Signal Analysis Lecture Notes

Sergio Peignier

Contents

Course Format and Student Responsibilities

The course is heavily practical, with only few hours of practical sessions and a 2-hour recap session at the end. Due to this limited contact time, **you are expected to acquire the majority of the theoretical foundations on their own**. **Proactive engagement is essential: you must review relevant concepts, read library documentation, and attempt exercises independently**. **Without active learning outside the lab sessions, you are unlikely to fully understand the theoretical and practical aspects of signal analysis**. Consequently, **you are expected to take responsibility for their learning, preparing in advance for practical sessions, and actively reflecting on the analysis results**. This course thus combines hands-on technical skills, signal interpretation, and self-directed learning, preparing students for rigorous, independent analysis of time-series data.

Course Overview and Objectives

This course is designed to provide you with hands-on experience in analyzing 1D temporal biological signals using Python. The focus is on practical data processing, signal analysis, and the application of core mathematical tools in a physiological context.

You will primarily work within a Jupyter Notebook environment, leveraging standard Python libraries such as NumPy, Pandas, and Scikit-learn. The course emphasizes good software engineering practices, including: i) Modular organization of code via functions, ii) Clear, well-commented notebook cells, Careful reading of official documentation to understand and correctly use library functions.

Students will learn to handle biological time-series data efficiently while following good programming principles. This includes organizing code clearly, documenting functions with docstrings, and selecting appropriate parameters for library functions based on the data characteristics.

Using biological 1D signals, you will apply tools such as: Autocorrelation to study temporal regularity, Fourier series and transforms to identify dominant frequency components, Band-pass filtering to isolate frequency bands of physiological interest, Peak detection to locate relevant events or features in the time series.

Importantly, you are expected to link the mathematical properties of the signals (e.g., periodicity, dominant frequencies, smoothness) with underlying physiological phenomena, such as heart rhythm, neural activity, or other cyclic biological processes.

The theoretical knowledge that should be acquired by the students **before** the practicals is located in Sections 1, 2, 3 and 4. The Section 5 contain exercises. The aim of these exercises is to help you integrate the different concepts covered in this lecture more effectively. If you can answer the questions correctly, you have understood the concepts; otherwise, review them again. These exercises will not be solved during the practical sessions. You should solve them independently in preparation for the practical sessions. Section 6 contains a summary with the most useful libraries for each topic covered here, along with references to the libraries' official web-pages and other useful tutorials. Section 7 contains guidelines summarizing good practices for writing, testing, and understanding code in Python or any scientific computing environment. Section 8 contains guidelines summarizing good methodological practices for data analysis. Finally Section 9 contains the questions that you are meant to solve during the practical sessions.

1 Common Preprocessing Steps in Signal Processing

Before analyzing or modeling any signal, it is crucial to preprocess the raw data to remove artifacts, normalize scales, and highlight the meaningful structure of the signal. Poor preprocessing can lead to misleading interpretations. This section presents the most common preprocessing steps, together with their intuition, theoretical foundations, and applications.

1.1 Outlier Detection and Filtering

Intuition

In real-world recordings, sensors often produce occasional spikes or erroneous measurements (e.g., due to movement, poor contact, or transmission noise). These outliers can distort summary statistics, correlation estimates, and spectral analyses. Detecting and removing them helps stabilize downstream computations.

Theoretical Explanation

Let $x_1, x_2, \ldots, x_N$ be a set of observations. A robust way to detect outliers is to use the **interquartile range (IQR)** method:

$$\text{IQR} = Q_3 - Q_1$$

where Q_1 and Q_3 are the first and third quartiles of the data. An observation x_i is considered an outlier if:

$$x_i < Q_1 - k \cdot \text{IQR} \quad \text{or} \quad x_i > Q_3 + k \cdot \text{IQR}$$

with k typically equal to 1.5 (moderate filtering) or 3 (aggressive filtering).

This method is robust because quartiles are not affected by extreme values, unlike the mean and standard deviation.

1.2 Centering and Scaling

Intuition

Different signals or features may have different average values or amplitudes. Many analysis techniques assume that all variables are on comparable scales. Centering and scaling ensure that each feature contributes equally to the analysis.

Theoretical Explanation

Given a signal $x(t)$, we define:

$$\tilde{x}(t) = \frac{x(t) - \mu_x}{\sigma_x}$$

where μ_x is the mean and σ_x is the standard deviation:

$$\mu_x = \frac{1}{N}\sum_{t=1}^{N} x(t), \qquad \sigma_x = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(x(t) - \mu_x)^2}$$

The centered and reduced signal $\tilde{x}(t)$ has zero mean and unit variance. This transformation preserves shape but standardizes amplitude.

Application

Centering and scaling are often applied before:

- **Fourier transforms:** to remove DC offsets that create peaks at zero frequency.
- **Correlation analyses:** to make signals comparable.
- **Further analysis:** to ensure features have similar numerical ranges.

1.3 Normalization

Intuition

Normalization rescales the amplitude of a signal to a fixed range, e.g. $[0, 1]$ or $[-1, 1]$, making it easier to compare signals with different units or magnitudes.

Theoretical Explanation

Given a signal $x(t)$ with minimum and maximum values $x_{\min}$ and $x_{\max}$, the normalized signal is:

$$x_{\text{norm}}(t) = \frac{x(t) - x_{\min}}{x_{\max} - x_{\min}}$$

Alternatively, in bipolar normalization:

$$x_{\text{norm}}(t) = 2 \cdot \frac{x(t) - x_{\min}}{x_{\max} - x_{\min}} - 1$$

Normalization changes amplitude but not shape or frequency content.

2 Autocorrelation

2.1 Intuition

Intuitively, autocorrelation measures the **degree to which a time-series is correlated with a shifted version of itself**. Let a set of observations $\{x_{t\in T}\}$ indexed by time $t = (1, 2, \dots)$, the autocorrelation measures to which extent the values x_t are correlated with values $x_{t-\tau}$ for all t and for a given lag τ. When the correlation is close to zeros for all lags τ, it indicates that the series behaves like independent noise, otherwise the series has some kind of temporal dependence or seasonality.

Let us consider a simple and intuitive example: let us consider daily temperatures such that x_i denotes the temperature i-th day. It is reasonable to imagine that if today is warm then tomorrow will also be warm, then we could expect a positive autocorrelation at lag $\tau = 1$. Moreover it is reasonable to imagine that the temperature for a one year lag will also be similar, then we should expect a positive correlation for $\tau = 365$.

Assessing the autocorrelation can reveal temporal dependencies, such as repeating patterns or cycles. Identifying such kinds of temporal patterns, can guide the selection of the methods for further analysis. Moreover some statistical methods rely on independence assumptions, and hence autocorrelation analysis may help detecting and possibly removing temporal dependencies. The assessment of regression or forecasting models often aims at ensuring that residuals are uncorrelated, and then autocorrelation analysis applied to the residuals can help validating the model.

2.2 Theoretical Foundations

Stationarity Autocorrelation is only defined for **weakly stationary processes**. Indeed, the autocorrelation, by definition is only a function of the lag, and thus assumes implicitly that the correlation between the signal and its shifted version **only** depends on the lag and not on the time. Then without stationarity, non-constant variance may produce misleading results for autocorrelation.

Then, $\{x_t\}$ is weakly stationary if:

1. $E[x_t] = \mu$ is constant.
2. $\mathrm{Var}(x_t) = \sigma^2$ is finite and constant.
3. $\mathrm{Cov}(x_t, x_{t-\tau})$ depends only on lag τ and not on t.

In real-world applications, signals are rarely perfectly stationary. In order to use autocorrelation tools, we can either assume local stationarity over short windows where the properties are approximately stationary, or normalize the signal's trends before computing the autocorrelation.

Autocovariance For a stationary process with mean μ, the **autocovariance function** is defined as:

$$\gamma(\tau) = \mathrm{Cov}(x_t, x_{t-\tau}) = E[(x_t - \mu)(x_{t-\tau} - \mu)]$$

Autocorrelation The **autocorrelation function (ACF)** is simply the normalized version:

$$\rho(\tau) = \frac{\gamma(\tau)}{\gamma(0)} = \frac{\mathrm{Cov}(x_t, x_{t-\tau})}{\mathrm{Var}(x_t)}$$

Thus, $\rho(\tau) \in [-1, 1]$ measures the correlation between x_t and $x_{t-\tau}$.

Continuous autocorrelation For a continuous signal $x(t)$, the autocorrelation function at lag τ is:

$$\rho(\tau) = \int_{-\infty}^{\infty} x(t)\, x(t-\tau)\, dt.$$

Discrete autocorrelation For a discrete-time signal $x[n]$, it is defined as

$$\rho(k) = \sum_{n=-\infty}^{\infty} x[n]\, x[n-k].$$

Sample Autocorrelation For a finite sample $\{x_1, \ldots, x_N\}$. The empirical autocovariance at lag τ is:

$$\hat{\gamma}(\tau) = \frac{1}{N-\tau} \sum_{t=\tau+1}^{N} (x_t - \bar{x})(x_{t-\tau} - \bar{x})$$

Where $\bar{x} = \frac{\sum_{t=1}^{N} x_t}{N}$ is the empirical mean value for the variables $\{x_t\}$.
Then, the sample autocorrelation function is simply defined as:

$$\hat{\rho}(h) = \frac{\hat{\gamma}(\tau)}{\hat{\gamma}(0)}$$

Correlogram is a plot representing $\hat{\rho}(h)$ as a function of the lag τ. For large N, if the true process is white noise (i.e., noise without autocorrelation), approximate 95% confidence bounds are given by $\pm 2/\sqrt{N}$. When autocorrelations exceed these bounds, the null "no autocorrelation" hypothesis can be rejected.

2.3 Recommended Lecture Notes, Slides, and Videos

Lecture Notes

- Penn State STAT510: Lesson 1 – Time Series Basics
- University of Southampton: Time Series Analysis Lecture Slides

Videos

- What is Autocorrelation? — Time Series Analysis in Python (Egor Howell)
- What is Autocorrelation? (Iain Explains)

3 Fourier Transform

3.1 Intuition

Let us consider a signal in the time domain, for example, a sound wave or an electrical signal over time. The Fourier Transform (FT) is a function that tells you **what frequencies constitute the signal**. To do so, the FT expresses a complex signal as a sum of simpler sinusoidal components (sines/cosines) of different frequencies and amplitudes. According to this decomposition, the FT gives a continuous spectrum showing how much of each frequency is present. In other words, the FT allows to **map the signal from the time domain (values as a function of time) to the frequency domain (values as a function of frequencies)**. This mapping is particularly useful, since it allows to better analyze, filter and understand a signal in its frequency domain rather than the time domain. For example, the FT allows to analyze periodicities, harmonics, and spectral content of signals, revealing its underlying structure, the FT also allows to simplify operations like filtering and convolution.

The **inverse Fourier transform** allows us to perform the inverse mapping **from the frequency domain back to the time domain**, ensuring no information is lost (apart from sampling or rounding errors). Therefore it could be possible to map a signal to the frequency domain, then filter out some undesired frequencies, and map the signal back to the temporal domain to obtain the cleaned temporal signal.

3.2 Euler's Formula

A central mathematical foundation of the Fourier Transform is **Euler's formula**, which expresses the complex exponential function in terms of trigonometric functions:

$$e^{i\theta} = \cos(\theta) + i\sin(\theta)$$

Similarly,

$$e^{-i\theta} = \cos(\theta) - i\sin(\theta)$$

By adding and subtracting these two identities, we can express sine and cosine as combinations of complex exponentials:

$$\cos(\theta) = \frac{e^{i\theta} + e^{-i\theta}}{2}, \qquad \sin(\theta) = \frac{e^{i\theta} - e^{-i\theta}}{2i}$$

This relationship is fundamental in Fourier analysis because it allows any real-valued oscillation (a sine or cosine wave) to be represented compactly as an exponential term $e^{i\omega t}$.

3.3 Continuous Fourier Transform

For a continuous signal $f(t)$, the Fourier Transform is defined as:

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\, e^{-i\omega t}\, dt$$

and the inverse transform is:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega)\, e^{i\omega t}\, d\omega$$

Where $F(\omega)$ describes the amplitude and phase of each sinusoidal component at frequency ω. In the Fourier Transform, the kernel $e^{-i\omega t}$ can be understood as a combination of a cosine term (representing the real part) and a sine term (representing the imaginary part). Hence, the Fourier Transform measures how much of each *sinusoidal component* (sine and cosine at frequency ω) is present in the signal $f(t)$.

3.4 Fourier Transform Properties

Linearity property FT is a **linear operator**: The FT of a sum equals the sum of the FTs.

$$\mathcal{F}\{af(t) + bg(t)\} = aF(\omega) + bG(\omega)$$

Proof:

$$\begin{aligned}\mathcal{F}\{af(t) + bg(t)\} &= \int_{-\infty}^{\infty} [af(t) + bg(t)]\, e^{-i\omega t}\, dt \\ &= a\int_{-\infty}^{\infty} f(t)\, e^{-i\omega t}\, dt + b\int_{-\infty}^{\infty} g(t)\, e^{-i\omega t}\, dt \\ &= aF(\omega) + bG(\omega)\end{aligned}$$

Time shift property A shift in time corresponds to a phase shift in frequency. Thus, a delay in time by t_0 multiplies the spectrum by a complex exponential of phase $-\omega t_0$.

$$\mathcal{F}\{f(t - t_0)\} = e^{-i\omega t_0} F(\omega)$$

Proof:

$$\mathcal{F}\{f(t - t_0)\} = \int_{-\infty}^{\infty} f(t - t_0)\, e^{-i\omega t}\, dt$$

Let $u = t - t_0$, so $t = u + t_0$ and $dt = du$. Then:

$$\begin{aligned}\mathcal{F}\{f(t - t_0)\} &= \int_{-\infty}^{\infty} f(u)\, e^{-i\omega(u+t_0)}\, du \\ &= e^{-i\omega t_0}\int_{-\infty}^{\infty} f(u)\, e^{-i\omega u}\, du \\ &= e^{-i\omega t_0} F(\omega)\end{aligned}$$

Frequency shift property Multiplying by a complex exponential in time shifts the spectrum in frequency. Hence, a modulation in time corresponds to a shift in the frequency domain.

$$\mathcal{F}\{f(t)e^{i\omega_0 t}\} = F(\omega - \omega_0)$$

Proof:

$$\begin{aligned}\mathcal{F}\{f(t)e^{i\omega_0 t}\} &= \int_{-\infty}^{\infty} f(t)\, e^{i\omega_0 t}\, e^{-i\omega t}\, dt \\ &= \int_{-\infty}^{\infty} f(t)\, e^{-i(\omega-\omega_0)t}\, dt \\ &= F(\omega - \omega_0)\end{aligned}$$

Convolution theorem **Property:** Convolution in time corresponds to multiplication in frequency, and vice versa.

$$\mathcal{F}\{(f * g)(t)\} = F(\omega)\, G(\omega)$$

where

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t - \tau)\, d\tau$$

The Inverse Form (Dual Relationship) states that:

$$\mathcal{F}^{-1}\{F(\omega) * G(\omega)\} = \frac{1}{2\pi}\, f(t)\, g(t)$$

Proof:

$$\begin{aligned}\mathcal{F}\{(f*g)(t)\} &= \int_{-\infty}^{\infty} (f*g)(t)\, e^{-i\omega t}\, dt \\ &= \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f(\tau)\, g(t-\tau)\, d\tau\right] e^{-i\omega t}\, dt\end{aligned}$$

Interchanging the order of integration (by Fubini's theorem):

$$\mathcal{F}\{(f*g)(t)\} = \int_{-\infty}^{\infty} f(\tau) \left[\int_{-\infty}^{\infty} g(t-\tau)\, e^{-i\omega t}\, dt\right] d\tau$$

Let $u = t - \tau$ so $dt = du$:

$$\begin{aligned}\mathcal{F}\{(f*g)(t)\} &= \int_{-\infty}^{\infty} f(\tau) \left[\int_{-\infty}^{\infty} g(u)\, e^{-i\omega(u+\tau)}\, du\right] d\tau \\ &= \int_{-\infty}^{\infty} f(\tau)\, e^{-i\omega\tau}\, d\tau \int_{-\infty}^{\infty} g(u)\, e^{-i\omega u}\, du \\ &= F(\omega)\, G(\omega)\end{aligned}$$

Thus, convolution in the time domain is equivalent to pointwise multiplication in the frequency domain. This theorem explains why filtering is efficient in the frequency domain.

Parseval's theorem The total energy (or power) of a signal in time equals that in the frequency domain. Fourier Transform preserves energy — it is a unitary transform up to scaling.

$$\int_{-\infty}^{\infty} |f(t)|^2\, dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} |F(\omega)|^2\, d\omega$$

Proof: Starting from the inverse FT:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega)\, e^{i\omega t}\, d\omega$$

Then:

$$\begin{aligned}\int_{-\infty}^{\infty} |f(t)|^2\, dt &= \int_{-\infty}^{\infty} f(t)\, f^*(t)\, dt \\ &= \int_{-\infty}^{\infty} f(t) \left[\frac{1}{2\pi} \int_{-\infty}^{\infty} F^*(\omega')\, e^{-i\omega' t}\, d\omega'\right] dt \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} F^*(\omega') \left[\int_{-\infty}^{\infty} f(t)\, e^{-i\omega' t}\, dt\right] d\omega'\end{aligned}$$

The inner integral is just $F(\omega')$. Therefore:

$$\int_{-\infty}^{\infty} |f(t)|^2\, dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega') F^*(\omega')\, d\omega' = \frac{1}{2\pi} \int_{-\infty}^{\infty} |F(\omega)|^2\, d\omega$$

3.5 Discrete Fourier Transform (DFT)

For a sequence $x_0, x_1, \ldots, x_{N-1}$ the DFT is:

$$X_k = \sum_{n=0}^{N-1} x_n\, e^{-i2\pi kn/N}, \quad k = 0, 1, \ldots, N-1$$

The inverse DFT is:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k\, e^{i2\pi kn/N}, \quad n = 0, 1, \ldots, N-1$$

Where ω is the normalized angular frequency (radians per sample). Unlike the continuous FT, the DTFT is **periodic** in ω with period 2π:

$$X(\omega + 2\pi) = X(\omega)$$

In discrete systems, signals are sampled at rate f_s. Frequencies above the Nyquist limit ($f_s/2$) appear as lower-frequency components due to *aliasing*. Hence, proper sampling and filtering are essential for accurate Fourier analysis.

Fundamental Properties of the DFT The fundamental properties described for the Continuous FT also apply to the DFT.

3.6 Fast Fourier Transform (FFT)

The FFT is *not a different transform*. It is an efficient algorithm for computing the discrete version of the Fourier Transform, the *Discrete Fourier Transform (DFT)*. When you have a digital signal with N data points, the naive DFT incurs in $\sim N^2$ operations (the formal notation of such a time-complexity is $\mathcal{O}(N^2)$) The FFT reduces this to $\sim N \log N$ operations (the formal notation of such a time-complexity is $\mathcal{O}(N \log N)$) by exploiting symmetries in the complex exponentials, using a so-called divide-and-conquer strategy. Intuitively, the FFT algorithm rewrites the DFT computation so that redundant work is avoided by recursively breaking the problem into smaller pieces. Thus, the FFT's main objective is computational efficiency. This lecture does not aim at explaining the FFT algorithm in details, but relevant information can be found in the following videos and lecture notes.

3.7 Filtering

Intuition

Fourier transform is often used for filtering frequency bands of interest by attenuating unwanted components (noise, drift, high-frequency artifacts).

Common filters include:

- **Low-pass:** keeps slow components, removes high-frequency noise.
- **High-pass:** removes drifts and trends.
- **Band-pass:** isolates physiological ranges (e.g., 0.5–8 Hz for PPG).

3.8 Short-Time Fourier Transform (Short-Time Fourier Transform)

Many real-world signals (speech, music, biomedical signals) are inherently non-stationary. Their spectral properties evolve over time. A standard FT averages the signal's frequency content over its entire duration and thus cannot describe temporal variations. The Short-Time Fourier Transform (STFT) overcomes this limitation by computing local Fourier Transforms on overlapping windowed segments of the signal, producing a two-dimensional time–frequency representation often visualised as a *spectrogram*.

The key idea of the STFT is to: 1) select a short window function $w(t)$ that is non-zero only around $t = 0$, typically of width T (called the *window length*)); 2) slide this window across time; 3) at each time position, compute the Fourier Transform of the windowed segment; 4) stack these local spectra to form a time–frequency map. Choosing the window length induces a fundamental trade-off: a short window gives high time resolution but coarse frequency resolution, whereas a long window gives high frequency resolution but poor time localisation. This is a manifestation of the "time-frequency uncertainty principle".

Let $x(t)$ be a continuous-time signal and $w(t)$ a window function. The Short-Time Fourier Transform is defined as:

$$X(\tau, \omega) = \int_{-\infty}^{\infty} x(t)\, w(t - \tau)\, e^{-j\omega t}\, dt,$$

where τ is the time shift (window centre) and ω the angular frequency.

In discrete time, using a window of length N, the STFT is:

$$X[n,k] = \sum_{m=0}^{N-1} x[m]\, w[m-n]\, e^{-j\,2\pi km/N},$$

where n is the discrete time-frame index and k the frequency-bin index.

3.9 Recommended Lecture Notes, Slides, and Videos

Videos

- But what is the Fourier Transform? (3Blue1Brown)
- The Fast Fourier Transform (FFT): Most Ingenious Algorithm Ever?
- Discrete Fourier Transform - Simple Step by Step
- The FFT Algorithm - Simple Step by Step

Lecture Notes and Slides

- Imperial College London Lecture 4 - Frequency Domain Analysis and Fourier Transform
- Harvard Lecture 8: Fourier Transforms.
- Oxford Lecture 7 – The Discrete Fourier Transform.
- University of Illinois - Lecture 5: Short-Time Fourier Transform and Filterbanks

4 Peak Detection in Time Series Analysis

4.1 Intuition

Peak detection is a fundamental operation in time series and signal processing. Intuitively, peak detection aims at *finding "bumps" or "spikes"* in a data sequence. Imagine monitoring vibrations of a rotating machine: each revolution produces a small pulse. Detecting peaks lets you estimate the rotation period and spot abnormalities if one cycle deviates.

4.2 Objectives and Applications

- **Feature extraction:** Peaks often correspond to events of interest (e.g., heartbeats, mechanical shocks). For example this kind of tool could help detecting R-peaks in ECGs.
- **Pattern recognition:** Repeated peaks indicate periodic or cyclic behavior, as well as events boundaries.
- **Anomaly detection:** An unexpected large peak may reveal faults, shocks, or bursts. For example this kind of tool could help to determine neuron firing burst in EEGs.
- **Preprocessing:** Peaks are often used to align, segment, or synchronize signals before further analysis.

4.3 Theoretical Foundations

Simple Definition Peak detection links intuitive notions of "spikes" to mathematical definitions of local maxima. More formally, peak detection aims to identify **local maxima** (or minima) that represent meaningful events, such as pulses, spikes, or transient phenomena.

For a continuous-time signal $x(t)$, a *peak* occurs at time t_0 if

$$\frac{dx}{dt}(t_0) = 0 \quad \text{and} \quad \frac{d^2x}{dt^2}(t_0) < 0.$$

For a discrete-time signal $x[n]$, a point n_0 is a peak if

$$x[n_0] > x[n_0 - 1] \quad \text{and} \quad x[n_0] > x[n_0 + 1].$$

In practice, noise and fluctuations make this definition impractical, and real algorithms use additional criteria. In this case we usually smooth the data (e.g., computing some kind of moving average or applying a Gaussian filter). We also compare each point with its k neighbors to find local maxima. We often apply thresholds on amplitude, prominence, and spacing to ignore irrelevant fluctuations.

Thresholds and prominence To reduce false detections due to noise, we often define:

i) $H_{\min}$ the minimum peak height (the peak amplitude must exceed this threshold)

ii) $P_{\min}$ the minimum prominence (the peak must stand out relative to its surrounding local optima by at least $P_{\min}$)

iii) $D_{\min}$ the minimum distance (two detected peaks must be separated by at least $D_{\min}$ samples (or time units) to avoid detecting multiple peaks within one event.)

Then a local optimum n_0 is a valid peak if:

$$\begin{cases} x[n_0] > H_{\min}, \\ x[n_0] - \min(x[n_1], x[n_2]) \geq P_{\min}, \\ |n_0 - n_p| \geq D_{\min} \quad \forall n_p \in \mathcal{P}, \end{cases}$$

Where $x[n_1]$ and $x[n_2]$ are the local minima on each side of the peak.

Matched filtering interpretation When the peak shape is known, detection can be cast as the correlation between both signals. Let $h[t]$ be a template matching the expected peak and let $x[t]$ be our signal.

$$y[n] = \sum_k x[n-k]\, h[k] = (x * h)[n],$$

Peaks in $y[n]$ correspond to best matches between x and h.

Smoothing with Moving Averages In practical time-series analysis, raw data often contain high-frequency noise that produces many small, spurious local maxima. A common way to reduce this effect before peak detection is to apply a **moving average filter** (also called **running mean**).

For a signal $x[n]$, the moving average of window length M is defined as

$$y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k].$$

This operation replaces each sample with the average of its M most recent values, effectively smoothing short-term fluctuations.

The moving average acts as a **low-pass filter**: it attenuates rapid oscillations (high-frequency noise) while preserving slower trends and broader peaks. As a result, only the most prominent, wide peaks remain after smoothing—making them easier to detect reliably.

Therefore, such a moving average smoothing is equivalent to multiplying the signal's spectrum by a low-pass filter in the frequency domain. By filtering out noise before applying the peak detection algorithm, we improve both robustness and accuracy.

Because convolution in the time domain corresponds to multiplication in the frequency domain, moving averages (especially with long windows) can also be efficiently computed via the Fourier Transform: This is useful for large datasets, where performing the convolution directly is computationally expensive.

4.4 Lecture Notes and Video References

- Laurent Oudré, *Detecting Patterns in Time Series*, available at: `https://www.laurentoudre.fr/ast/AST1.pdf`
- TU Chemnitz, *Time Series Analysis – Selected Topics*, available at: `https://www.tu-chemnitz.de/mathematik/fima/public/LecturesAndMaterials/Zeitreihen.pdf`

Recommended Videos:

- AnalytiCode *2 Methods to Find Peaks in Mass Spectrometry Data Using Python* (`https://www.youtube.com/watch?v=nSTmyugeuA8`)
- Finxter AI Nuggets *Python Scipy signal.find_peaks() – A Helpful Guide* (`https://www.youtube.com/watch?v=75mdKyA76i8`)

5 Exercises

The aim of these exercises is to help you integrate the different concepts covered in this lecture more effectively. If you can answer the questions correctly, you have understood the concepts; otherwise, review them again. These exercises will not be covered during the practical sessions. You should solve them independently in preparation for the practical sessions.

5.1 Autocorrelation

1. Explain what the autocorrelation function represents and describe its expected shape for a sine wave and a rectangular pulse.
2. For $x(t) = A\cos(\omega_0 t)$, compute its autocorrelation function $\rho(\tau)$.
3. Show that $\rho(0) = \int |x(t)|^2 dt$, this corresponds to the total signal energy.
4. The white noise is defined as a discrete time random signal x_t such that $E[x] = 0$, $Var[x] = \sigma^2$ and $Cov(x_t, x_h) = 0 \; \forall t \neq h$. Compute the autocorrelation of the white noise
5. Explain how to use the autocorrelation to estimate the period of a noisy periodic signal. Why does it work even in noisy conditions?
6. For $x[n] = [2, 3, 1, 0]$, compute the autocorrelation $\rho[k]$.

5.2 Fourier Transform

1. Explain in simple words what is the Fourier Transform.
2. Develop sketches of proofs for the DFT properties based on the proofs for the continuous case.
3. Imagine you recorded a three-hour audio-book at home. Once you've finished, you realize that you left the oven on during this time, which has added a strange, low-pitched sound to the recording. You don't want to record it again. What can you do? Now let's imagine that the recording was made in the garden and very high-pitched birdsong was also recorded. What would you do?
4. Let $f(t) = e^{-at}u(t)$, where $u(t)$ is the unit step function and $a > 0$. Compute its Fourier transform $F(\omega)$ and discuss what the result tells you about the spectrum of a decaying exponential.
5. Explain why the FT linearity property could be useful in signal decomposition and filtering, propose a couple of realistic examples.
6. What happens with the total energy of a signal when you apply, for instance, a low-pass filtering?
7. Interestingly, it is also possible to compute the DFT of an image. In this case, instead of having a single sum over the 1D signal, we have a double sum over the x and y coordinates of the image. Similarly the frequency domain is now a 2D plane. Now, you are asked to blur an image using a 2D Gaussian filter. Explain conceptually how using the FFT can accelerate this operation compared to direct spatial convolution. What property of the Fourier transform enables this?
8. Imagine you have an MP3 file containing a three-hour audio-book recording. The sample rate is 22050 samples per second. How many samples does your signal have in total? You want to compute the DFT of the recording in order to filter it. Assuming that one elementary computing operation takes 10^{-7} seconds, how long would it take to compute the DFT using the naive approach? And using the FFT? Compute the time gain ratio.
9. Explain why the DTFT of a discrete-time sequence $x[n]$ is periodic with period 2π. Discuss how this periodicity influences how we interpret FFT results for sampled signals.

10. Let us imagine that you plan to record a friend of you playing the flute. In order to minimize the size of the file in your computer you plan to sample the signal at a rate equal to 700Hz. Knowing that the musical tuning standard is equal to 440 Hz, and assuming that the song may contain frequencies that are around this tuning standard, do you think it is a good idea? What should you do instead?

11. Explain why the classical Fourier Transform could be inadequate for analysing signals whose frequency content changes over time.

12. Construct a synthetic signal composed of a 100 Hz sine wave from $t = 0$ to 0.5 s and a 300 Hz sine wave from 0.5 s to 1 s. Sample at 2000 Hz. Compute and plot its spectrogram using a 100 ms window with 50% overlap. Describe the spectrogram.

5.3 Peak detection

1. Define a local maximum in a discrete-time signal. Explain how it differs from a global maximum.

2. Why does a high noise level create many false peaks?

3. Given $x[n] = [1, 2, 5, 3, 2, 1, 4, 2, 0]$, identify peaks if only a minimum height threshold $H_{\min} = 4$ is applied.

4. Explain what is meant by *peak prominence*. How is it computed? Why could it be an interesting criteria?

5. Explain the possible advantages and disadvantages of using a matched filtering to detect peaks? In which cases would you use such a tool?

6. Why is measuring inter-peak intervals useful? Give one example.

7. Now you want to detect minima instead of maxima in python, how can you proceed in practice?

8. Suggest an algorithm for online (real-time) peak detection on a micro-computer device. To do so, try to imagine the possible limitations and adapt your idea to cope with such constrains.

5.4 Ethical questions

The critical thinking framework could help you to better analyze the following questions (`https://www.criticalthinking.org/files/Concepts_Tools.pdf`).

1. Who should own physiological data: the person it comes from, the research team or the institution that collected it? Analyze the different positions.

2. You should analyse some physiological data from patients and consider a company's proposal to test their new pipeline. What criteria should you consider?

3. You and your colleague are analysing some time series data. Your colleague starts removing many 'aberrant' signals in order to 'obtain clearer results'. What would you do in this situation?

4. Suppose a researcher uses a filtering method without understanding its frequency response and publishes misleading conclusions. Is ignorance an ethical excuse? Why or why not?

5. If you were to design a wearable device that continuously monitors biological signals, which ethical considerations would you prioritise and why?

6. The government is proposing the implementation of a generalised health monitoring system to enable earlier intervention in the event of disease. What do you think about this?

6 Python Libraries for Signal and Time-Series Analysis

Python offers a rich ecosystem of open-source libraries that make it possible to implement, visualize, and experiment with the concepts of autocorrelation, Fourier transforms, and peak detection introduced in these notes. Below we summarize the most useful libraries for each topic, along with references to the libraries' official web-pages and other useful tutorials. Please take a quick look at this before the practical sessions.

6.1 Autocorrelation Analysis

- **NumPy** (`https://numpy.org/doc/stable/`)
  Core scientific computing library in Python. Provides: Efficient arrays and mathematical operations. Check the`numpy.correlate()` for computing discrete autocorrelation.
- **Pandas** (`https://pandas.pydata.org/docs/`)
  Provides tools for time-series data, check `Series.autocorr()` to compute lag-k autocorrelation.
- **Statsmodels** (`https://www.statsmodels.org/stable/`)
  Focused on statistical time-series analysis: Check the `acf()` and `pacf()` functions for autocorrelation and partial autocorrelation. As well as visualization tools for correlograms.

6.2 Fourier Transform and Frequency Analysis

- **NumPy FFT module** (`numpy.fft`). Check `fft()`, `ifft()`, `fftfreq()` for DFT and inverse transforms.
- **SciPy FFT module** (`https://docs.scipy.org/doc/scipy/reference/fft.html`) has Optimized FFT algorithms: `scipy.fft.fft()` and `scipy.fft.ifft()`. It has also an optimized implementation for the STFT: `scipy.signal.ShortTimeFFT()` and `scipy.signal.stft()`.
- **Librosa** (`https://librosa.org/doc/latest/`)
  Specialized for audio-spectral analysis: `librosa.stft()` and `librosa.istft()` for short-time Fourier transforms.

6.3 Peak Detection and Event Localization

- **SciPy Signal Processing module** (`https://docs.scipy.org/doc/scipy/reference/signal.html`)
  Contains the widely used function `scipy.signal.find_peaks()`.
- **NumPy and Pandas** can be combined for custom peak detection via local comparisons, rolling averages, or smoothing filters.
- **Signal smoothing:** can be achieved using `scipy.signal.savgol_filter` (Savitzky–Golay smoothing), or using Pandas' Rolling means : `data.rolling(window=M).mean()`

6.4 Visualization

For plotting the signals, the spectra or the correlograms, try:

- **Matplotlib** (`https://matplotlib.org/stable/gallery/index.html`)
- **Seaborn** (`https://seaborn.pydata.org/index.html`)

6.5 Some useful tutorials

- MachineLearningPlus: Time Series Analysis in Python
- Real Python: Fast Fourier Transform (FFT) in Python
- SciPy documentation: `find_peaks()`

7 Good Programming Practices

Developing reliable and readable code is an essential skill. The following guidelines summarize good practices for writing, testing, and understanding code in Python or any scientific computing environment. Programming in scientific computing is not only about obtaining correct results, but also about writing code that others (and your future self) can understand, verify, and reuse. Good practices are part of scientific rigor.

Understand the tools you use.

- Before using any function, carefully read its official documentation. Understand the purpose of the function, the meaning of each input and output, and the available parameters.
- Always verify that the chosen parameters are appropriate for your application.
- A good test for understanding a function: imagine explaining it to a friend. If you cannot clearly describe its purpose and behavior, or if many questions arise, revisit the documentation and examples.

Develop and test incrementally.

- Avoid running large, complex scripts all at once. Instead, build and execute small blocks of code step by step, ensuring that each part behaves as expected.
- Once each component works correctly and you understand its behavior, integrate it into a larger program or notebook.
- Use interactive environments (e.g., Jupyter Notebook) to visualize intermediate outputs during development.

Debug systematically.

- When an error occurs, start reading the message from the bottom: that part often contains the most informative description of the problem.
- Search the exact error message online to understand its meaning—Python's community and Stack Overflow often provide detailed explanations.
- If a function or script produces unexpected results, insert diagnostic statements (e.g., `print()`) at different points to trace the execution and locate where the problem occurs.

Write clear and documented code.

- Always include a **docstring** at the beginning of each function, describing:
  - The purpose of the function.
  - Its input parameters and their types.
  - The outputs and their expected formats.
- Add comments to clarify nontrivial parts of the code or mathematical reasoning behind a computation.
- Use **explicit variable and function names** that reflect their role (e.g., `sampling_rate`, `compute_autocorrelation()`, `fft_spectrum`).
- Follow a consistent style.

Validate and verify results.

- Use simple test cases for which you already know the expected output (for example, the autocorrelation of a sine wave).
- If possible, compare your results with analytical formulas or results from reliable libraries.
- Plot intermediate results to visually confirm their plausibility—visual inspection is often the fastest way to detect mistakes in signal processing.

Ensure reproducibility.

- Keep all code, data, and configuration parameters together in an organized folder structure.
- Use random seeds (`numpy.random.seed()`) when randomness is involved, so experiments can be reproduced.
- Document the environment: Python version, library versions, and hardware information.

Version control and collaboration.

- Use a version control system (e.g., Git) to track code changes and experiment history. Regularly commit working versions with meaningful messages (may not apply for small practicals such as this one, but in general it is a good practice).
- Use notebooks or Markdown files to record explanations, assumptions, and interpretations of results.

Efficiency and scalability.

- Prefer vectorized operations (`NumPy` arrays) instead of loops when possible—they are faster and more concise.
- Profile the code if performance becomes critical, and optimize only the bottlenecks (try to understand why it is a bottleneck and try to find solutions).

Visualization and interpretation.

- Always label axes, include units, and use clear titles and legends.
- Visual confirmation can reveal trends or anomalies that numbers alone might hide.

Continuous learning.

- Explore example notebooks from official library documentation.
- Compare different implementations for the same concept (e.g., FFT using `numpy.fft` vs. `scipy.fft`).
- Discuss and share code with peers: teaching or reviewing others' code is one of the fastest ways to deepen understanding.

8 Methodological Guideline for Data Analysis

Understand the study and dataset

- Read the dataset documentation: what was measured, how, and why.
- Identify variables, sampling rates, conditions...
- Identify possible limitations (missing data, noise, bias, no control data).
- Summarize the dataset structure and objectives in your own words.

Inspect the data

- Check data types, consistency between the different measures.
- Load and plot examples of raw signals to spot artifacts or missing values.
- Compute simple statistics (mean, variance, median, max, min).

Define objectives and workflow

- Clearly state what you want to find
- Draft a step-by-step workflow and a few backup strategies.

Choose and understand your tools

- List all needed libraries.
- Read documentation for each function before using it.
- Test them on synthetic or simple data to verify behavior, before applying them on the real data.

Process the data incrementally

- Apply transformations step by step, visualizing at every stage.
- Check that each result makes physiological and mathematical sense.

Analyze and interpret

- Compare with theory or known values.
- Always interpret results critically: do they make sense, or could they be artifacts?

Ensure robustness and reproducibility

- Test sensitivity to parameters
- Use version control and save environment details (which versions for the libraries are you using?)
- Keep your code modular, documented, and reproducible.

Ethical considerations

- Ensure compliance with data-use agreements and anonymity rules.
- Avoid over-interpreting results; acknowledge uncertainties.
- Store and share only what you can share.

9 Practical Sessions Question

In this practical session we will focus on the following PPG data from the physionet database: `https://physionet.org/content/multimodal-nback-music/1.0.0/`, in order to answer the following question:

Can you tell from the PPG signal whether the background music influenced patient 4's heart rate?

You can take a look at the following chapter [1] to learn about other important signal processing techniques applied to PPG signals.

References

[1] E. Mejia-Mejia, J. Allen, K. Budidha, C. El-Hajj, P. A. Kyriacou, and P. H. Charlton. Photoplethysmography signal processing and synthesis. In *Photoplethysmography*, pages 69–146. Elsevier, 2022.