-
-
-
-
-
-

Plotting NumPy series

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

```
from pandas import read_csv

from matplotlib.pyplot import subplots, show

from numpy import arange, linspace, zeros
```

```
df = read_csv("data/EEG_background.txt", delim_whitespace=True)

df.head()
```

```
        FP1      FP2        F3       F4  ...      EO2      EM1      EM2
        PHO
0  -7.4546  22.8428   6.28159  15.6212  ...  13.7021  12.9109  13.7034
    9.37573
1 -11.1060  21.4828   6.89088  15.0562  ...  13.7942  13.0194  13.7628
    9.44731
2 -14.4000  20.0907   7.94856  14.1624  ...  13.8982  13.1116  13.8239
    9.51796
3 -17.2380  18.7206   9.36857  13.0093  ...  14.0155  13.1927  13.8914
    9.58770
4 -19.5540  17.4084  11.06040  11.6674  ...  14.1399  13.2692  13.9652
    9.65654

[5 rows x 28 columns]
```

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

```
df.shape
```

```
(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

```
sr = 256          # Sampling rate: 1 / seconds

duration = 5      # seconds

df_np = df.to_numpy()

data = df_np[:duration*sr, :19] ## SF Comment, needs to explain this
    above

data.shape
```

```
(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.

```
def plot_series(data, sr):
    '''
    Time series plot of multiple time series
    Data are normalised to mean=0 and var=1

    data: nxm numpy array. Rows are time points, columns are channels
    sr: sampling rate, same time units as period
    '''
    from numpy import flip

    samples = data.shape[0]
    sensors = data.shape[1]

    period = samples // sr

    time = linspace(0, period, period*sr)

    offset = 5 # for mean=0 and var=1 normalised data

    # Calculate means and standard deviations of all columns
    means = data.mean(axis=0)
    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.

```
from pandas import read_csv
from matplotlib.pyplot import subplots, yticks, legend, rcParams, show
from numpy import arange, linspace, zeros

from scipy.fftpack import fft
```

```
df = read_csv("data/EEG_absence.txt", delim_whitespace=True)

sr = 256
duration = 5

df_np = df.to_numpy()

data = df_np[:duration*sr, :2] ## SF, needs explanation

df.head()
```

```
        FP1       FP2       F3       F4  ...       EO2       EM1       EM2
                PHO
0  -6.9732  30.00060  60.9815 -23.047  ...  20.8242  20.3583  21.1760
    14.5002
1 -15.1590  22.85930  62.2845 -24.359  ...  20.8289  20.3292  21.1118
    14.5056
2 -23.3680  15.85860  63.2742 -25.353  ...  20.8337  20.3120  21.0367
    14.5109
3 -31.5560   9.05790  63.9646 -26.034  ...  20.8327  20.3002  20.9580
    14.5161
4 -39.6840   2.45328  64.4026 -26.451  ...  20.8248  20.2862  20.8843
    14.5212

[5 rows x 28 columns]
```

```
data.shape
```

```
(1280, 2)
```

```
def plot_series(data, sr):
    '''
    Time series plot of multiple time series
    Data are normalised to mean=0 and var=1

    data: nxm numpy array. Rows are time points, columns are channels
    sr: sampling rate, same time units as period

    leg: Legend of figure, uses column index
    '''

    samples = data.shape[0]
    sensors = data.shape[1]

    period = samples // sr

    time = linspace(0, period, period*sr)

    offset = 5 # for mean=0 and var=1 normalised data

    # Calculate means and standard deviations of all columns
    means = data.mean(axis=0)
    stds = data.std(axis=0)

    # Plot each series with an offset of 2 times the standard
       deviations
    fig, ax = subplots(figsize=(7, 5))

    ax.plot(time, (data - means)/stds + offset*arange(sensors-1,-1,-1))
       ;

    ax.plot(time, zeros((samples, sensors)) + offset*arange(sensors
       -1,-1,-1),'--',color='gray');

    yticks([]);

    ax.set(xlabel='Time')
```

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

```
data_fft = fft(data, axis=0)

data_fft.shape
```

```
(1280, 2)
```

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

```
amplitude = (2.0 / rows) * abs(data_fft[:rows//2, :])

fig, ax = subplots()

ax.plot(freqs, amplitude);

show()
```

```
fig, ax = subplots()

ax.plot(freqs, amplitude);

ax.set_xlim(0, 10);
ax.set_xlabel('Frequency (Hz)', fontsize=20)
ax.set_ylabel('Amplitude (abs)', fontsize=20);

show()
```

Keypoints

-
-
-