

- 
- 
- 
- 
- 
- 

### Plotting NumPy series

As an example, let us import a time series data. This represent human electroencephalogram (EEG) as recorded during normal background activity.

```
1 from pandas import read_csv
2
3 from matplotlib.pyplot import subplots, show
4
5 from numpy import arange, linspace, zeros
```

```
1 df = read_csv("data/EEG_background.txt", delim_whitespace=True)
2
3 df.head()
```

```
1          FP1      FP2      F3      F4  ...      E02      EM1      EM2
2  0  -7.4546  22.8428   6.28159  15.6212  ...   13.7021  12.9109  13.7034
3    9.37573
4  1 -11.1060  21.4828   6.89088  15.0562  ...   13.7942  13.0194  13.7628
5    9.44731
6  2 -14.4000  20.0907   7.94856  14.1624  ...   13.8982  13.1116  13.8239
7    9.51796
8  3 -17.2380  18.7206   9.36857  13.0093  ...   14.0155  13.1927  13.8914
9    9.58770
10 4 -19.5540  17.4084  11.06040  11.6674  ...   14.1399  13.2692  13.9652
11    9.65654
12
13 [5 rows x 28 columns]
```

To see the names of the channels (or recording sensors) we can use `head` function.

```
1 df.shape
```

```
1 (2373, 28)
```

### Numpy Plot

The data in the above dataframe `df` is converted to Numpy arrays, here called `df_np`.

Time in the rows, sensors in the columns

```
1 sr = 256          # Sampling rate: 1 / seconds
2
3 duration = 5       # seconds
4
5 df_np = df.to_numpy()
6
7 data = df_np[:duration*sr, :19] ## SF Comment, needs to explain this
   above
8
9 data.shape
```

```
1 (1280, 19)
```

### Python Function

Please execute the following function definition before proceeding. The function code takes data and creates a plot of all columns as time series, one above the other. When you execute the function code nothing happens. Similar to the import, running a function code will only activate it and make it available for later use.

```
1 def plot_series(data, sr):
2     '''
3     Time series plot of multiple time series
4     Data are normalised to mean=0 and var=1
5
6     data: nxm numpy array. Rows are time points, columns are channels
7     sr: sampling rate, same time units as period
8     '''
9     from numpy import flip
10
11     samples = data.shape[0]
12     sensors = data.shape[1]
13
14     period = samples // sr
15
16     time = linspace(0, period, period*sr)
17
18     offset = 5 # for mean=0 and var=1 normalised data
19
20     # Calculate means and standard deviations of all columns
21     means = data.mean(axis=0)
22     stds = data.std(axis=0)
```

```
23
24     # Plot each series with an offset of 2 times the standard
      deviations
25     fig, ax = subplots(figsize=(7, 8))
26
27     ax.plot(time, (data - means)/stds + offset*arange(sensors-1,-1,-1))
      ;
28
29     ax.plot(time, zeros((samples, sensors)) + offset*arange(sensors
      -1,-1,-1), '--', color='gray');
30
31     ax.set_xlabel('Time')
32     ax.set_yticks(offset*arange(sensors))
33     ax.set_yticklabels(flip(arange(sensors)+1))
```

```
1 plot_series(data, sr);
2 show()
```

### How to create a function

```
1 def my_plot1(data):
2
3     fig, ax = subplots()
4
5     ax.plot(data)
```

```
1 my_plot1(data)
2 show()
```

```
1 def my_plot2(data, factor):
2     '''
3     this is just a test
4     '''
5
6     columns = data.shape[1]
7
8     offset = arange(columns)
9
10    fig, ax = subplots()
11
12    ax.plot(data + offset*factor)
```

```
1 my_plot2(data, 100)
2 show()
```

## FourierSpectrum

The Fourier spectrum decomposes the time series into a sum of sine waves. The spectrum gives the coefficients of each of the sine wave components. The coefficients are directly related to the amplitudes needed to optimally fit the sum of all sine waves to recreate the original data.

However, the assumption behind the Fourier transform is that the data are provided as in infinitely long stationary time series. These assumptions are not fulfilled as the data are finite and stationarity of a biological system can typically not be guaranteed. Thus, interpretation needs to be cautious.

We import the Fourier transform function `fft` from `scipy.fftpack` and can use it to transform all columns at the same time.

```
1 from pandas import read_csv
2 from matplotlib.pyplot import subplots, yticks, legend, rcParams, show
3 from numpy import arange, linspace, zeros
4
5 from scipy.fftpack import fft
```

```
1 df = read_csv("data/EEG_absence.txt", delim_whitespace=True)
2
3 sr = 256
4 duration = 5
5
6 df_np = df.to_numpy()
7
8 data = df_np[:duration*sr, :2] ## SF, needs explanation
9
10 df.head()
```

1		FP1	FP2	F3	F4	...	E02	EM1	EM2
			PHO						
2	0	-6.9732 14.5002	30.00060	60.9815	-23.047	...	20.8242	20.3583	21.1760
3	1	-15.1590 14.5056	22.85930	62.2845	-24.359	...	20.8289	20.3292	21.1118
4	2	-23.3680 14.5109	15.85860	63.2742	-25.353	...	20.8337	20.3120	21.0367
5	3	-31.5560 14.5161	9.05790	63.9646	-26.034	...	20.8327	20.3002	20.9580
6	4	-39.6840 14.5212	2.45328	64.4026	-26.451	...	20.8248	20.2862	20.8843
7									
8		[5 rows x 28 columns]							

```
1 data.shape
```

```
1 (1280, 2)
```

```
1 def plot_series(data, sr):
2     '''
3     Time series plot of multiple time series
4     Data are normalised to mean=0 and var=1
5
6     data: nxm numpy array. Rows are time points, columns are channels
7     sr: sampling rate, same time units as period
8
9     leg: Legend of figure, uses column index
10    '''
11
12    samples = data.shape[0]
13    sensors = data.shape[1]
14
15    period = samples // sr
16
17    time = linspace(0, period, period*sr)
18
19    offset = 5 # for mean=0 and var=1 normalised data
20
21    # Calculate means and standard deviations of all columns
22    means = data.mean(axis=0)
23    stds = data.std(axis=0)
24
25    # Plot each series with an offset of 2 times the standard
26    # deviations
27    fig, ax = subplots(figsize=(7, 5))
28    ax.plot(time, (data - means)/stds + offset*arange(sensors-1,-1,-1))
29    ;
30    ax.plot(time, zeros((samples, sensors)) + offset*arange(sensors
31    -1,-1,-1), '--', color='gray');
32
33    yticks([]);
34    ax.set(xlabel='Time')
```

```
1 plot_series(data[:, :2], sr)
2 show()
```

```
1 data_fft = fft(data, axis=0)
2
3 data_fft.shape
```

```
1 (1280, 2)
```

```
1 rows = data.shape[0]
2 freqs = (sr/2)*linspace(0, 1, rows//2)
```

```
3 amplitude = (2.0 / rows) * abs(data_fft[:rows//2, :])
4
5 fig, ax = subplots()
6
7 ax.plot(freqs, amplitude);
8
9 show()
```

```
1 fig, ax = subplots()
2
3 ax.plot(freqs, amplitude);
4
5 ax.set_xlim(0, 10);
6 ax.set_xlabel('Frequency (Hz)', fontsize=20)
7 ax.set_ylabel('Amplitude (abs)', fontsize=20);
8
9 show()
```

### Keypoints

- 
- 
-