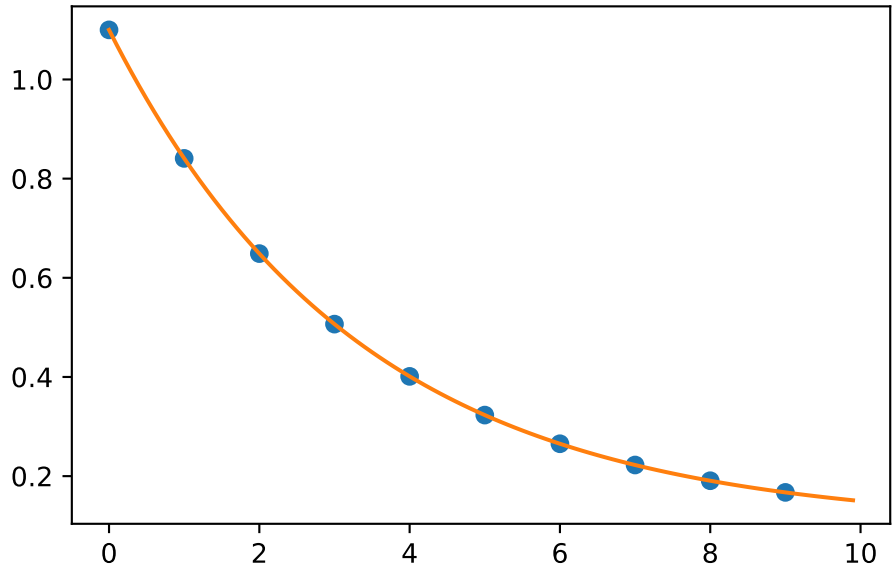
```
def func(x, a, b, c):
    return a * np.exp(-b * x) + c

x = np.arange(0,10,1)
y = func(x, 1,0.3,0.1)
plt.plot(x, y, 'o')
plt.show()
```



Fit function

```
from scipy import optimize
popt, pcov = optimize.curve_fit(func, x, y, p0=(1,1,1))
x_fit = np.arange(0,10,0.1)
plt.plot(x, y, 'o')
plt.plot(x_fit, func(x_fit, *popt))
plt.show()
print(popt)
```




```
[1.  0.3 0.1]
```

Convolve and Deconvolve

- Signals can be distorted
- Multiplying something to signal with `convolve`
- Remove known distortion from signal `deconvolve`
- Where do you see use cases?

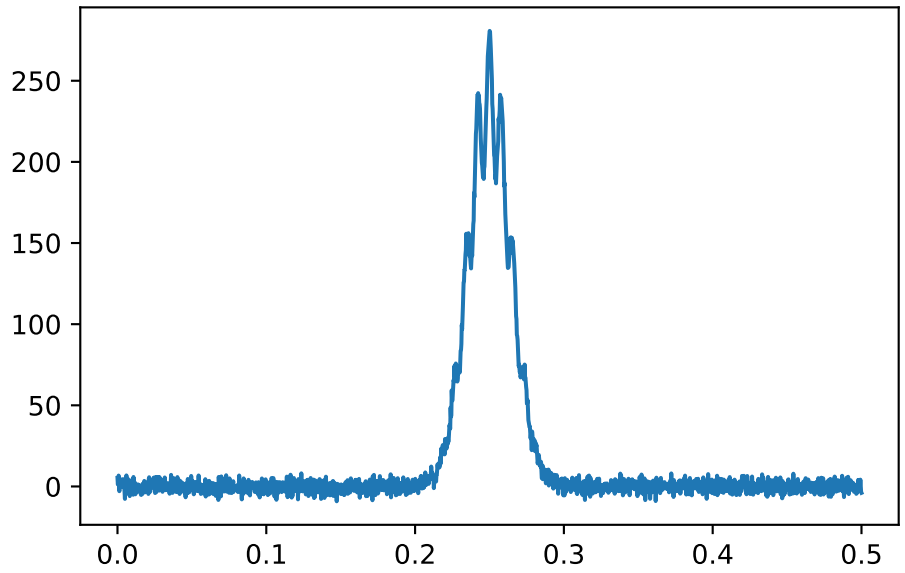
Signal analysis (power)

```
time = np.linspace(start=0, stop=0.5, num=2000)
mu, sigma = 0.25, 0.01
```

```
sinewave = np.sin(time * 250 * np.pi)
gaussian = (1 / (np.sqrt(2 * np.pi * np.square(sigma))) *
            np.exp(-(np.square(time - mu) /np.square(2 * sigma))))
ripple = gaussian * sinewave
```

Signal analysis (power)

```
np.random.seed(0)
trace = ripple + gaussian*6 + np.random.normal(0, 3, size=time.shape)
plt.plot(time, trace)
plt.show()
```



Signal analysis (power)

- Module `signal`
- Power spectrum to analyse frequencies
- e.g. `welch`

Power spectrum

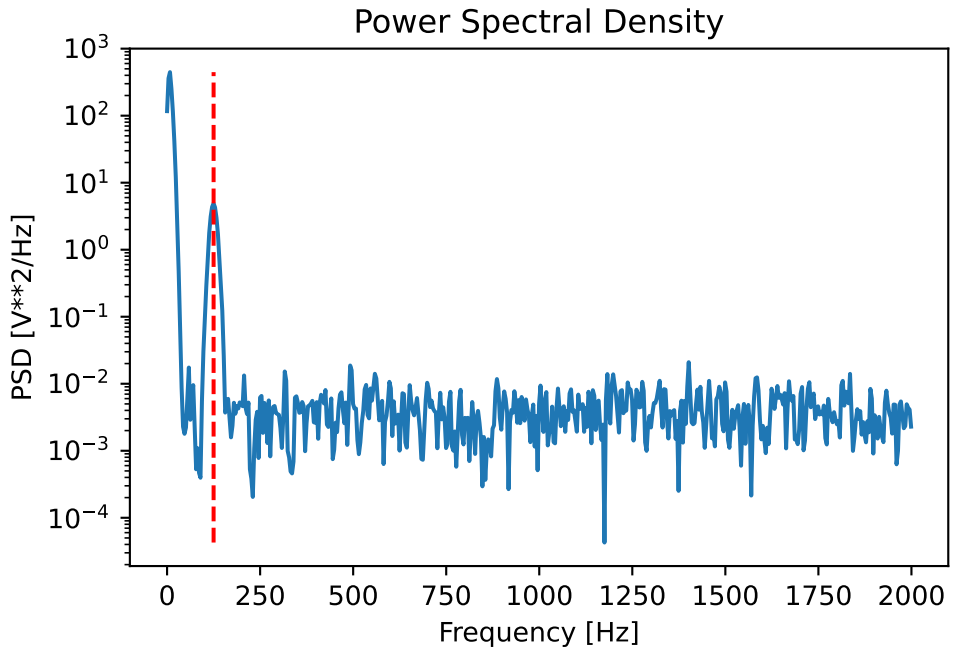
```
f, Pxx_den = signal.welch(trace, fs=1/time[1], nperseg=1024, average='median')
plt.semilogy(f, Pxx_den)
plt.vlines(125, Pxx_den.min(), Pxx_den.max(),color='r', linestyle='--')
```

```
plt.xlabel('Frequency [Hz]')
plt.ylabel('PSD [V**2/Hz]')
plt.title('Power Spectral Density')
plt.show()
```

```
Text(0.5, 0, 'Frequency [Hz]')
```

```
Text(0, 0.5, 'PSD [V**2/Hz]')
```

```
Text(0.5, 1.0, 'Power Spectral Density')
```



- Returns frequencies `f` and power densities `Pxx_den`

Filtering data

- Filtering data is used to remove different components from signal
- lowpass, highpass, bandpass
- different types of filters (Butterworth, Chebyshev, FIR, etc.)

Filter construction

- Need different parts
  - Nyquist frequency
  - Filter frequencies (0 to 1 - frequency/Nyquist)
  - Type of filter
  - Order ('strength')
- Outputs are filter coefficients

```
from scipy import signal
sampling_rate = 1/time[1]
nyquist = sampling_rate / 2
b, a = signal.butter(N=5, Wn=[100/nyquist, 250/nyquist], btype='band')
```

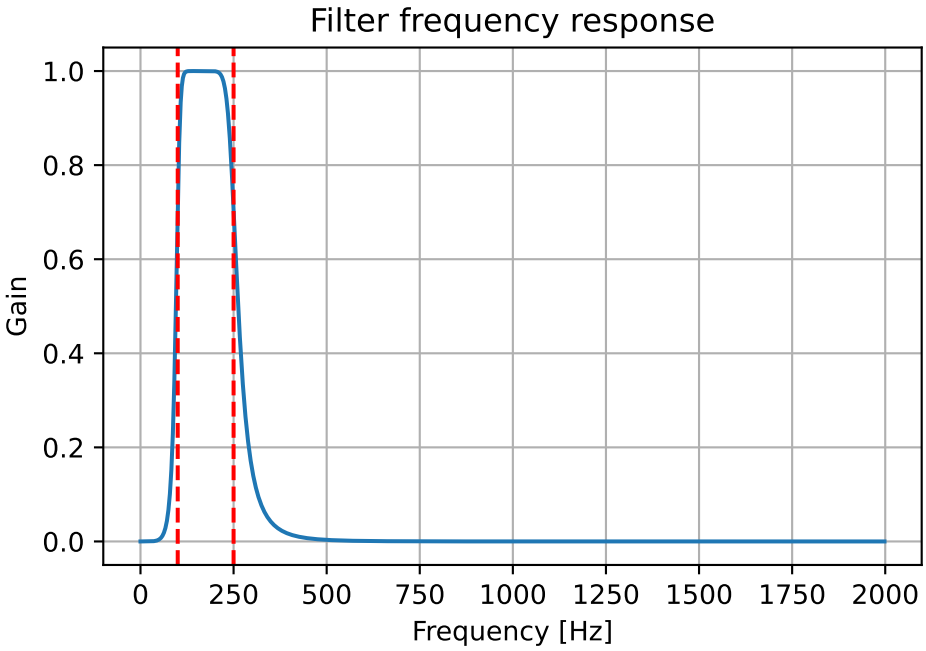
Filter construction

```
w, h = signal.freqz(b, a, fs=sampling_rate, worN=2000)
plt.plot(w, abs(h))
plt.title('Filter frequency response')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Gain')
plt.grid()
plt.axvline(100, color='r', linestyle='--')
plt.axvline(250, color='r', linestyle='--')
plt.show()
```

```
Text(0.5, 1.0, 'Filter frequency response')
```

```
Text(0.5, 0, 'Frequency [Hz]')
```

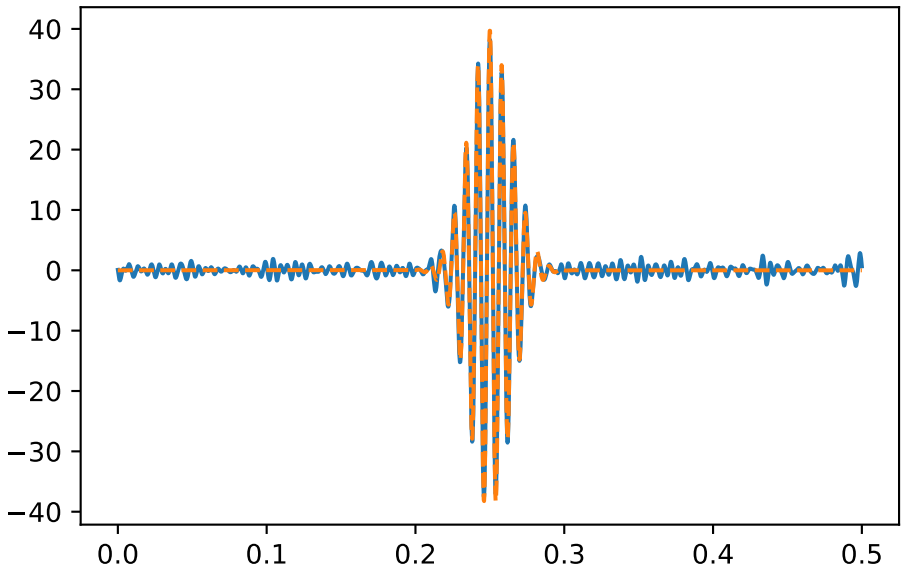
```
Text(0, 0.5, 'Gain')
```



Apply filter

- Different ways of filtering (IIR/FIR)
- one way or two ways (filtfilt)

```
ripple_filtered_trace = signal.filtfilt(b, a, trace)
plt.plot(time, ripple_filtered_trace)
plt.plot(time, ripple, '--')
plt.show()
```



Filter for lowpass

```
b, a = signal.butter(5, 80/nyquist, btype='low')
lowpass_filtered_trace = signal.filtfilt(b, a, trace)
plt.plot(time, trace)
plt.plot(time, lowpass_filtered_trace, '--')
plt.show()
```