Time Series

Last updated on 2024-08-06 | Edit this page

Download Chapter notebook (ipynb)

Mandatory Lesson Feedback Survey

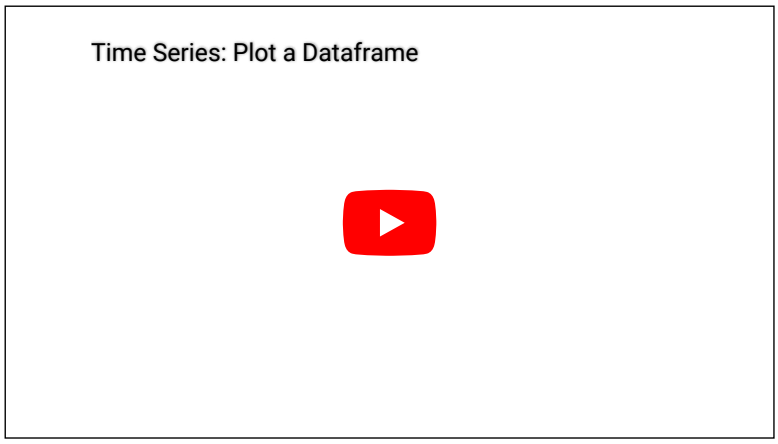
OVERVIEW

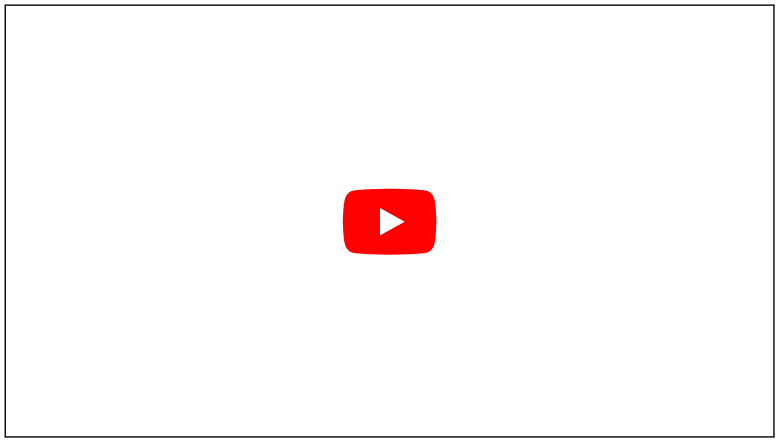
Questions

- How is timeseries data visualised?
- Why do we need to tidy up/filter the data?
- How to study correlation among timeseries data points?

Objectives

- Learning ways to display multiple time series.
- Understanding why filtering is needed.
- Explaining fourier spectrum of time series.
- Knowledge of correlation matrix of time series.





PREREQUISITES

- Dataframes 1 and 2
- Image Handling
- Basics of Numpy Arrays

PYTHON

```
from pandas import read_csv

from numpy import arange, zeros, linspace, sin, pi, c_, mean, var, array
from numpy import correlate, corrcoef, fill_diagonal, amin, amax, asarray
from numpy import around
from numpy.ma import masked_less, masked_greater

from matplotlib.pyplot import subplots, yticks, legend, axis, figure, show
```

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.

PYTHON

```
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 recordings
    sr: sampling rate, same time units as period
    '''

    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
    fig, ax = subplots(figsize=(7, 8))

    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([]);

    names = [str(x) for x in range(sensors)]
    legend(names)

    ax.set(xlabel='Time')

    axis('tight');

    return fig, ax
```

Example: Normal and Pathological EEG

As an example, let us import two sets of time series data and convert them to Numpy arrays, here called *data_back* and *data_epil*. They represent human electroencephalogram (EEG) as recorded during normal *background* activity and during an epileptic seizure called *absence* seizure.

PYTHON

```
df_back = read_csv("data/EEG_background.txt", delim_whitespace=True)
df_epil = read_csv("data/EEG_absence.txt", delim_whitespace=True)

sr = 256     # 1 / seconds
period = 6   # seconds
channels = 10

d1 = df_back.to_numpy()
d2 = df_epil.to_numpy()

data_back = d1[:period*sr:, :channels]
data_epil = d2[:period*sr:, :channels]
```

The `read_csv` function is used with the keyword argument `delim_whitespace`. When set to True, this allows to import data that are space-separated (rather than comma-separated). If you check the data .txt files, you will see that the numbers (which represent voltages) are indeed separated by space, not comma.

Next, three constants are assigned: The sampling rate, sr, which is given in number of samples recorded per seconds; the duration of the recording, period, which is given in seconds; and the number of columns, referred to as channels, to be extracted from the recording. We use the first 10 columns in this lesson.

The data are then converted from a data frame to Numpy arrays.

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

PYTHON

```
df_back.head()
```

OUTPUT

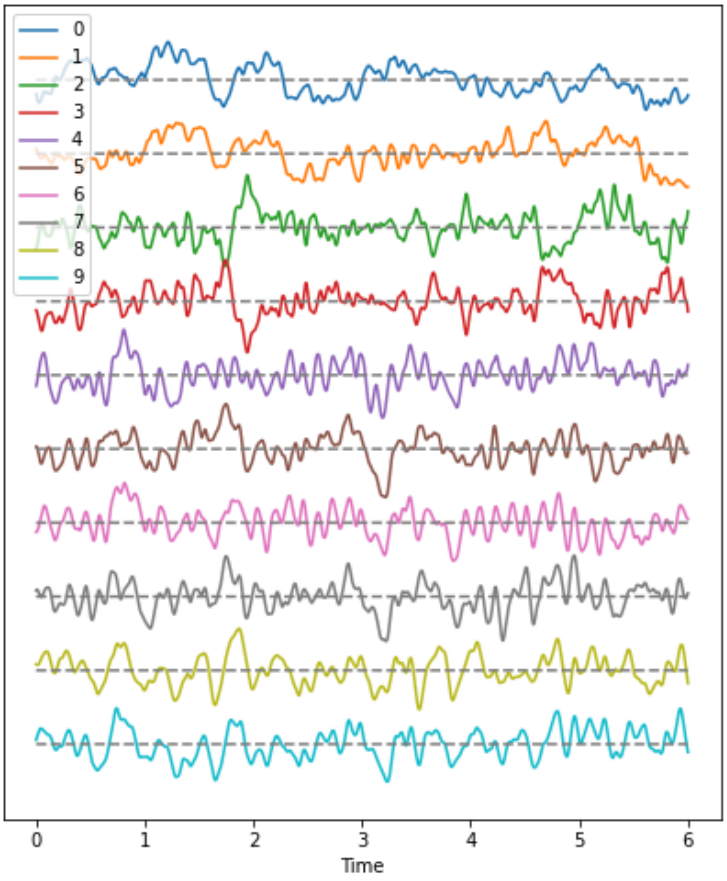
```
       FP1      FP2       F3       F4  ...      EO2      EM1      EM2      PHO
0  -7.4546  22.8428   6.28159  15.6212  ...  13.7021  12.9109  13.7034  9.37573
1 -11.1060  21.4828   6.89088  15.0562  ...  13.7942  13.0194  13.7628  9.44731
2 -14.4000  20.0907   7.94856  14.1624  ...  13.8982  13.1116  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]
```

The row indices and column names for the seizure data look the same. The names of the recording channels are from the commonly used 10-20 system to record voltages of brain activity from the scalp in humans. E.g. 'F' stands for the frontal lobe.

We can now use the plot function from above to plot the data. Note from the definition (first line) that two input arguments are required: the data set and the sampling rate.

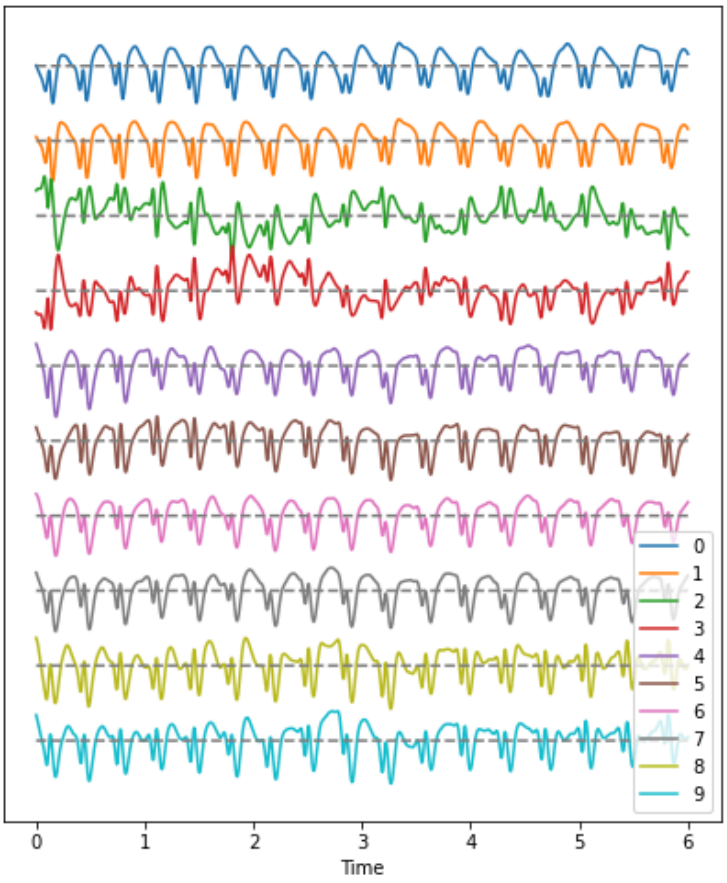
PYTHON

```
plot_series(data_back, sr)
show()
```



PYTHON

```
plot_series(data_epil, sr);
show()
```



Observations

1. Background:

- There are irregular oscillations of all recorded brain potentials.
- Oscillations recorded at different locations above the brain differ.
- Oscillations are not stable but modulated over time.
- There are different frequency components in each trace.

2. Epileptic Seizure:

- There are regular oscillations.
- Oscillations recorded at different locations are not identical but similar or at least related in shape.
- Despite some modulation, oscillations are fairly stable over time.
- There are repetitive motifs composed of two major components throughout the recording, a sharp *spike* and a slow *wave*.

Task

Quantify features of these time series data to get an overview. As a univariate feature we can use the frequency content. This takes into account the fact that the rows (or samplles) are not independent of each other but are organised along the time axis. In consequence, there are correlations between data points along the rows of each column and the **Fourier spectrum** can be used to identify these.

The Fourier spectrum assumes that the data are stationary and can be thought of as a superposition of regular sine waves with different frequencies. Its output will show which of the frequencies are present in the data and also their respective amplitudes.

As a bivariate feature we can use the cross-correlation matrix.

Work Through Example

Check the Numpy array containing the background and seizure data.

PYTHON

```
print(data_back.shape, data_epil.shape)
```

OUTPUT

```
(1536, 10) (1536, 10)
```

There are 1536 rows and 10 columns.

Display data with offset

Take a look at the code of the function `plot_series` that creates the time series plot. It requires the input of a data file where the row index is interpreted as time. In addition, the sampling rate sr is required to be able to extract the time scale. The sampling rate specifies the number of samples recorded per unit time.

The sensors or recording channels are assumed to be in the columns.

PYTHON

```
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
    '''

    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
    fig, ax = subplots(figsize=(7, 8))

    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([]);

    names = [str(x) for x in range(sensors)]
    legend(names)

    ax.set(xlabel='Time')

    axis('tight');

    return fig, ax
```

The declaration syntax `def` is followed by the function name and, in parenthesis, the input arguments. It is completed with a colon.

Following the declaration is the documentation of the function.

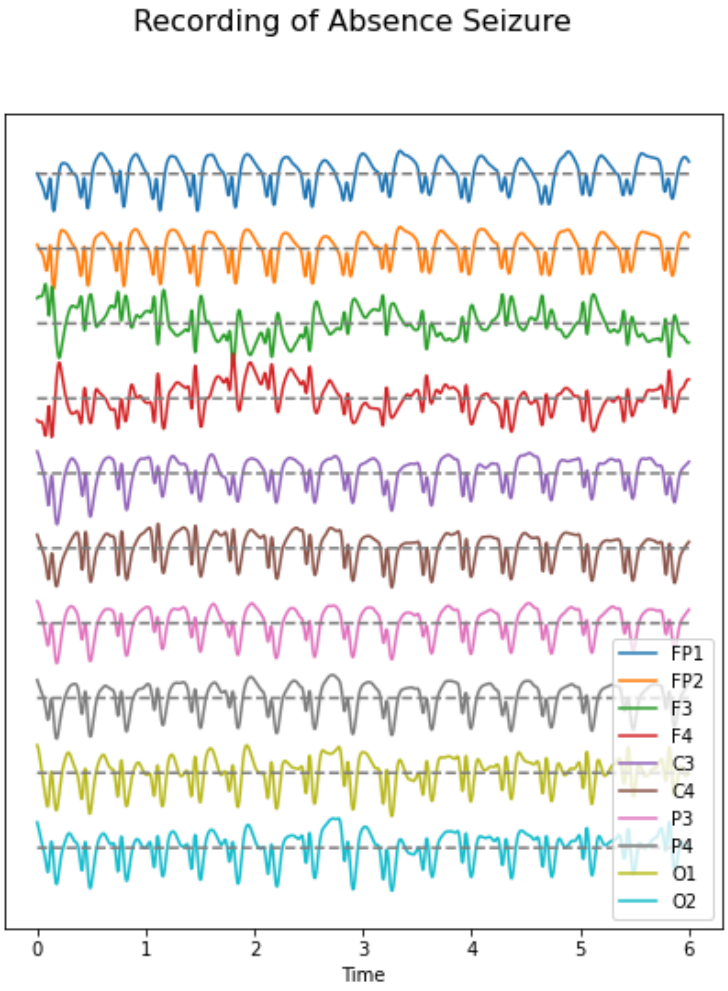
Next comes the function code, all indented.

The function closes with the optional output syntax `return` and any number of returned variables, anything that might be used as a product of running the function.

In our example, the figure environment and the coordinate system are ‘returned’, and can in principle be used to further modify the plot.

Here is how to call the function and then add a title and the sensor names to the display.

```python
(fig, ax) = plot_series(data_epil, sr)

names = df_back.columns[:channels]

fig.suptitle('Recording of Absence Seizure', fontsize=16);

legend(names);

show()
```



A function usually (but not necessarily) takes in one or several variables or values, processes them, and produces a specific result. The variable(s) given to a function and those produced by it are referred to as input arguments, and outputs respectively.

There are different ways to create functions in Python. In this course, we will be using `def` to implement our own functions. This is the easiest, and by far the most common method for declaring functions. The structure of a typical function defined using `def` can be see in the `plot_series` example:

There are several points to remember in relation to functions:

- The name of a function follows same principles as that of any other variable. It must be in lower-case characters.
- The input arguments of a function, e.g. *data* and *sr* in the example, are essentially variables whose scope is the function. That is, they are only accessible within the function itself, and not from outside the function.

- Variables defined inside of a function, should not use the same name as variables defined outside. Otherwise they may override each other.

It is important for a function to only perform one specific task. As such it can be used independent of the current context. Try to avoid incorporating separable tasks into a single function.

Once you start creating functions for different purposes you can start to build your own library of ready-to-use functions to address different needs. This is the primary principle of a popular programming paradigm known as functional programming.

Filtering

Data sets with complex waveforms contain many different components which may or may not be relevant for a specific question.
Data filtering is applied to take out specific components. Component in this context refers to 'frequency', i.e. the number of cycles per unit of time. Thus a small number refers to low frequencies with long periods (cycles) and a large number to high frequencies with short periods.

Let us see a simple example how both low- and high-frequency components can be filtered (suppressed) in the example time series.

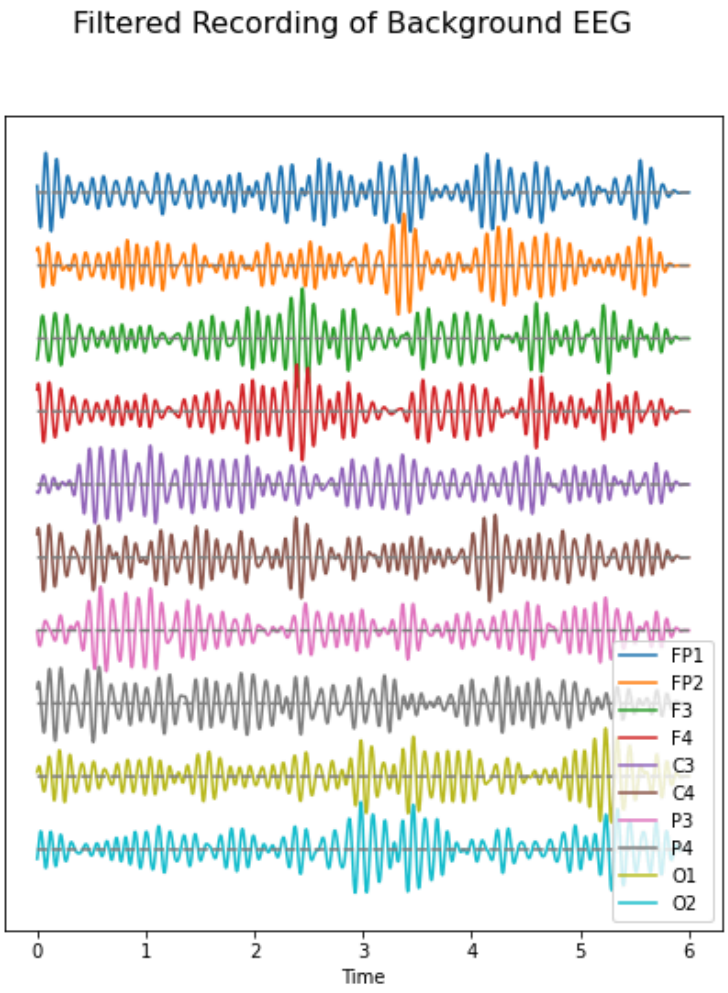
Here is a simple function which takes two additional input arguments, the low and the high cut-off.

PYTHON

```python
def data_filter(data, sr, low, high):
    """
    Filtering of multiple time series.

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

    low:  Low cut-off frequency (high-pass filter)
    high: High cut-off frequency (low-pass filter)

    return: filtered data
    """

    from scipy.signal import butter, sosfilt

    order = 5

    filter_settings = [low, high, order]

    sos = butter(order, (low,high), btype='bandpass', fs=sr, output='sos')

    data_filtered = zeros((data.shape[0], data.shape[1]))

    for index, column in enumerate(data.transpose()):
        forward = sosfilt(sos, column)
        backwards = sosfilt(sos, forward[-1::-1])
        data_filtered[:, index] = backwards[-1::-1]

    return data_filtered
```

PYTHON

```
data_back_filt = data_filter(data_back, sr, 8, 13)

(fig, ax) = plot_series(data_back_filt, sr)

fig.suptitle('Filtered Recording of Background EEG', fontsize=16);

legend(names);

show()
```



The frequency range from 8 to 13 Hz is referred to as alpha band in the electroencephalogram. It is thought that this represents a kind of idling rhythm of the brain, i.e. an activity where the brain is not actively processing sensory input.

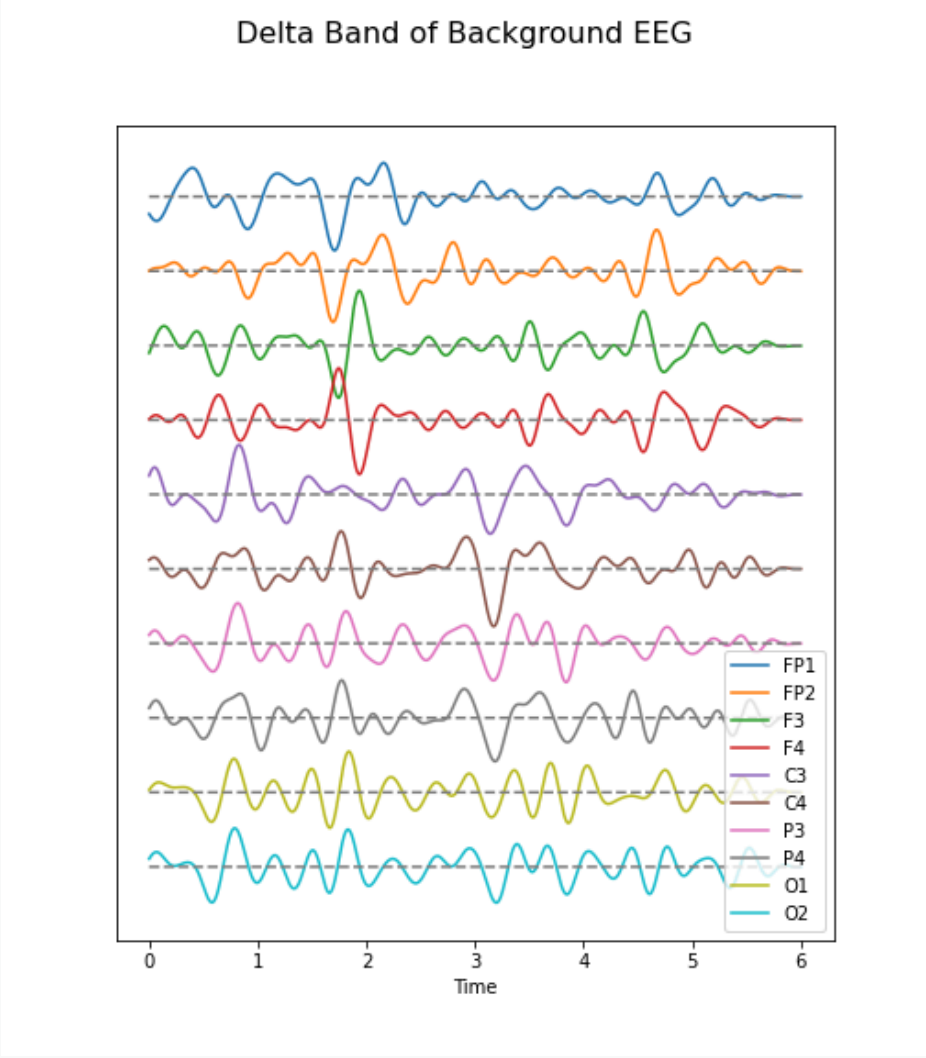
DIY1: BAND-PASS FILTERED DATA

Create figures of the delta (1-4 Hz) band for both the background and the seizure EEG. Note the differences.

Solution

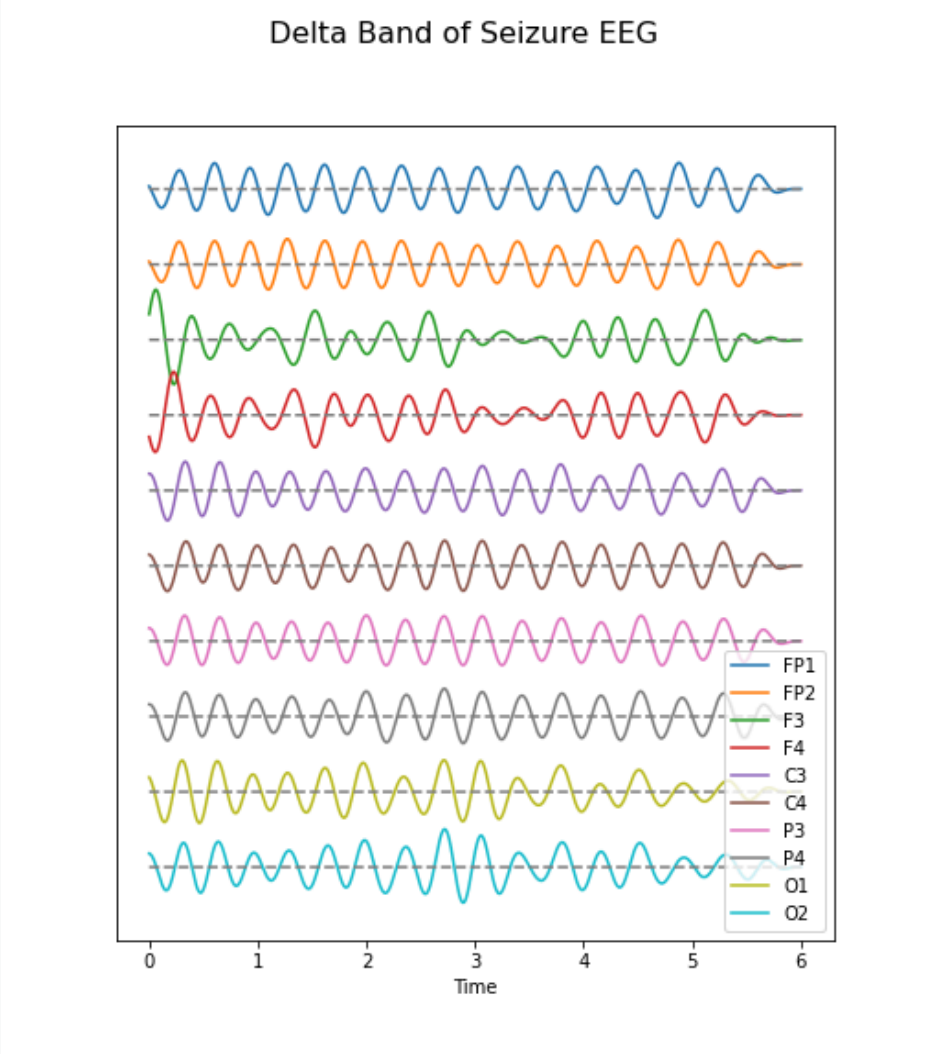
PYTHON

```
data_back_filt = data_filter(data_back, sr, 1, 4)

(fig, ax) = plot_series(data_back_filt, sr)

fig.suptitle('Delta Band of Background EEG', fontsize=16);

legend(names);

show()
```



PYTHON

```
data_epil_filt = data_filter(data_epil, sr, 1, 4)

(fig, ax) = plot_series(data_epil_filt, sr)

fig.suptitle('Delta Band of Seizure EEG', fontsize=16);

legend(names);

show()
```



Fourier Spectrum

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.

Fourier Transform of EEG data

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

PYTHON

```
from scipy.fftpack import fft

data_back_fft = fft(data_back, axis=0)
```

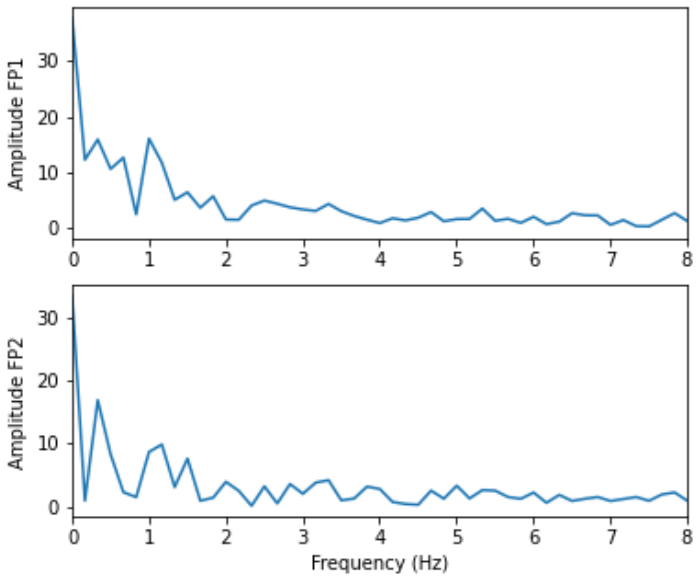
To plot the results, a couple of steps are required.
First, we obtain a Fourier spectrum for every data column, so we need to define how many plots we want to have. If we take only columns, we can display them all in one go.

Second, the Fourier transform results in twice the number of complex coefficients (positive and negative) of which we only need the first half.

Third, the Fourier transform outputs complex numbers. To display the 'amplitude' of a frequency (the coefficient corresponding to the amplitude of the sine wave with that frequency) we take the absolute value of the complex numbers.

PYTHON

```
no_win = 2

rows = data_back.shape[0]

freqs = (sr/2)*linspace(0, 1, int(rows/2))

amplitudes_back = (2.0 / rows) * abs(data_back_fft[:rows//2, :2])


fig, axes = subplots(figsize=(6, 5), ncols=1, nrows=no_win, sharex=False)

names = df_back.columns[:2]

for index, ax in enumerate(axes.flat):
    axes[index].plot(freqs, amplitudes_back[:, index])
    axes[index].set_xlim(0, 8)
    axes[index].set(ylabel=f'Amplitude {names[index]}')

axes[index].set(xlabel='Frequency (Hz)');

show()
```



We can see that in these two channels, the main amplitude contributions lie in the low frequencies, below 2 Hz.

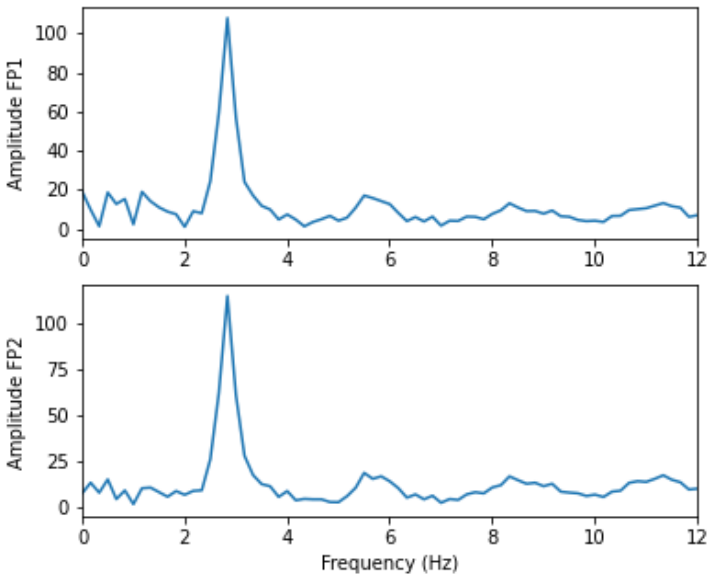
Let us compare the corresponding figure for the case of seizure activity:

PYTHON

```
data_epil_fft = fft(data_epil, axis=0)
```

PYTHON

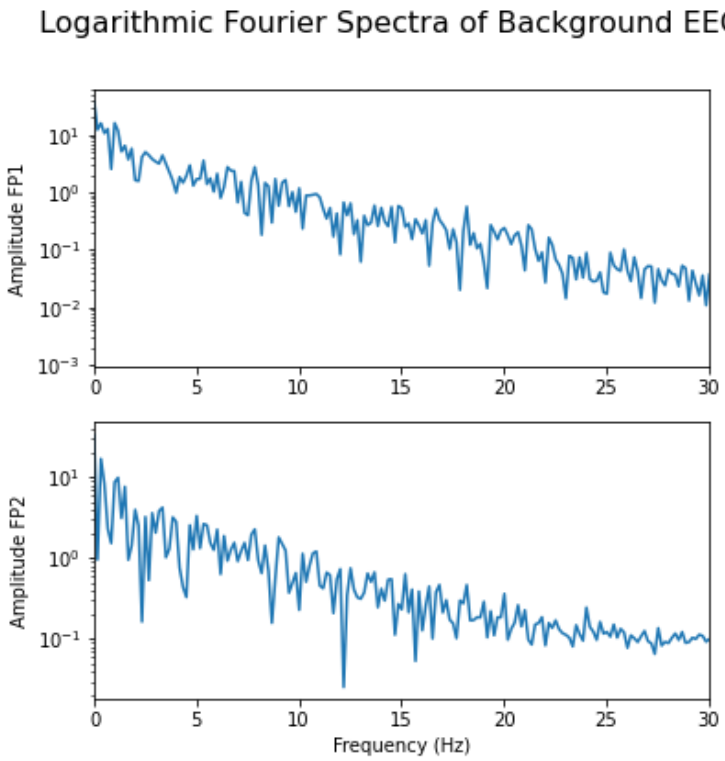
```
fig, axes = subplots(figsize=(6, 5), ncols=1, nrows=no_win, sharex=False)

names = df_epil.columns[:2]

amplitudes_epil = (2.0 / rows) * abs(data_epil_fft[:rows//2, :2])

for index, ax in enumerate(axes.flat):
    axes[index].plot(freqs, amplitudes_epil[:, index])
    axes[index].set_xlim(0, 12)
    axes[index].set(ylabel=f'Amplitude {names[index]}')

axes[index].set(xlabel='Frequency (Hz)');

show()
```



The main frequency of the epileptic rhythm is between 2 and 3 Hz.
As we can see above in the Fourier spectra above, the amplitudes are high for low frequencies and tend to decrease with increasing frequency. Sometimes it is useful to see the high frequencies enhanced. This can be achieved with a logarithmic plot of the powers.

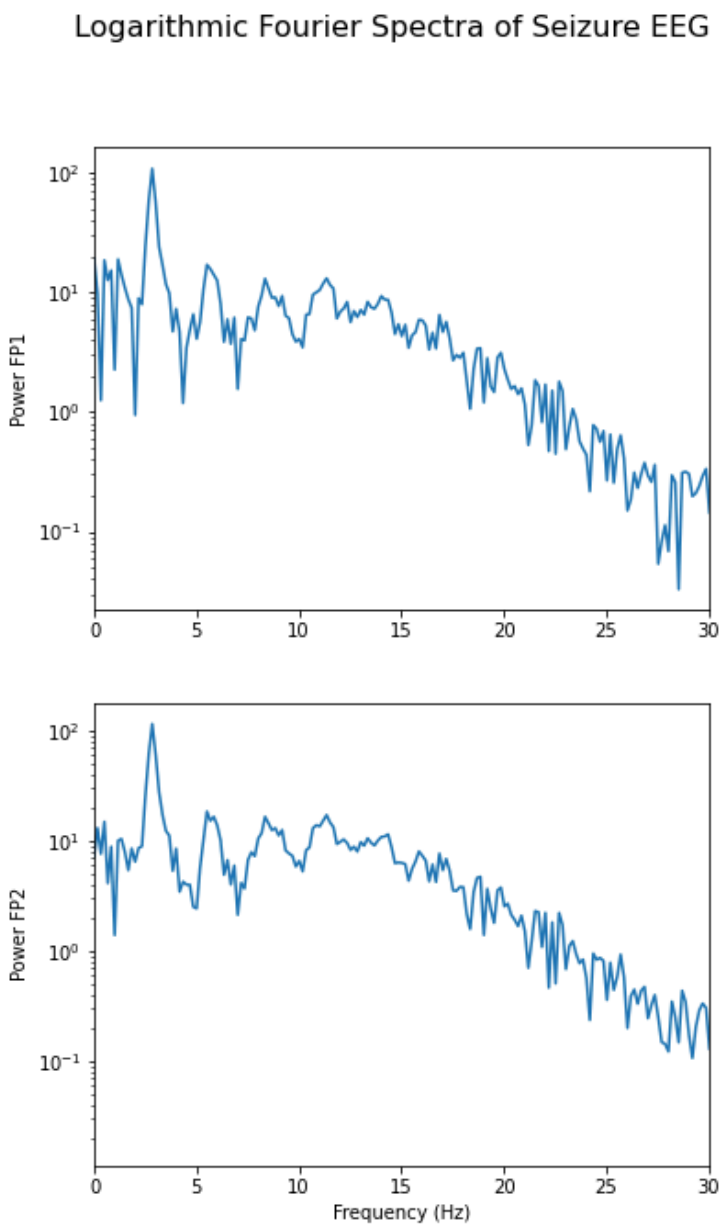
PYTHON

```
fig, axes = subplots(figsize=(6, 6), ncols=1, nrows=no_win, sharex=False)

for index, ax in enumerate(axes.flat):

    axes[index].plot(freqs, amplitudes_back[:, index])
    axes[index].set_xlim(0, 30)
    axes[index].set(ylabel=f'Amplitude {names[index]}')
    axes[index].set_yscale('log')

axes[no_win-1].set(xlabel='Frequency (Hz)');
fig.suptitle('Logarithmic Fourier Spectra of Background EEG', fontsize=16);

show()
```



And for the seizure data:

```python
fig, axes = subplots(figsize=(6, 10), ncols=1, nrows=no_win, sharex=False)

for index, ax in enumerate(axes.flat):

    axes[index].plot(freqs, amplitudes_epil[:, index])
    axes[index].set_xlim(0, 30)
    axes[index].set(ylabel=f'Power {names[index]}')
    axes[index].set_yscale('log')

axes[no_win-1].set(xlabel='Frequency (Hz)');
fig.suptitle('Logarithmic Fourier Spectra of Seizure EEG', fontsize=16);

show()
```



In the spectrum of the absence data, it is now more obvious that there are further maxima at 6, 9, 12, and perhaps 15Hz. These are integer multiples or 'harmonics' of the basic frequency at around 3Hz, also referred to as the fundamental frequency.

A feature that can be used as a summary statistic, is to caclulate the **band power** for each channel. The band power can be obtained as the sum of all powers within a specified range of frequencies, also called the 'band'. The band power thus represents a single number.

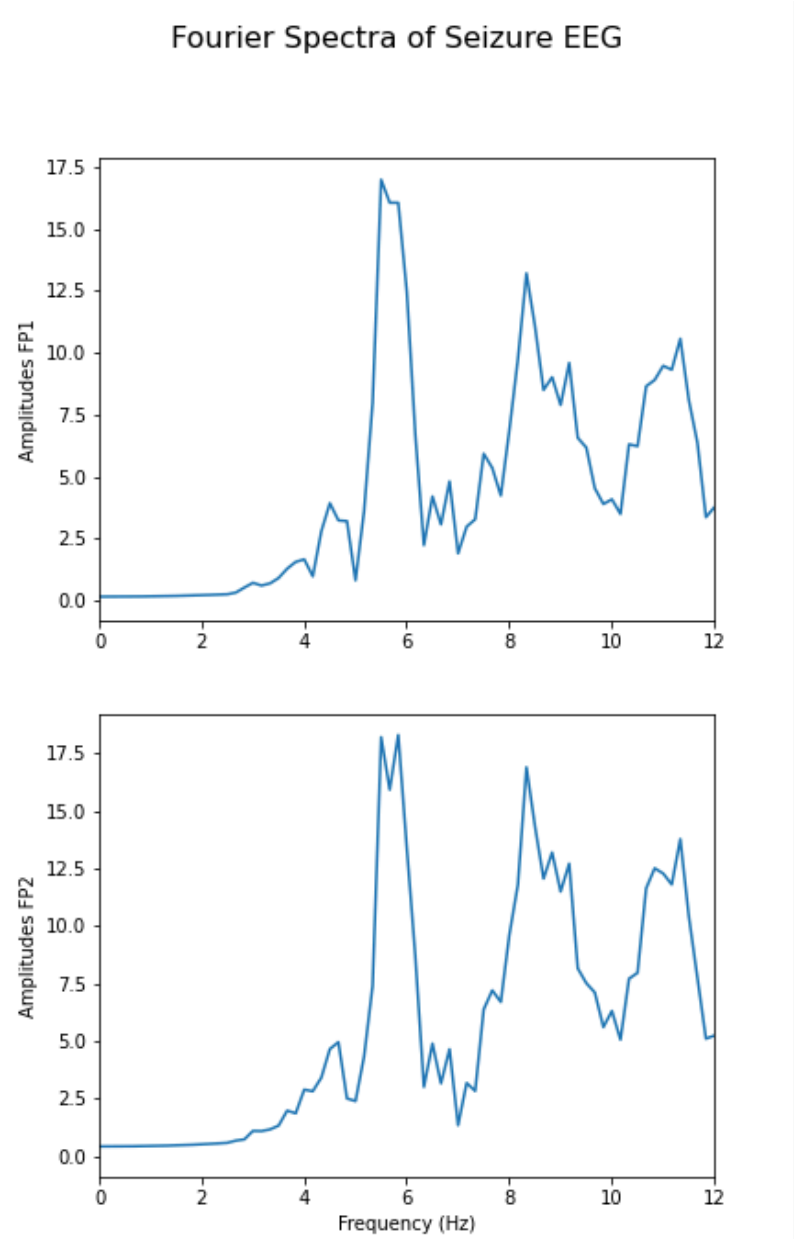
DIY2: FOURIER SPECTRA OF FILTERED DATA

Calculate and display the Fourier spectra of the first two channels filtered between 4 and 12 Hz for the absence seizure data. Can you find harmonics?

Solution

PYTHON

```
data_epil_filt = data_filter(data_epil, sr, 4, 12)

data_epil_fft = fft(data_epil_filt, axis=0)

rows = data_epil.shape[0]

freqs = (sr/2)*linspace(0, 1, int(rows/2))

amplitudes_epil = (2.0 / rows) * abs(data_epil_fft[:rows//2, :no_win])

fig, axes = subplots(figsize=(6, 10), ncols=1, nrows=no_win, sharex=False)

for index, ax in enumerate(axes.flat):
    axes[index].plot(freqs, amplitudes_epil[:, index])
    axes[index].set_xlim(0, 12)
    axes[index].set(ylabel=f'Amplitudes {names[index]}')
axes[no_win-1].set(xlabel='Frequency (Hz)');

fig.suptitle('Fourier Spectra of Seizure EEG', fontsize=16);

show()
```

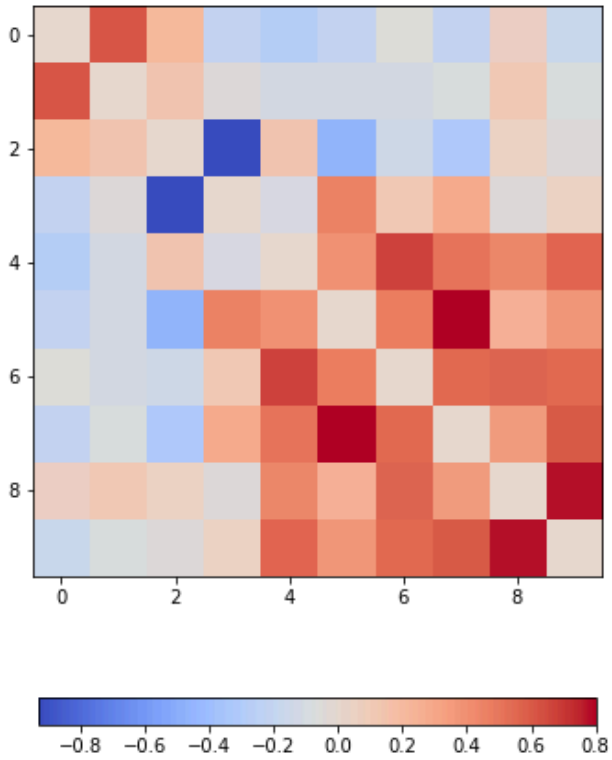


Cross-Correlation Matrix

As one example of a multivariate analysis of time series data, we can calculate the cross-correlation matrix.

Here it is done for the background:

```python
corr_matrix_back = corrcoef(data_back, rowvar=False)

fill_diagonal(corr_matrix_back, 0)

fig, ax = subplots(figsize = (8,8))

im = ax.imshow(corr_matrix_back, cmap='coolwarm');

fig.colorbar(im, orientation='horizontal', shrink=0.68);

show()
```



The diagonal is set to zero. This is done to improve the visual display. If left to one, the diagonal tends to dominate the visual impression even though it is trivial and nothing can be learned from it.

Looking at the non-diagonal elements, we find:

- two strongly correlated series (indices 5 and 7)
- two strongly anti-correlated series (indices 3 and 4)
- a block of pronounced correlations between series with indices 4 through 9)

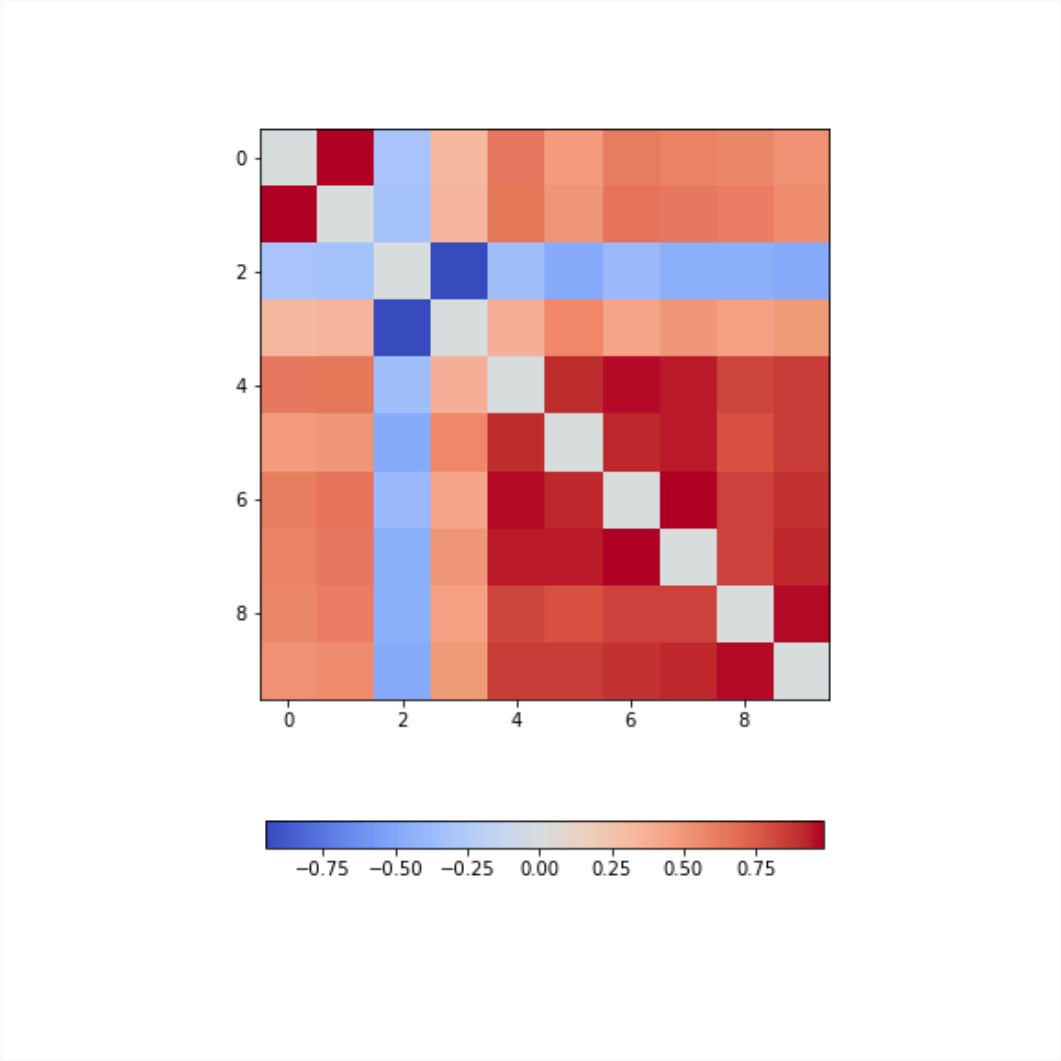
DIY3: DISPLAY THE CORRELATION MATRIX FOR THE SEIZURE DATA

Calculate the correlation matrix for the seizure data and compare the correlation pattern to the one from the background data.

Solution

PYTHON

```
corr_matrix_epil = corrcoef(data_epil, rowvar=False)

fill_diagonal(corr_matrix_epil, 0)

fig, ax = subplots(figsize = (8,8))

im = ax.imshow(corr_matrix_epil, cmap='coolwarm');

fig.colorbar(im, orientation='horizontal', shrink=0.68);

show()
```



We find - a number of pairs of strongly correlated series - two strongly anti-correlated series (in- dices 3 and 4) - a block of pronounced correlations between series with indices 4 through 9).

So interestingly, while the time series changes dramatically in shape, the correlation pattern still shows some qualitative resemblance.

All results shown so far, represent the recording of the segment of 6 seconds chosen at the beginning. The human brain produces time-dependent voltage changes 24 hours a day and as seeing only a few seconds is only a partial view. The next step to investigate is to show how the features found for one segment vary over time.

Exercises

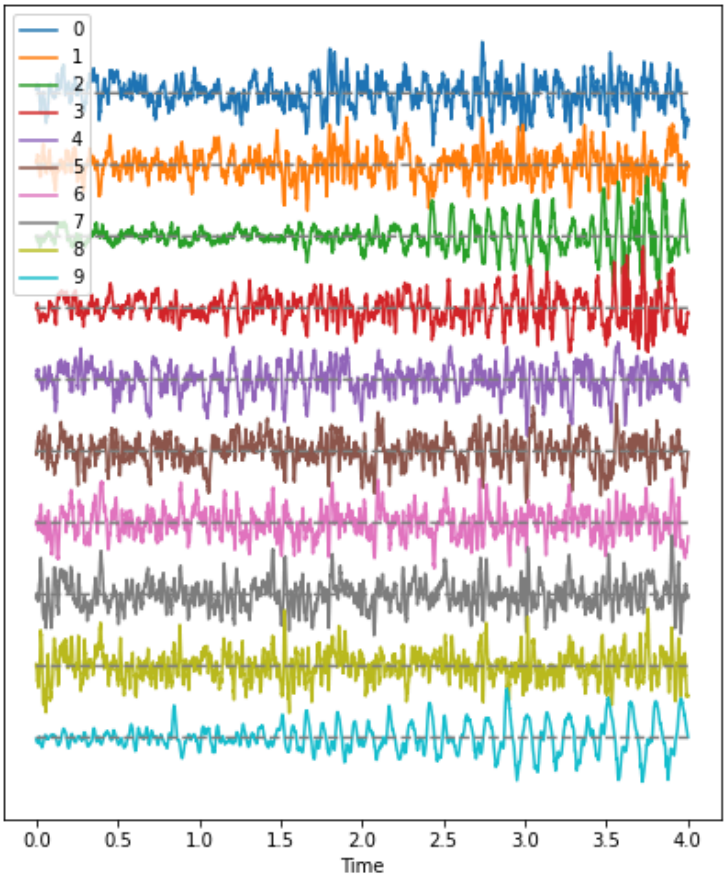
END OF CHAPTER EXERCISES

Pathological Human Brain Rhythms

Look at the image of brain activity from a child at the start of an epileptic seizure. It shows 4 seconds of evolution of the first 10 channels of a seizure rhythm at sampling rate sr=1024.

PYTHON

```
path = 'data/P1_Seizure1.csv'

data = read_csv(path, delimiter=r"\s+")

data_P1 = data.to_numpy()

sr = 1024

period = 4

channels = 10

plot_series(data_P1[:sr*period, :channels], sr);

show()
```



Using the code from this lesson, import the data from the file `P1_Seizure1.csv` and generate an overview of uni- and multivariate features in the following form:

1. Pick the first two seconds of the recording as background, the last two second as epileptic seizure rhythm. Use the first ten channels in both cases. Data should have the shape (2048, 10).
2. Filter the data to get rid of frequencies below 1 Hz and frequencies faster than 20 Hz.
3. Plot time series of both.
4. Fourier transform both filtered data sets and display the Fourier spectra of the first 4 channels. What are the strongest frequencies in the two sets?
5. Plot the correlation matrices of both data sets. Which channels show the strongest change in correlations?

Solution

PYTHON

```
from pandas import read_csv

from numpy import arange, zeros, linspace, sin, pi, c_, mean, var, array

from numpy import correlate, corrcoef, fill_diagonal, amin, amax, asarray

from numpy import around, triu_indices

from numpy.ma import masked_less, masked_greater

from scipy.fftpack import fft

from matplotlib.pyplot import subplots, yticks, legend, axis, figure, show
```

Q1

Extract data for first and last two seconds

PYTHON

```
sr = 1024

duration = 2 # seconds
data_n = data.to_numpy()

# dat_back = data.iloc[:duration*sr, :10]
dat_back = data_n[:duration*sr, :10]
dat_epil = data_n[data_n.shape[0] - duration*sr:, :10]

time = linspace(0, 1, duration*sr)

print(dat_back.shape, dat_epil.shape, time.shape)
```

OUTPUT

```
(2048, 10) (2048, 10) (2048,)
```

Q2 and Q3

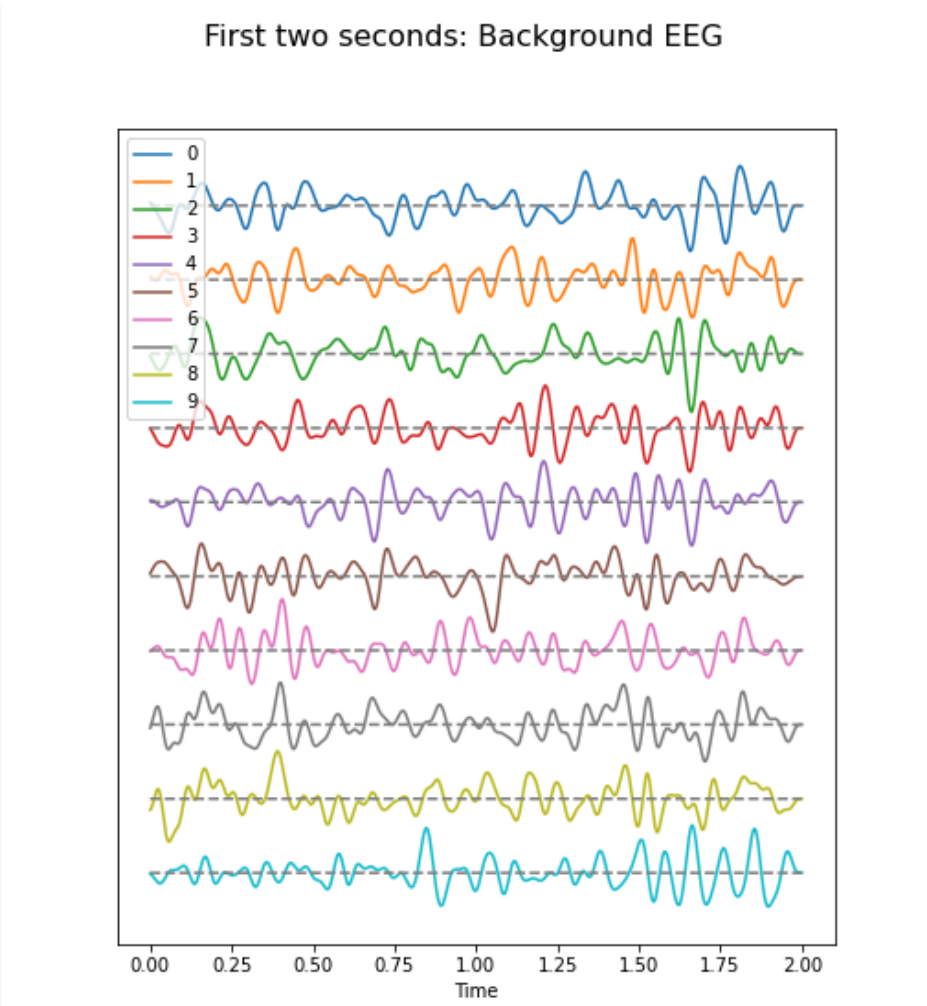
Filter and display data

PYTHON

```python
def data_filter(data, sr, low, high):
    """
    Filtering of multiple time series.

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

    low:  Low cut-off frequency (high-pass filter)
    high: High cut-off frequency (low-pass filter)

    return: filtered data
    """

    from scipy.signal import butter, sosfilt

    order = 5

    filter_settings = [low, high, order]

    sos = butter(order, (low,high), btype='bandpass', fs=sr, output='sos')

    data_filtered = zeros((data.shape[0], data.shape[1]))

    for index, column in enumerate(data.transpose()):
        forward = sosfilt(sos, column)
        backwards = sosfilt(sos, forward[-1::-1])
        data_filtered[:, index] = backwards[-1::-1]

    return data_filtered
```

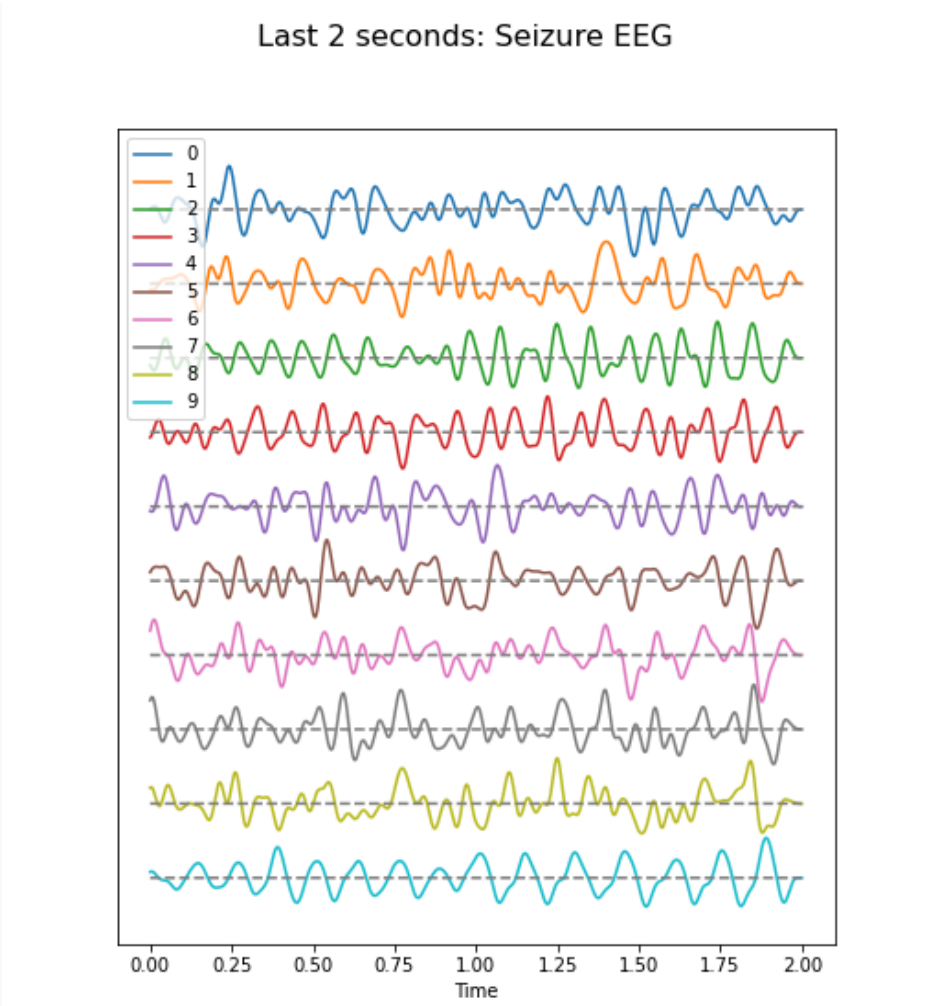
PYTHON

```python
dat_back_filt = data_filter(dat_back, sr, 1, 20)

(fig, ax) = plot_series(dat_back_filt, sr)

fig.suptitle('First two seconds: Background EEG', fontsize=16);

show()
```



PYTHON

```
dat_epil_filt = data_filter(dat_epil, sr, 1, 20)

(fig, ax) = plot_series(dat_epil_filt, sr)

fig.suptitle('Last 2 seconds: Seizure EEG', fontsize=16);

show()
```



Q4

Fourier Transform

PYTHON

```
rows = dat_back.shape[0]

dat_back_fft = fft(dat_back_filt, axis=0)

powers_back2 = (2.0 / rows) * abs(dat_back_fft[:rows//2, :])

dat_epil_fft = fft(dat_epil_filt, axis=0)

powers_epil2 = (2.0 / rows) * abs(dat_epil_fft[:rows//2, :])
```

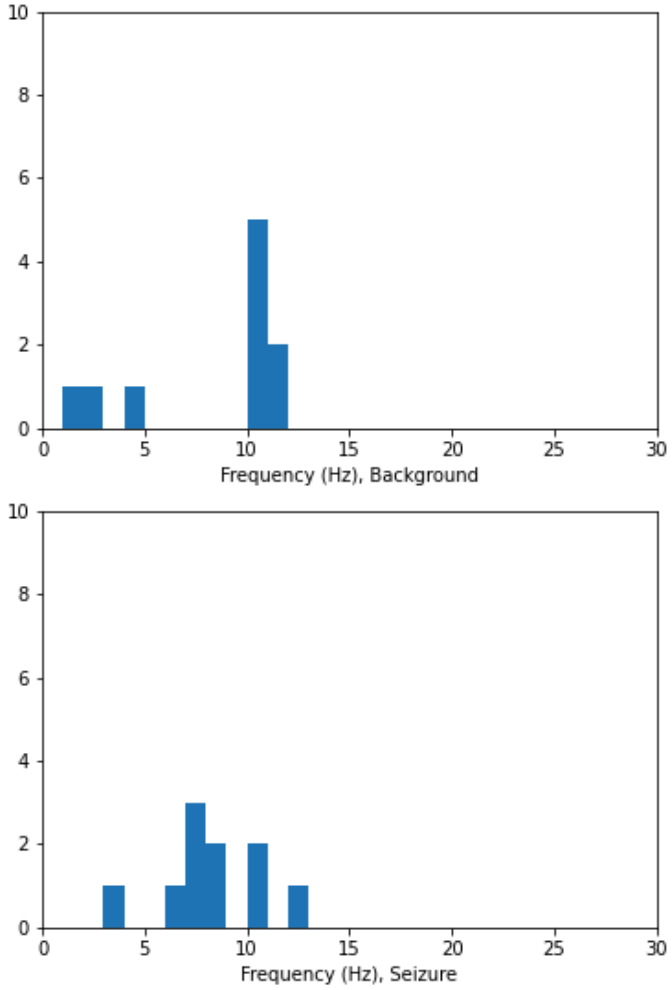
You can see the frequencies with largest power visually from a display of the Fourier spectrum. Here is an example code how to extract those values for each channel using Numpy functions.

PYTHON

```
# Get the maxima of powers for each channel

powermax_back2 = amax(powers_back2, axis=0)
powermax_epil2 = amax(powers_epil2, axis=0)


# Get the frequency index of that maximum

powermax_back2_index = zeros(powers_back2.shape[1])

for ind, pow in enumerate(list(powers_back2.transpose())):
    pow_ind = list(pow).index(powermax_back2[ind])
    powermax_back2_index[ind] = pow_ind

powermax_epil2_index = zeros(powers_epil2.shape[1])

for ind, pow in enumerate(list(powers_epil2.transpose())):
    pow_ind = list(pow).index(powermax_epil2[ind])
    powermax_epil2_index[ind] = pow_ind



powermax_back2_freq = asarray(powermax_back2_index)*(sr / 2 / powers_back2.shape[0])
powermax_epil2_freq = asarray(powermax_epil2_index)*(sr / 2 / powers_epil2.shape[0])

print('Frequencies of max power in background (Hz): ', '\n', around(powermax_back2_freq, decimals=1))
print('')
print('Frequencies of max power in seizure (Hz): ', '\n', around(powermax_epil2_freq, decimals=1))

#print(powermax_back2_freq, '\n',  powermax_epil2_freq)

fig, ax = subplots(nrows=2, figsize=(6,9))

binwidth = 1

(counts1, bins1, bars1) = ax[0].hist(powermax_back2_freq, bins=arange(0, 50 + binwidth, binwidth))
ax[0].set_xlim(0, 30)
ax[0].set_ylim(0, 10)
ax[0].set(xlabel='Frequency (Hz), Background')

(counts2, bins2, bars2) = ax[1].hist(powermax_epil2_freq, bins=arange(0, 50 + binwidth, binwidth))
ax[1].set_xlim(0, 30)
ax[1].set_ylim(0, 10)
ax[1].set(xlabel='Frequency (Hz), Seizure');

show()
```

OUTPUT

```
Frequencies of max power in background (Hz):
 [ 2.5 11.   4.5 10.5 10.5  1.5 10.5 10.5 10.5 11. ]

Frequencies of max power in seizure (Hz):
 [ 6.   7.5 10.  10.  12.   8.   3.5  7.   7.   8. ]
(0.0, 30.0)
(0.0, 10.0)
(0.0, 30.0)
(0.0, 10.0)
```



Fourier spectrum with max frequency

PYTHON

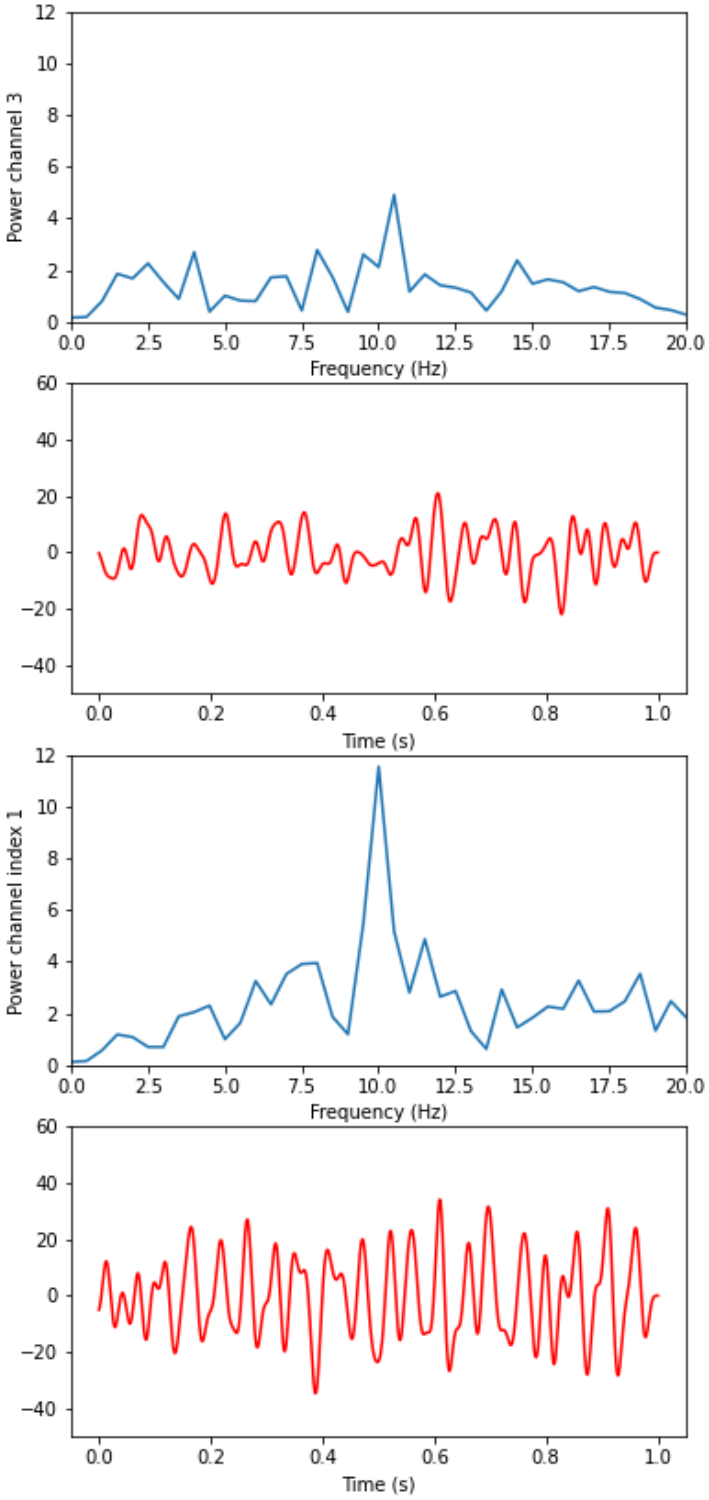
```
print('Maximum count: background: ', amax(counts1))

print('Maximum count: seizure:    ', amax(counts2))

print('Frequency with maximum count: background: ', '10-11 Hz')

print('Frequency with maximum count: seizure:    ', ' 7- 8 Hz')
```

OUTPUT

```
Maximum count: background:  5.0
Maximum count: seizure:     3.0
Frequency with maximum count: background:  10-11 Hz
Frequency with maximum count: seizure:      7- 8 Hz
```

PYTHON

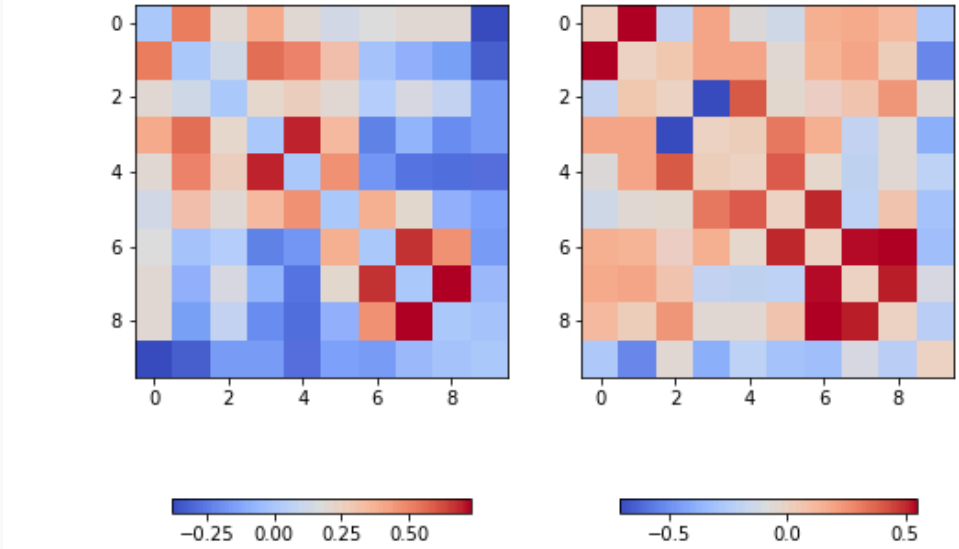
```
freqs = (sr/2)*linspace(0, 1, int(rows/2))

fig, axes = subplots(figsize=(6, 14), ncols=1, nrows=4, sharex=False)

axes[0].plot(freqs, powers_back2[:, 3])
axes[0].set_xlim(0, 20)
axes[0].set_ylim(0, 12)
axes[0].set(ylabel=f'Power channel 3')
axes[0].set(xlabel='Frequency (Hz)');

axes[1].plot(time, dat_back_filt[:, 3], c='r')
axes[1].set(xlabel='Time (s)');
axes[1].set_ylim(-50, 60)

axes[2].plot(freqs, powers_epil2[:, 3])
axes[2].set_xlim(0, 20)
axes[2].set_ylim(0, 12)
axes[2].set(ylabel='Power channel index 1')
axes[2].set(xlabel='Frequency (Hz)');

axes[3].plot(time, dat_epil_filt[:, 3], c='r')
axes[3].set(xlabel='Time (s)');
axes[3].set_ylim(-50, 60)

show()
```



Q5 ### Correlation Matrix

PYTHON

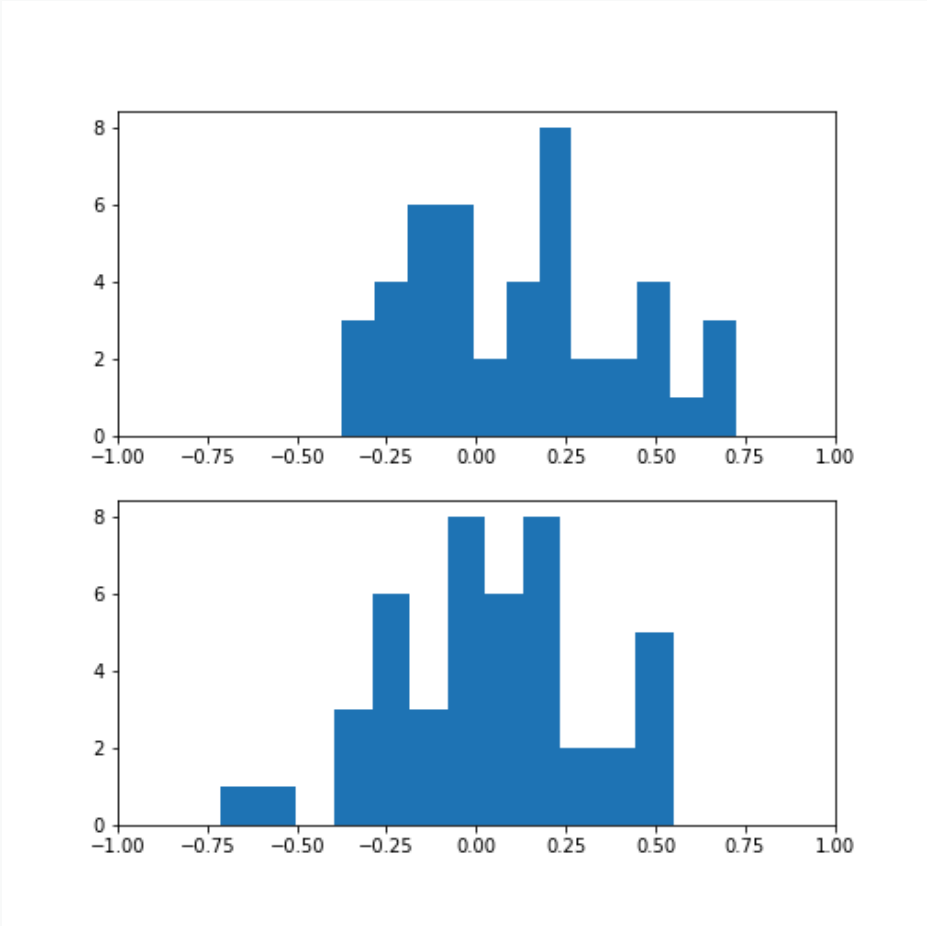
```
corr_matrix_back2 = corrcoef(dat_back_filt, rowvar=False)

fill_diagonal(corr_matrix_back2, 0)

corr_matrix_epil2 = corrcoef(dat_epil_filt, rowvar=False)

fill_diagonal(corr_matrix_epil2, 0)

fig, ax = subplots(figsize = (8,8), ncols=2)

im1 = ax[0].imshow(corr_matrix_back2, cmap='coolwarm');
fig.colorbar(im1, ax=ax[0], orientation='horizontal', shrink=0.8)

im2 = ax[1].imshow(corr_matrix_epil2, cmap='coolwarm');
fig.colorbar(im2, ax=ax[1], orientation='horizontal', shrink=0.8);

show()
```



Hist of Correlation Coefficients

PYTHON

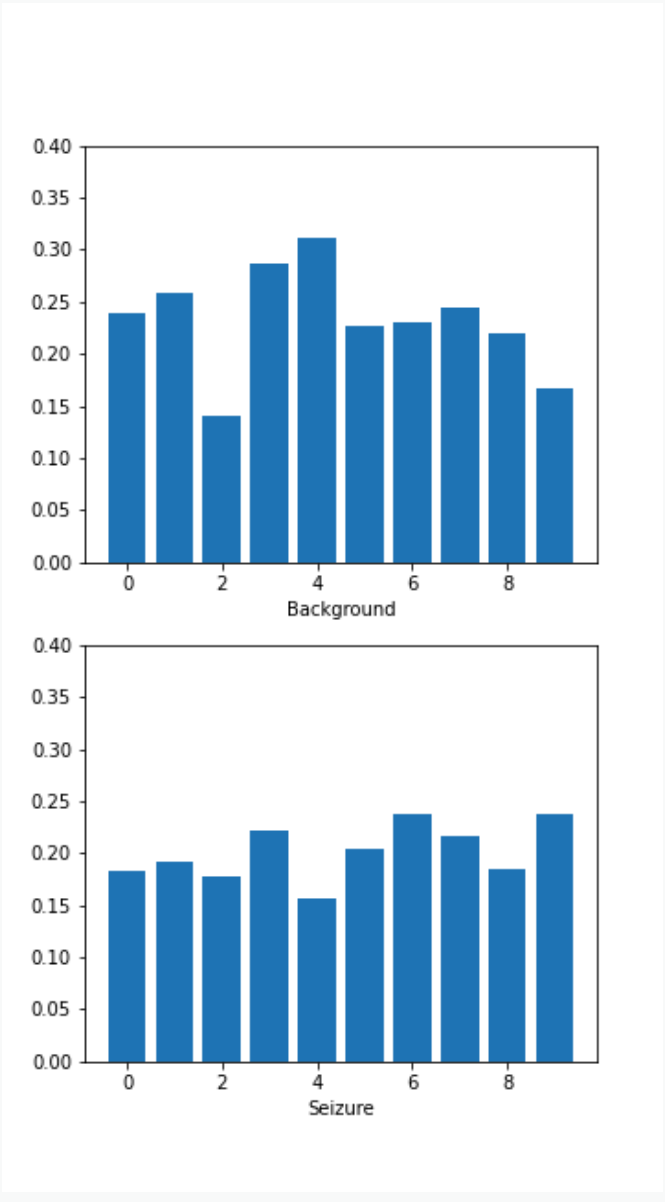
```
channels = dat_back.shape[1]

corr_coeffs_back2 = corr_matrix_back2[triu_indices(channels, k=1)]
corr_coeffs_epil2 = corr_matrix_epil2[triu_indices(channels, k=1)]

fig, ax = subplots(nrows=2)

ax[0].hist(corr_coeffs_back2, bins = 12);
ax[0].set_xlim(-1, 1)

ax[1].hist(corr_coeffs_epil2, bins = 12);
ax[1].set_xlim(-1, 1);

show()
```



Channel Correlations

PYTHON

```
corr_coeffs_back2_mean = mean(abs(corr_matrix_back2), axis=0)
corr_coeffs_epil2_mean = mean(abs(corr_matrix_epil2), axis=0)

fig, ax = subplots(nrows=2, figsize=(5,9))

bins = arange(corr_coeffs_back2_mean.shape[0])

ax[0].bar(bins, corr_coeffs_back2_mean);
ax[0].set_xlabel('Background')
ax[0].set_ylim(0, 0.4)

ax[1].bar(bins, corr_coeffs_epil2_mean);
ax[1].set_xlabel('Seizure')
ax[1].set_ylim(0, 0.4);

show()
```



PYTHON

```
corr_back2_mean = mean(corr_coeffs_back2_mean)
corr_epil2_mean = mean(corr_coeffs_epil2_mean)

print('Average correlation background: ', around(corr_back2_mean, decimals=2))
print('Average correlation seizure:    ', around(corr_epil2_mean, decimals=2))
```

OUTPUT

```
Average correlation background:  0.23
Average correlation seizure:     0.2
```

Channels with strongest correlations during the seizure

PYTHON

```
threshold = 0.2

corr_coeffs_epil2_large = corr_coeffs_epil2_mean > threshold

corr_coeffs_epil2_large
```

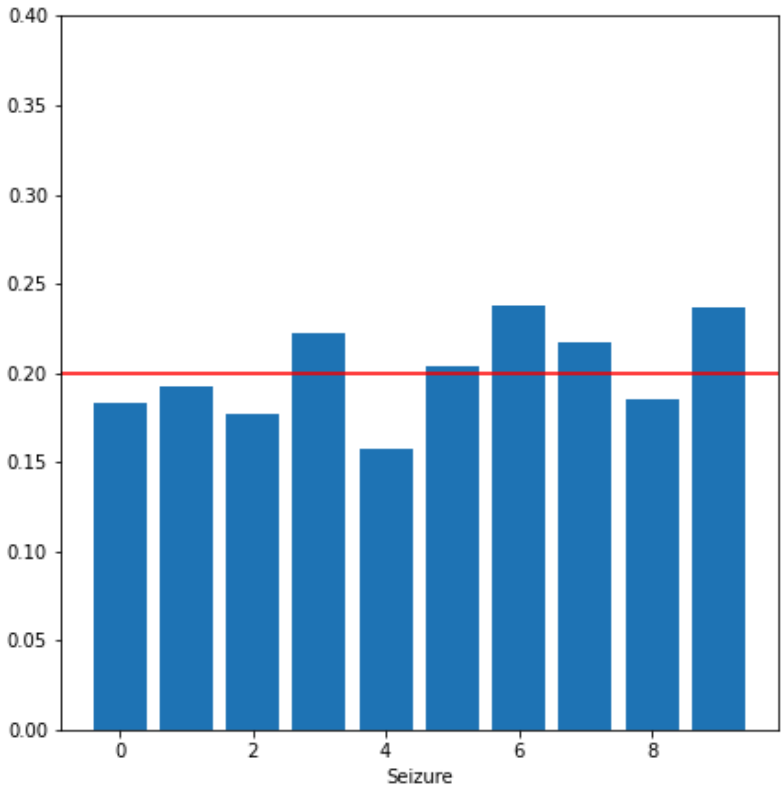
OUTPUT

```
array([False, False, False,  True, False,  True,  True,  True, False,
        True])
```

A horizontal line can be used to indicate the threshold:

PYTHON

```
fig, ax = subplots()

bins = arange(corr_coeffs_back2_mean.shape[0])

ax.bar(bins, corr_coeffs_epil2_mean);
ax.set_xlabel('Seizure')
ax.set_ylim(0, 0.4);
ax.axhline(y=threshold, c='r');

show()
```



To find the indices of the channels that are larger than the threshold (i.e. those displaying True), we can use Numpy function `nonzero`:

PYTHON

```
from numpy import nonzero

nonzero(corr_coeffs_epil2_large)
```

OUTPUT

```
(array([3, 5, 6, 7, 9]),)
```

KEY POINTS

- `plot_series` is a Python function created to display multiple timeseries plots.
- Data filtering is applied to take out specific and relevant components.
- The Fourier spectrum decomposes the time series into a sum of sine waves.
- Cross-correlation matrix is used for multivariate analysis.