



UNIVERSITÀ DEGLI STUDI DI GENOVA

Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei
Sistemi

Corso di Laurea Triennale in Informatica
Anno Accademico 2021/2022

Analisi di serie temporali Aspetti Applicativi

Candidato
Alex Valle

Relatore
Prof. Francesca Odone

Indice

1 Introduzione	3
1.1 Premessa	3
1.2 Obiettivi	3
1.3 Tecnologie utilizzate	3
1.4 Suddivisione del lavoro	3
2 Studio dei metodi e delle applicazioni	5
2.1 Cos'è una serie temporale	5
2.2 Metodi per la gestione dei dati	6
2.2.1 Pacchetti utilizzati	6
2.2.2 Caricamento di un dataset	7
2.2.3 Rinomina delle colonne relative alle serie	7
2.2.4 Scelta dell'indice	8
2.2.5 Individuazione dei valori nulli e possibili soluzioni	9
2.2.6 Filtraggio	12
2.3 Componenti di una serie temporale	15
2.3.1 Trend	15
2.3.2 Stagionalità	16
2.3.3 Residui	17
2.3.4 Decomposizione di una serie	18
2.4 Stazionarietà	21
2.4.1 Dickey Fuller Test	22
2.5 Smoothing	25
2.5.1 Moving average	25
2.5.2 Exponential	26
2.5.3 Double Exponential	27
2.6 Autocorrelazione ed Autocorrelazione parziale	30
2.6.1 Funzione di Autocorrelazione	30
2.6.2 Funzione di Autocorrelazione Parziale	34
3 Gestione dei dataset forniti	36
3.1 Descrizione dei dataset	36
3.2 Rinomina delle colonne dei dataset	37
3.3 Gestione dei valori nulli	37
3.4 Scelta dell'indice di tabella per ogni dataset	39
3.5 Filtraggio dei dataset	39
4 Analisi del problema e Soluzioni	44
4.1 Prima soluzione	44
4.1.1 Idea generale	44
4.1.2 Implementazione	44
4.1.3 Analisi di complessità	52
4.1.4 Analisi dei risultati e Conclusioni	52
4.2 Seconda Soluzione	54
4.2.1 Idea generale	54

4.2.2	Implementazione	54
4.2.3	Analisi di complessità	57
4.2.4	Analisi dei risultati e Conclusioni	57
4.3	Conclusioni Finali	58
5	Bibliografia	59

1 Introduzione

In tale capitolo introduttivo verranno spiegati gli obiettivi e la suddivisione del tirocinio in merito al tempo disponibile, cercando di spiegare, in maniera sintetica, come è stato svolto ed organizzato.

1.1 Premessa

Lo scopo di questo elaborato è quello di descrivere l'esperienza di tirocinio svolta presso l'Università Degli Studi di Genova con durata complessiva di 300 ore con inizio 25 novembre 2022 e fine 27 febbraio 2023. Il tirocinio è stato svolto per la maggior parte del tempo da remoto con incontri, volti all'andamento di esso, sia mediante l'utilizzo di Teams (piattaforma sviluppata da Microsoft) che in presenza, presso il dipartimento.

1.2 Obiettivi

L'obiettivo di questo tirocinio è stato quello di provare a individuare, se esistenti, uno o più metodi, relativi all'analisi di serie temporali, che permettano l'analisi di una serie di dati relative a soggetti sani e patologici con lo scopo di analizzare il cammino.

I dati utilizzati sono stati analizzati precedentemente da un gruppo di ricerca del dipartimento mediante l'ausilio di tecniche differenti da quelle utilizzate durante il tirocinio. Come risultato finale, in caso di un metodo sicuro e soddisfacente all'analisi, i risultati ottenuti sarebbero serviti al gruppo di ricerca come un'ulteriore conferma delle analisi da loro eseguite e/o come un'analisi da un'ottica differente, da quella adottata da loro, possa comunque portare a conclusioni simili.

1.3 Tecnologie utilizzate

Come scelta tecnologica principale, per lo sviluppo dell'intero svolgimento del tirocinio, si è optato il linguaggio di programmazione python, dovuto alla sua semplicità, velocità (nella scrittura di codice) e ampia community, che fornisce molti dei pacchetti utilizzati per eseguire analisi di ogni genere. Per tenere traccia del lavoro, e per esplorare velocemente le modifiche eseguite, è stato utilizzato Git come VCS (version control system) e Github come servizio di hosting per i repository.

1.4 Suddivisione del lavoro

Per poter garantire la conoscenza necessaria allo sviluppo di un metodo relativo allo scopo del tirocinio, il lavoro è stato principalmente suddiviso in due fasi: studio dei metodi relativi all'analisi di serie temporali e ricerca di un metodo per l'analisi del problema, con successivo sviluppo.

Nella prima fase sono stati studiati i metodi ed applicazioni di tecniche volte allo studio di serie temporali, sia da un lato pratico che da un lato teorico. Per quanto riguarda il lato pratico, queste tecniche sono state studiate mediante la ricerca di

articoli e videotutorial online sull'applicazione di esse e poi, in una fase successiva, messe in pratica con piccoli esempi mediante l'utilizzo di dataset di vario genere, forniti in maniera gratuita da siti web trovati su internet, per poterne capire meglio il funzionamento. Per quanto riguarda il lato teorico di esse è stato necessario studiare una piccola base di statistica inferenziale ed altre nozioni generali, per interpretare al meglio i risultati e le tecniche utilizzate in ambito applicativo.

Nella seconda fase si è passati alla ricerca di un metodo, che utilizzi tecniche dell'analisi di serie temporali, per risolvere il problema richiesto. Prima di essere passare allo sviluppo vero e proprio, essendo che i dati forniti erano in uno stato "grezzo", è stato necessario applicare tutte le tecniche di manipolazione dei dati come filtraggio, rinomina delle colonne, tecniche per la sostituzione di valori nulli etc ... per poter ottenere un dataset pulito e lavorabile dal punto di vista applicativo.

2 Studio dei metodi e delle applicazioni

In tale capitolo verranno spiegati i metodi, le applicazioni ed i concetti generali studiati durante la prima fase del tirocinio, durata circa 1 mese, accostando ad ogni di essi una relativa implementazione e/o utilizzo. In primo luogo si troverà una definizione di serie temporale, seguita da una spiegazione dei metodi per manipolare i dati inerenti a serie temporali su python, le componenti principali di una serie temporale e la loro visualizzazione ed infine metodi per l'analisi di essi.

Molti degli esempi forniti in questa sezione fanno riferimento a dataset disponibili al sito “UCI Machine Learning Repository” [6] più precisamente, come esempio, è stato utilizzato un dataset relativo alla qualità dell’aria della città di Beijing [9] (Pechino).

2.1 Cos’è una serie temporale

In statistica descrittiva, una serie storica (o temporale) si definisce come un insieme di variabili casuali ordinate rispetto al tempo, ed esprime la dinamica di un certo fenomeno nel tempo. Le serie storiche vengono studiate sia per interpretare un fenomeno, individuando componenti di trend, di ciclicità, di stagionalità e/o di accidentalità, sia per prevedere il suo andamento futuro [20]. In altre parole una serie storica (o temporale) è un’insieme/serie di dati campionati ed indicizzati nel tempo ad intervalli regolari come ore, giorni o anni.

In termini più matematici: indichiamo con \mathbf{Y} il fenomeno (ad esempio il prezzo della benzina dall’anno 1970 all’anno 2010) ed indichiamo con \mathbf{Y}_t un’osservazione al tempo t , con t un numero intero compreso tra 1 a T , dove T è il numero totale degli intervalli o periodi. Una serie temporale viene espressa in questa maniera

$$\mathbf{Y}_t = \{\mathbf{Y}_1, \mathbf{Y}_2, \dots, \mathbf{Y}_T\}$$

Esempio (*Prezzo della benzina*). Se consideriamo come fenomeno \mathbf{Y} il prezzo della benzina dal 1970 al 2010 avremmo come numero totale di osservazioni (o numero totale di periodi) $T = 40$ dove:

- \mathbf{Y}_1 : prezzo della benzina all’anno 1970
- \mathbf{Y}_2 : prezzo della benzina all’anno 1971
- $\mathbf{Y}_T = \mathbf{Y}_{40}$: prezzo della benzina all’anno 2010

2.2 Metodi per la gestione dei dati

In questo sottocapitolo verranno spiegati i metodi studiati ed utilizzati per la gestione dei dati in python, più nello specifico: come caricare un dataset, una possibile rinomina delle colonne relative alle serie per una maggiore comprensione, scelta di un indice, individuazione dei valori nulli ed infine una sezione relativa al filtraggio.

Cosa viene inteso per dataset Per dataset si intende un insieme di serie (nella nostra applicazione temporali) relative ad un'unica applicazione.

Esempio (*Qualità dell'aria*). Consideriamo, per esempio, come applicazione le misurazioni di diversi parametri relativi alla qualità dell'aria di Genova: livello di CO₂, livello di SO₂, livello di NO₂ e temperatura in °C. In un dataset possiamo considerare ogni parametro come una serie temporale diversa ma indicizzata nel tempo in ugual maniera, quindi se queste misurazioni avvengono ogni ora avemmo per ogni istante di tempo t le misurazioni per ogni parametro in quell'istante.

- Y_1 : livello di CO₂, SO₂, NO₂ e temperatura in °C all'istante 1.
- Y_2 : livello di CO₂, SO₂, NO₂ e temperatura in °C all'istante 2.
- ...
- Y_T : livello di CO₂, SO₂, NO₂ e temperatura in °C all'istante T .

Ogni parametro in un dataset viene rappresentato come una colonna.

Da questo momento in poi, nel report, quando si parlerà di dataset varrà inteso il tipo `pandas.DataFrame`, cioè il modo in cui un dataset viene interpretato all'interno di python, mentre per serie si intenderà il tipo `pandas.Series` oppure un semplice tipo `list` di python.

2.2.1 Pacchetti utilizzati

Per effettuare tutte le manovre relative all'elaborazione e manipolazione dei dati, sono state utilizzate funzionalità fornite da pacchetti python come `pandas`, `numpy` e `scipy`, essi semplificano la scrittura di codice e velocizzano il tempo di sviluppo, organizzando in maniera ottimale i dati. Per essere utilizzati essi necessitano prima di essere installati tramite il gestore di pacchetti di python `pip` e successivamente importati.

Snippet per l'installazione dei pacchetti

```
pip install pandas
pip install numpy
pip install scipy
```

Snippet per il caricamento in python

```
import pandas as pd
import numpy as np
from scipy import signal
```

2.2.2 Caricamento di un dataset

Per poter utilizzare i dati all'interno di python è stata utilizzata la funzionalità di pandas `read_csv` dove, ogni colonna fa riferimento ad una serie.

Snippet

```
dataset = pd.read_csv('data.csv')
```

2.2.3 Rinomina delle colonne relative alle serie

In qualche caso, il primo passo da eseguire, è quello di rinominare le colonne relative ad ogni serie così da poter avere una rappresentazione più accurata del dataset. Tramite l'utilizzo della funzione `display` e del metodo `head` del dataset possiamo controllare i primi 5 valori di un dataset controllando anche così il nome di ogni colonna.

Snippet

```
display(dataset.head())
```

0	1	2013	3	1	0	6.0	6.0	4.0	8.0	300.0	81.0	-0.5	1024.5	-21.4	0.0	NNW	5.7	Tiantan		
1	2	2013	3	1	1	6.0	29.0	5.0	9.0	300.0	80.0	-0.7	1025.1	-22.1	0.0	NW	3.9	Tiantan		
2	3	2013	3	1	2	6.0	6.0	4.0	12.0	300.0	75.0	-1.2	1025.3	-24.6	0.0	NNW	5.3	Tiantan		
3	4	2013	3	1	3	6.0	6.0	4.0	12.0	300.0	74.0	-1.4	1026.2	-25.5	0.0	N	4.9	Tiantan		
4	5	2013	3	1	4	5.0	5.0	7.0	15.0	400.0	70.0	-1.9	1027.1	-24.5	0.0	NNW	3.2	Tiantan		

Figure 1: output del metodo head prima della rinomina delle colonne

Come si può notare nell'immagine 1 i nomi delle colonne non hanno nessun nome significativo, con il seguente esempio potremmo cambiare il nome delle colonne.

Snippet

```
# definizione di una lista di nomi per le colonne del dataset
dataset.columns = [ 'no',      'year',     'month',    'day',     'hour',
                     'pm2_5',    'pm10',     'so2',      'no2',      'co',
                     'o3',       'temp',     'pres',     'dewp',    'rain',
                     'wd',       'wspm',     'station']
```

```
display(dataset.head())
```

0	1	2013	3	1	0	6.0	6.0	4.0	8.0	300.0	81.0	-0.5	1024.5	-21.4	0.0	NNW	5.7	Tiantan	
1	2	2013	3	1	1	6.0	29.0	5.0	9.0	300.0	80.0	-0.7	1025.1	-22.1	0.0	NW	3.9	Tiantan	
2	3	2013	3	1	2	6.0	6.0	4.0	12.0	300.0	75.0	-1.2	1025.3	-24.6	0.0	NNW	5.3	Tiantan	
3	4	2013	3	1	3	6.0	6.0	4.0	12.0	300.0	74.0	-1.4	1026.2	-25.5	0.0	N	4.9	Tiantan	
4	5	2013	3	1	4	5.0	5.0	7.0	15.0	400.0	70.0	-1.9	1027.1	-24.5	0.0	NNW	3.2	Tiantan	

Figure 2: output del metodo head dopo la rinomina delle colonne

Come si può notare nell’immagine 2 assegnando la lista delle colonne come attuali nomi per le serie del dataset riusciamo ad ottenere un’interpretazione più accurata.

2.2.4 Scelta dell’indice

Molte delle funzionalità fornite da `pandas` ed altri pacchetti python richiedono che il dataset sia indicizzato nel tempo nel corretto modo. In figura 2 si può notare che la prima colonna senza nome e la seconda colonna con nome `no`, indichino il numero di riga per ogni misurazione, la differenza è che la prima colonna è generata automaticamente dal pacchetto `pandas`, ed impostata di default come indice, mentre la seconda con nome `no` è fornita direttamente dal file csv precedentemente caricato. Per l’analisi della maggior parte delle serie temporali un’indicizzazione per numero di riga non è significativa, sarebbe molto più conveniente lavorare avendo come indice di tabella la data ed ora di ogni effettiva misurazione. A questo proposito il dataset caricato ci fornisce delle colonne (`year`, `month`, `day` e `hour`) indicizzate per tempo e relative ad ogni misurazione, che possono essere riformattate insieme ed usate come indice per il dataset.

Snippet

```
# unificazione delle colonne relative al tempo per ogni
# istante di tempo t in una nuova colonna
new_index_column = []
for i in range(len(dataset.year)):
    new_index_column.append("%s/%s/%s %s:0:0"
                           % (dataset.day[i], dataset.month[i],
                              dataset.year[i], dataset.hour[i]) )

# elimina le colonne relative al tempo
del dataset["year"], dataset["month"],
dataset["day"], dataset["hour"],
dataset["no"]

# imposta/crea la nuova colonna date e converti in datetime
dataset['date'] = new_index_column
dataset['date'] = pd.to_datetime(dataset.date, dayfirst=True)

# imposta come index la nuova colonna date
dataset.set_index("date", inplace=True)
display(dataset.head())
```

	pm2_5	pm10	so2	no2	co	o3	temp	pres	dewp	rain	wd	wspm	station
date													
2013-03-01 00:00:00	6.0	6.0	4.0	8.0	300.0	81.0	-0.5	1024.5	-21.4	0.0	NNW	5.7	Tiantan
2013-03-01 01:00:00	6.0	29.0	5.0	9.0	300.0	80.0	-0.7	1025.1	-22.1	0.0	NW	3.9	Tiantan
2013-03-01 02:00:00	6.0	6.0	4.0	12.0	300.0	75.0	-1.2	1025.3	-24.6	0.0	NNW	5.3	Tiantan
2013-03-01 03:00:00	6.0	6.0	4.0	12.0	300.0	74.0	-1.4	1026.2	-25.5	0.0	N	4.9	Tiantan
2013-03-01 04:00:00	5.0	5.0	7.0	15.0	400.0	70.0	-1.9	1027.1	-24.5	0.0	NNW	3.2	Tiantan

Figure 3: output del metodo head dopo aver impostato l'indice

Come si può notare dall'output del metodo `head` nell'immagine 3 `date` è stato impostato come indice di tabella e quindi da questo momento in poi possiamo accedere al dataset, scegliendo le misurazioni interessate, utilizzando la data.

Periodo di campionamento Un'altra importante modifica possibile è l'impostazione del periodo di campionamento del dataset. Impostare un corretto indice non basta a massimizzare il corretto funzionamento delle funzionalità di analisi delle serie temporali, bisogna anche specificare l'istante di tempo che occorre tra una misurazione e l'altra. Per fare ciò `pandas` fornisce un metodo che imposta il periodo di campionamento a quello desiderato, nel nostro caso sappiamo che le misurazioni sono state campionate ogni ora.

Snippet

```
# imposta il periodo di campionamento
# del dataset ad ogni ora
dataset = dataset.asfreq("h")
```

2.2.5 Individuazione dei valori nulli e possibili soluzioni

La presenza di valori nulli in una serie può avere molteplici cause, ad esempio, l'impossibilità da parte dello strumento di campionare ad un certo istante di tempo *t*, se pensiamo ad una fotocamera che acquisisce delle coordinate relative ad un soggetto, l'uscita di esso dall'obbiettivo renderà i valori nulli a quell'istante di tempo.

Indifferentemente dal motivo per cui dei valori nulli sono presenti, in una serie o un dataset, la loro presenza può causare molti problemi, sia nel corretto funzionamento di alcune funzionalità per l'analisi sia perché avere dei valori in determinati punti della serie, in certe applicazioni, potrebbe essere un problema.

`pandas` fornisce dei metodi utili, e semplici, alla soluzione di questo problema ma ovviamente ogni problema è diverso e quindi, per applicazioni specifiche, potrebbe essere necessario cercare soluzioni differenti. In questa sezione ci limitiamo a descrivere le funzionalità fornite da `pandas` per la soluzione a questo problema.

Controllo Per controllare la presenza di valori nulli `pandas` fornisce un metodo chiamato `isna` che ritorna un dataset dove, per ogni misurazione, indica `True` se la misurazione è `NaN` altrimenti `False`. Sommando i valori `True` per ogni colonna

possiamo controllare quanti valori nulli sono presenti per ogni serie.

Snippet

```
# controllo valori nulli
dataset.isna().sum()

pm2_5      677
pm10      597
so2      1118
no2      744
co      1126
o3      843
temp      20
pres      20
dewp      20
rain      20
wd      78
wspm      14
station      0
dtype: int64
```

Figure 4: output della somma dei valori nulli relativi ad ogni serie del dataset

Come si puo notare dall'output del comando in figura 4 il nostro dataset contiene molteplici valori nulli, ora vediamo come poter risolvere il seguente problema

Back fill o Forward fill pandas tramite l'utilizzo della funzione `fillna` fornisce la possibilità di “riempire” i valori nulli in due diverse modalità:

- **Back fill**: permette di sostituire i valori nulli con la successiva misurazione valida.
- **Forward fill**: permette di sostituire i valori nulli propagando l'ultima valida misurazione alla prossima valida.

Entrambi i metodi gestiscono i valori nulli più o meno nella stessa maniera, la scelta di uno piuttosto che l'altro cambia da caso in caso dipendentemente dal risultato ottenuto dopo l'utilizzo di essi.

Per i nostri esempi il risultato nell'utilizzo di un metodo piuttosto che l'altro portava comunque ad un risultato soddisfacente, quindi vediamo ora un esempio di utilizzo.

Snippet

```
# riempimento dei valori nulli
# mediante il metodo di forward fill
dataset = dataset.fillna(method="ffill")
dataset.isna().sum()
```

```

pm2_5      0
pm10      0
so2       0
no2       0
co        0
o3        0
temp      0
pres      0
dewp      0
rain      0
wd        0
wspm      0
station    0
dtype: int64

```

Figure 5: output della somma dei valori nulli relativi ad ogni serie del dataset dopo l'utilizzo del metodo `fillna`.

Interpolazione Un possibile metodo, non fornito dalle funzionalità del pacchetto `pandas`, che potrebbe essere utilizzato è quello di interpolare i dati così da poter colmare i vuoti creati dai valori nulli. Questa funzionalità non è stata sviluppata in quanto non fine allo scopo di questo tirocinio ma, in certe applicazioni, si potrebbe voler utilizzare una metodologia basata su questa tecnica per ottenere una rappresentazione più accurata dei dati.

Altro metodo non convenzionale Un altro metodo non convenzionale, non presente tra le funzionalità fornite, è quello di poter eliminare i valori nulli dalle serie semplicemente eliminandoli. Ovviamente questo concetto di eliminare una misurazione nulla va contro a tutte le premesse fatte fino ad ora, non avendo così una “reale” serie temporale poiché mancherebbe una misurazione ad un determinato istante di tempo t , e molte delle analisi che si vorrebbero poter fare su una serie risulterebbero inapplicabili. Tuttavia, in qualche applicazione particolare (come vedremo in seguito), una funzionalità che semplicemente elimina i valori nulli da una serie potrebbe tornare comoda.

Snippet

```

import math as math # import del pacchetto math

def delete_nan(series: pd.Series | list):
    """ elimina i valori nulla da una serie """
    new_series = []
    for idx, value in enumerate(series):
        if not math.isnan(value):
            new_series.append(value)
    return np.array(new_series)

```

2.2.6 Filtraggio

Nella maggior parte dei casi quando si parla di serie temporali fornite direttamente da apparecchiature che eseguono le misurazioni, i dati si presentano in maniera “grezza” ed è quindi necessario filtrarli per poter rimuovere una buona parte del rumore presente. Questo passaggio è molto importante se parliamo di dati non elaborati in quanto avere del rumore in una serie temporale o, più in generale, in qualsiasi tipo di segnale, porta a leggere delle misurazioni “false”. Solitamente il rumore indesiderato risiede nelle frequenze alte del segnale, quindi, a questo proposito, vediamo come poter filtrare una sorgente “grezza” di dati mediante l’utilizzo del modulo `signal` del pacchetto `scipy`.

Filtro di Butterworth Per poter rimuovere una buona parte del rumore, come scelta generale, in questo tirocinio, si è utilizzato il filtro di Butterworth che permette di regolare parametri come la frequenza di taglio e l’inclinazione (slope) della curva di taglio.

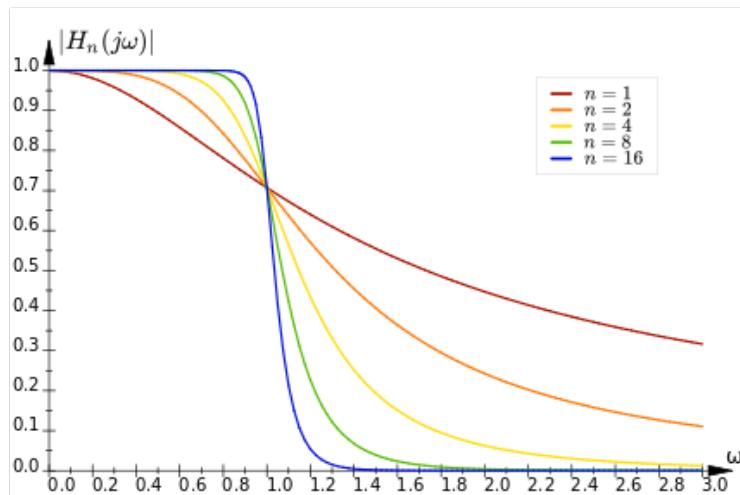


Figure 6: Filtro di Butterworth normalizzato.

Per poter apprezzare la differenza tra un segnale filtrato da un segnale “grezzo” seguirà un esempio su come questo filtro possa essere applicato tramite l’utilizzo delle funzionalità fornite dal modulo precedentemente citato.

Esempio (Utilizzo del filtro di Butterworth). Consideriamo un segnale la cui distanza tra ogni misurazione è di $\frac{1}{30}$ di secondo, presenta una frequenza di campionamento di 30Hz.

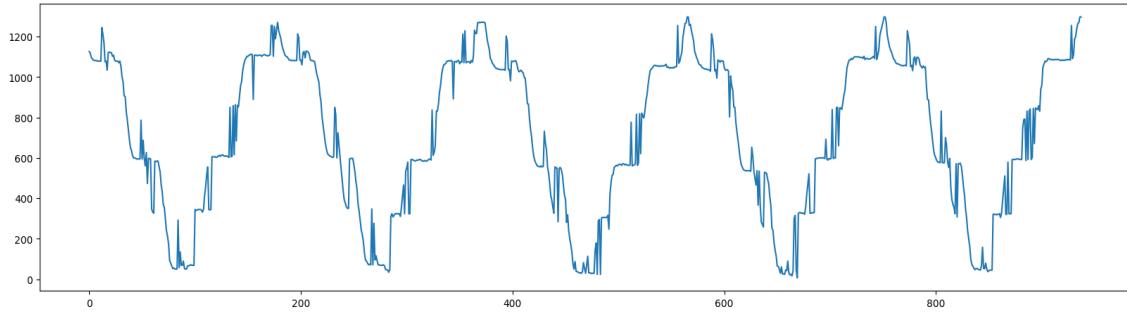


Figure 7: Segnale non filtrato.

In figura 7 possiamo notare come nel segnale in questione sia presente parecchio rumore, facilmente visibile da tutti i picchi presenti nel grafico. Proviamo ad applicare il filtro di Butterworth con una frequenza di taglio sulle 2Hz ed un'inclinazione della curva di taglio di 5, lasciando così passare le basse frequenze.

Snippet

```
# slope/inclinazione della curva di taglio
inclinazione = 5

# frequenza di taglio
frequenza_taglio = 2

# frequenza di campionamento
frequenza_di_campionamento = 30

# creazione del filtro di Butterworth
b, a = signal.butter(inclinazione,
                      frequenza_taglio,
                      fs=frequenza_di_campionamento)

# applicazione del filtro sul segnale
segnalet_filtrato = signal.filtfilt(b, a, series)
```

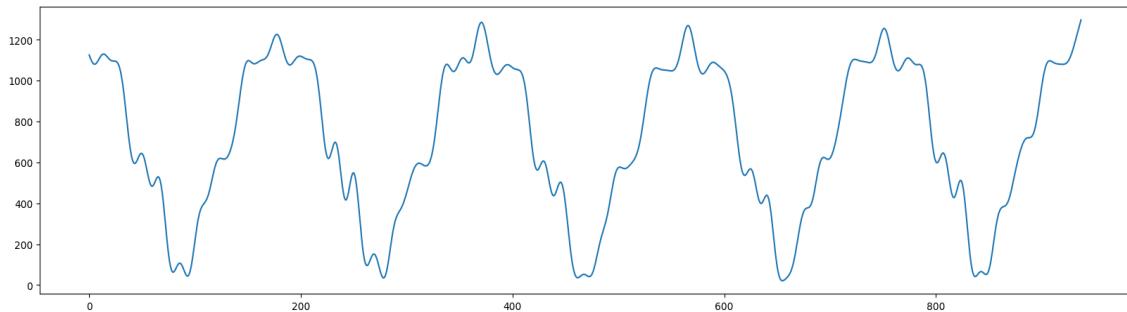


Figure 8: Segnale filtrato.

Nel grafico in figura 8 viene rappresentato il segnale dopo l'applicazione del filtro passa basse di Butterworth, come si può notare, una buona parte del rumore non è più presente.

Scelta dei parametri e del filtro La scelta del filtro, e dei suoi parametri, cambia da applicazione ad applicazione in base al tipo di filtraggio che si vuole applicare ai dati. Per fare un esempio, se vogliamo rimuovere le frequenze alte sceglieremo un filtro passa-basse con una frequenza di taglio ed ordine che ci permettano di ottenere un risultato soddisfacente, mentre se volgiamo rimuovere le frequenze basse sceglieremo un filtro passa-alte con gli adeguati parametri.

2.3 Componenti di una serie temporale

In questo sottocapitolo ci soffermeremo a spiegare le componenti principali che compongono una serie temporale per poterne analizzare l'andamento e/o eventuali pattern ricorrenti.

Molte volte è utile suddividere una serie temporale in più diverse componenti distinte per poterne analizzare singolarmente il loro comportamento e quindi successivamente riuscire ad inferire sul generale andamento della serie. Possiamo quindi pensare ad una serie temporale come un insieme di 3 componenti principali: Trend (andamento/tendenza), Stagionalità e Residui (più comunemente detta Noise).

2.3.1 Trend

La componente di trend è un pattern nei dati che mostra il movimento (andamento) di una serie verso valori relativamente più alti o più bassi in un lungo periodo di tempo. In altre parole, il trend si osserva quando la serie temporale presenta una pendenza crescente o decrescente. La tendenza di solito si verifica per un certo periodo di tempo e poi scompare, non si ripete. Ad esempio, una nuova canzone, che diventa di tendenza per un po' di tempo e poi scompare. Non c'è alcuna possibilità che torni in tendenza [8].

Il trend potrebbe essere:

- **Uptrend:** L'analisi delle serie temporali mostra un andamento generale al rialzo, quindi si tratta di Uptrend.
- **Downtrend:** L'analisi delle serie temporali mostra un andamento al ribasso, quindi si tratta di un downtrend.
- **Trend orizzontale o stazionario:** Se non si osserva alcun pattern, si parla di trend orizzontale o stazionario.

Se pensiamo al trend come ad una retta, essa sarà la retta che meglio approssima i dati. Se la retta ha coefficiente angolare positivo allora essa mostrerà un andamento generale al rialzo, se ha coefficiente angolare negativo allora essa mostrerà un andamento generale al ribasso, mentre se avrà coefficiente angolare uguale a 0 non si osserverà alcun pattern. Ovviamente nella realtà non sempre una retta basta ad avere una buona approssimazione dei dati, quindi esistono anche altri tipi di trend come il trend esponenziale.

Nella pratica dell'analisi delle serie temporali in python il trend viene stimato mediante l'utilizzo di una funzione chiamata *moving average* (trattata in una sezione successiva).

Esempio (esempio pratico di trend). Consideriamo come serie temporale una serie la cui per ogni osservazione abbiamo la temperatura media giornaliera nella città di Beijing a partire dal 01/01/2013 al 01/01/2017.

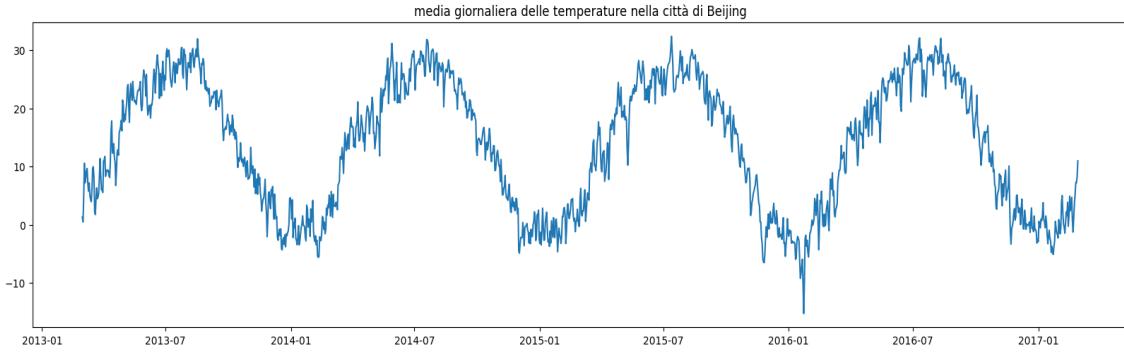


Figure 9: Grafico della temperatura media giornaliera nella città di Beijing.

Consideriamo ora un periodo di un anno (365 giorni) il trend avrà un grafico come il seguente

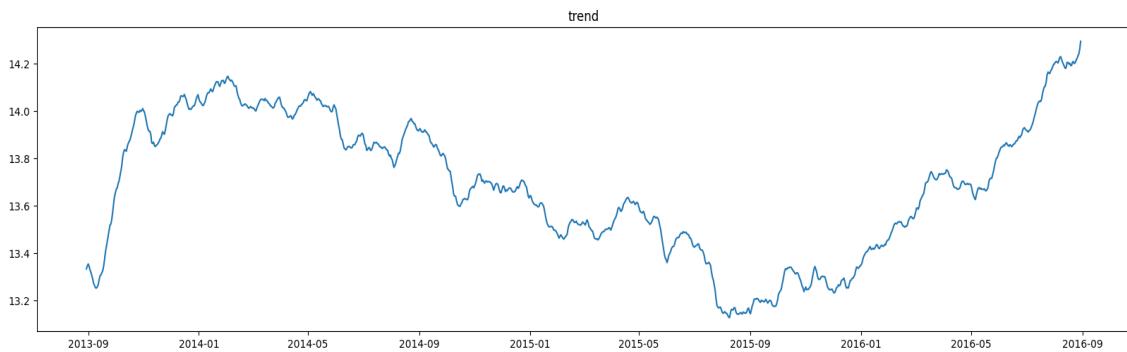


Figure 10: Trend della serie in figura 9.

Come possiamo vedere dal grafico in figura 10 il trend viene stimato utilizzando la media dei precedenti 365 giorni, ovviamente la scelta del periodo su cui stimare il trend influisce direttamente sul grafico.

2.3.2 Stagionalità

La stagionalità è un aspetto cruciale dell’analisi delle serie temporali. Poiché le serie temporali sono indicizzate in avanti nel tempo, sono soggette a fluttuazioni stagionali. Ad esempio, ci aspettiamo che le vendite di gelati siano maggiori nei mesi estivi e minori in quelli invernali.

La stagionalità può manifestarsi in diversi intervalli di tempo, come giorni, settimane o mesi. La chiave per l’analisi delle serie temporali è capire come la stagionalità influisce sulle nostre serie [3].

In sintesi possiamo quindi pensare alla stagionalità come un pattern che si ripete ad intervalli/periodi specifici nel tempo.

Esempio (esempio pratico di stagionalità). Se consideriamo come serie temporale la stessa serie utilizzata nell’esempio precedente del trend, quindi una serie la cui

ogni osservazione indica la temperatura media giornaliera nella città di Beijing a partire dal 01/01/2013 al 01/01/2017

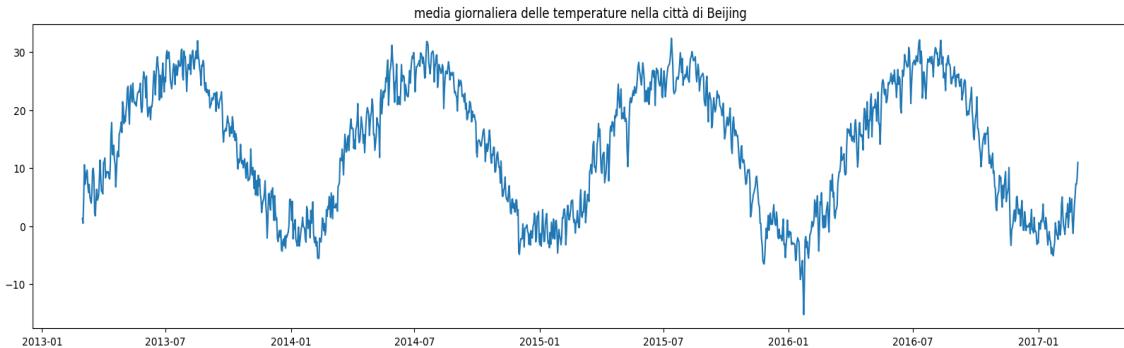


Figure 11: Grafico della temperatura media giornaliera nella città di Beijing.

e consideriamo un periodo di un anno (365 giorni) la stagionalità avrà un grafico come il seguente

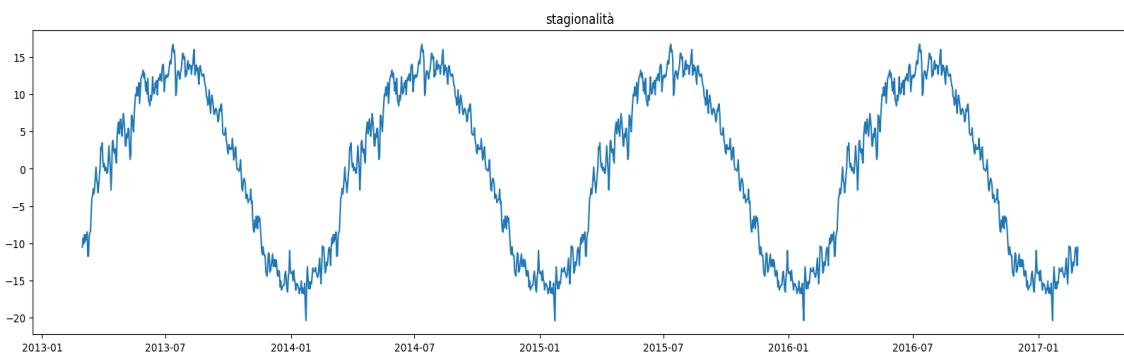


Figure 12: Stagionalità del grafico in figura 11.

Il grafico in figura 12 rappresenta la stagionalità della serie rappresentata in figura 11. Se guardiamo con occhio attento il grafico della stagionalità, possiamo notare come ad intervalli di un anno esso si ripeta, creando così il pattern stagionale.

2.3.3 Residui

La componente di residui, o più comunemente detta noise, può essere considerata come la parte restante tra il trend e la stagionalità, quindi una sorta di errore.

Un metodo per poter capire meglio come rappresenta la componente di residui è considerare, ad esempio, una serie temporale con trend orizzontale e nessun pattern stagionale, ed applicare la definizione di residui, quindi tutto quello che rimane tolto il trend e la stagionalità.

Esempio (esempio pratico di stagionalità). Se consideriamo la serie temporale utilizzata negli esempi del trend e della stagionalità, la componente dei residui avrà un grafico come il seguente

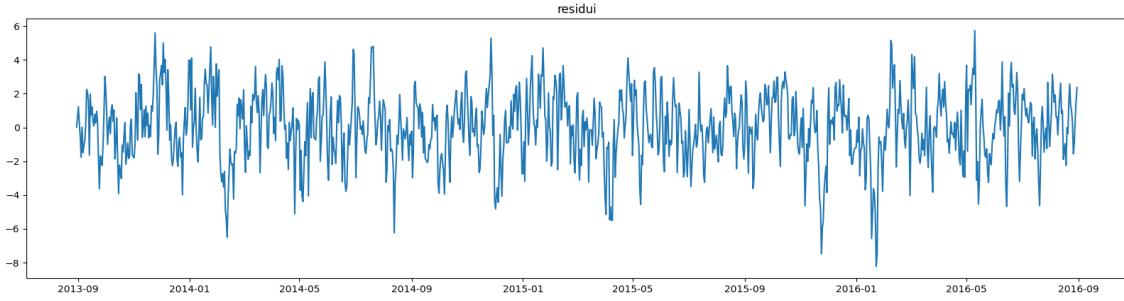


Figure 13: Residui della media giornaliera delle temperature nella città di Beijing.

2.3.4 Decomposizione di una serie

Esistono due modalità distinte per la decomposizione di una serie temporale nelle sue 3 componenti principali. La prima modalità è detta additiva in cui le componenti vengono semplicemente sommate, mentre la seconda è detta moltiplicativa dove le componenti, come suggerisce il termine, vengono moltiplicate.

Decomposizione additiva Una decomposizione additiva consiste nella somma delle componenti. Se consideriamo una serie temporale ad un istante t allora essa sarà composta da

$$y_t = S_t + T_t + R_t$$

dove y_t è l'osservazione, S_t è la componente di stagionalità, T_t è la componente di Trend ed R_t è la componente dei residui, tutti all'istante di tempo t .

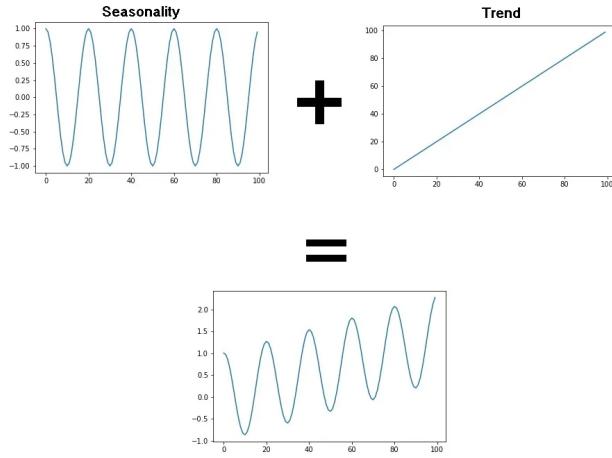


Figure 14: Decomposizione additiva (escluso residui). [ref imag [4]]

Decomposizione moltiplicativa Una decomposizione moltiplicativa consiste nella moltiplicazione delle componenti. Se consideriamo una serie temporale ad un istante t allora essa sarà composta da

$$y_t = S_t \times T_t \times R_t$$

dove y_t è l'osservazione, S_t è la componente di stagionalità, T_t è la componente di Trend ed R_t è la componente dei residui, tutti all'istante di tempo t .

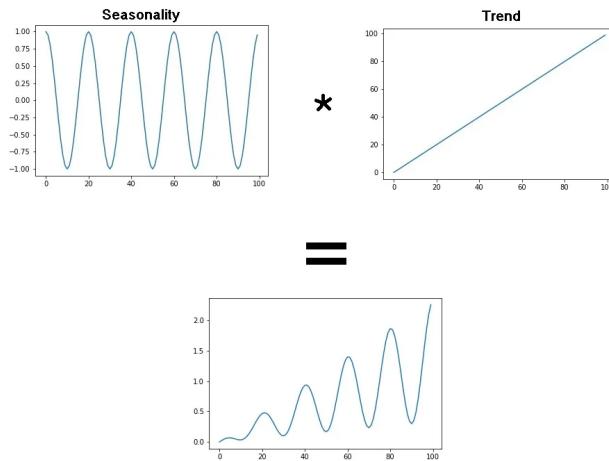


Figure 15: Decomposizione moltiplicativa (escluso residui). [ref imag [4]]

Scelta della modalità di decomposizione La scelta della modalità di decomposizione di una serie temporale è importanti poiché la sua decomposizione potrebbe apparire insensata se la modalità sbagliata. Una decomposizione additiva è principalmente appropriata se il magnitudo della stagionalità non varia con il livello della serie temporale mentre se, la variazione della componente di stagionalità, appare proporzionale al livello della serie una modalità moltiplicativa potrebbe essere più appropriata (per capire la seguente frase guarda le figure 14 e 15).

Decomposizione in python Vediamo ora come poter decomporre una serie temporale utilizzando la funzione `seasonal_decompose` fornita dalle funzionalità del pacchetto `statsmodels`.

Installazione del pacchetto

```
pip install statsmodels
```

Snippet

```
# import del pacchetto
from statsmodels.tsa.seasonal import seasonal_decompose

# periodo utilizzato per il calcolo di trend e stagionalità
periodo = 365
```

```
# decomposizione della serie
decomposition = seasonal_decompose(serie, period=periodo)

trend      = decomposition.trend      # trend
stag       = decomposition.seasonal  # stagionalità
residui   = decomposition.resid    # residui
```

2.4 Stazionarietà

In questo sottocapitolo verrà spiegato il concetto di stazionarietà di una serie temporale e come poter capire se la serie interessata sia stazionaria.

Stazionarietà di una serie temporale Per essere stazionaria una serie temporale deve soddisfare una lista di requisiti:

1. **Media costante nel tempo:** La media della serie non deve essere una funzione del tempo. Il grafico rosso, in figura 16, non è stazionario perché la media aumenta nel tempo [5].

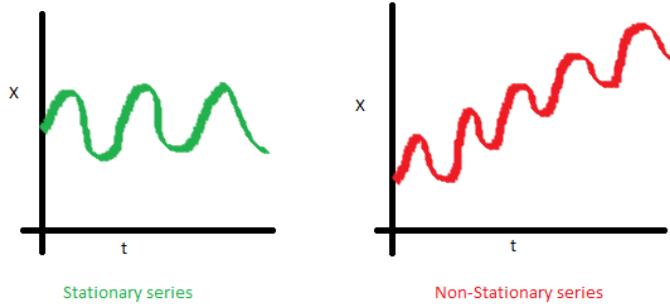


Figure 16: Media non costante nel tempo.

2. **Varianza costante nel tempo:** La varianza della serie non deve essere funzione del tempo. Questa proprietà è nota come omoschedasticità. Nel grafico rosso, in figura 17, si noti la variazione della varianza dei dati nel tempo [5].

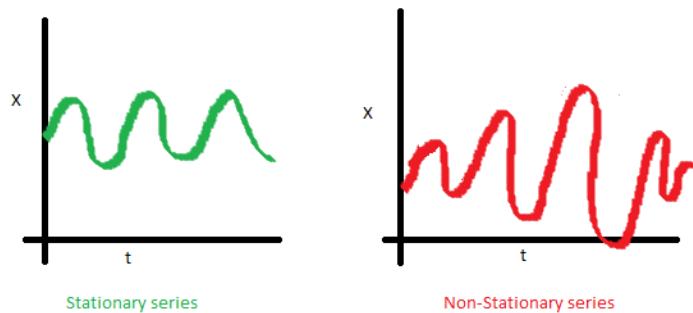


Figure 17: Varianza non costante nel tempo.

3. **Covarianza costante nel tempo:** Infine, la covarianza del termine i e del termine $(i+m)$ non deve essere funzione del tempo. Nel grafico, in figura 18, si può notare che lo spread diventa più vicino all'aumentare del tempo. Pertanto, la covarianza non è costante nel tempo per la "serie rossa" [5].

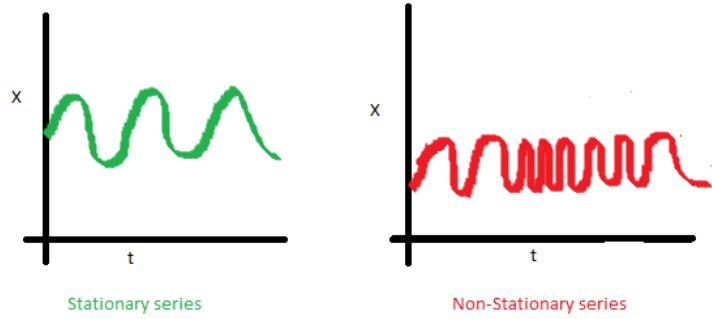


Figure 18: Covarianza non costante nel tempo.

Per capire meglio il concetto di covarianza costante nel tempo consideriamo una sequenza di variabili casuali, esse si definiscono con stazionarietà debole o stazionarietà della covarianza se:

- Tutti i termini della sequenza hanno la stessa media.
- La covarianza tra due termini qualsiasi della sequenza dipende solo dalla posizione relativa dei due termini e non dalla loro posizione assoluta.

Per posizione relativa di due termini si intende la distanza che li separa l'uno dall'altro nella sequenza mentre per posizione assoluta, si riferisce al punto in cui si trovano nella sequenza [12].

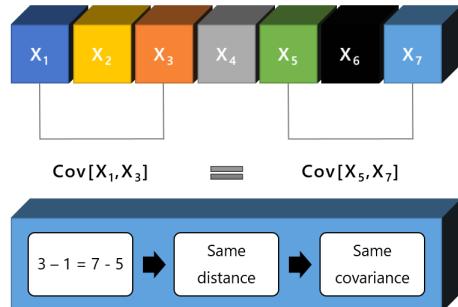


Figure 19: Stazionarietà della covarianza.

L'importanza di una serie stazionaria L'importanza di avere una serie temporale stazionaria deriva dal fatto che molti dei teoremi che a livello statistico valgono per le variabili casuali indipendenti valgono anche per variabili casuali stazionarie [5].

2.4.1 Dickey Fuller Test

Ci sono due modi per verificare la stazionarietà di una serie temporale. Il primo consiste nell'osservare i dati. Visualizzando i dati dovrebbe essere facile identificare una variazione della media o una variazione dei dati. Per una valutazione più accurata esiste il test di Dickey-Fuller [5].

In statistica, il test di Dickey Fuller verifica l'ipotesi nulla della presenza di una radice unitaria in un modello autoregressivo di una serie temporale. L'ipotesi alternativa

è diversa a seconda della versione del test utilizzata, ma di solito è la stazionarietà o la trend-stazionarietà (media e varianza costanti nel tempo) [16].

Modello autoregressivo Senza scendere troppo nei dettagli matematici diamo solamente la definizione di cos'è un modello autoregressivo: In statistica, econometria ed elaborazione dei segnali, un modello autoregressivo (AR) è una rappresentazione di un tipo di processo casuale; come tale, viene utilizzato per descrivere alcuni processi variabili nel tempo in natura, economia, comportamento, ecc. Un semplice modello AR di prim'ordine è

$$y_t = \rho y_{t-1} + u_t$$

dove y_t è la nostra osservazione all'istante di tempo t , ρ è un coefficiente e u_t è l'errore (assunta essere white noise, quindi stazionaria) [16].

In altre parole, se pensiamo alla nostra serie temporale, ogni osservazione ad istante t dipende da un numero arbitrario osservazioni precedenti più un errore.

Se prendiamo come esempio il modello autoregressivo di prim'ordine mostrato sopra allora esso si dice avere una radice unitaria se il coefficiente ρ è uguale ad 1. Se è presente una radice unitaria allora il processo non sarà stazionario e dipenderà da t (per un approfondimento migliore [21] e [16]).

Utilizzo del test nella pratica Per poter utilizzare questo test nella pratica verranno sfruttate le funzionalità del modulo `statsmodels.tsa.stattools` fornite dal pacchetto `statsmodels` precedentemente utilizzato. Più precisamente la funzione fornita dal modulo è il test di “Dickey-Fuller aumentato” che fornisce un numero negativo, questo più è negativo tanto maggiore è il rifiuto dell'ipotesi nulla che esista una unit root (radice unitaria) ad un certo livello di confidenza [13].

Snippet

```
# import del modulo stattools
import statsmodels.tsa.stattools as sts

# esecuzione del test
adf = sts.adfuller(serie)

# test statistic
print('T-stat : {}'.format(adf[0]))

# p-value
print('P-value : {}'.format(adf[1]))

# numero di osservazioni utilizzate
print('n-val-used: {}'.format(adf[3]))

# valori critici
```

```

print('Valori critici:')
print('\t1% : {}'.format(adf[4]["1%"]))
print('\t5% : {}'.format(adf[4]["5%"]))
print('\t10% : {}'.format(adf[4]["10%"]))

T-stat      : -2.026639644095428
P-value     : 0.27501605636147636
n-val-used: 1455
Valori critici:
    1% : -3.4348523191002123
    5% : -2.8635284734563364
   10% : -2.567828646449617

```

Figure 20: Output dello snippet sopra indicato (augmented dickey Fuller test).

In questo caso facendo riferimento all'output ottenuto, in figura 20, il valore interessato è **T-stat**, esso è da confrontare con i valori delle variabili relative alle percentuali di valore critico. Per ogni valore critico controlliamo se **T-stat** è maggiore o minore. Nel nostro caso **T-stat** è maggiore del valore critico che si riferisce al 10% quindi si ha più del 10% di possibilità che la nostra ipotesi nulla non sia rifiutata. In altre parole abbiamo più del 10% di possibilità che la nostra serie temporale non sia stazionaria. Solitamente per considerare una serie temporale stazionaria si necessita almeno di 5% quindi un livello di confidenza (che il test rifiuti l'ipotesi nulla e che quindi vede la nostra serie stazionaria) del 95%. Per semplificarci la vita possiamo direttamente controllare il valore di **p-value** (valore compreso tra $0 < \text{p-value} < 1$), se esso risulta minore di 0.05 possiamo considerare la serie stazionaria.

Parametri del test Per poter applicare correttamente il test di Dickey Fuller su python è necessario anche impostare i giusti parametri. Se non vengono passati parametri opzionali il test cerca di trovare la migliore impostazione per la serie. Per poter impostare correttamente i parametri del test bisognerebbe andare più nel dettaglio di quest'ultimo ma questo esce fuori dallo scopo finale del tirocinio.

Come rendere una serie stazionaria Nel caso ci trovassimo di fronte ad una serie temporale non stazionaria esistono diverse tecniche per renderla stazionaria. Una delle tecniche più comuni, e che solitamente funziona per molte applicazioni, è calcolare la prima differenza (differenza di prim'ordine). Se consideriamo come Y una serie temporale e con \hat{Y} la medesima dopo aver calcolato la prima differenza ogni osservazione di essa sarà definita come

$$\hat{y}_t = y_{t+1} - y_t$$

dove \hat{y}_t è un'osservazione di \hat{Y} ed y_t un'osservazione di Y , ad un istante di tempo t .

2.5 Smoothing

In questa sezione parleremo di alcuni dei metodi che permettono di “levigare” le serie temporali così da poter essere successivamente analizzate.

2.5.1 Moving average

In statistica, la moving average (media mobile), chiamata anche rolling mean, è un calcolo che consente di analizzare dei dati creando una serie di medie di diversi sottoinsiemi dell’insieme dei dati.

Data una serie di numeri e un sottoinsieme fisso, spesso chiamato finestra (window), il primo elemento della media mobile si ottiene prendendo la media del sottoinsieme fisso iniziale della serie di numeri. Poi il sottoinsieme finestra viene modificato “spostandosi in avanti”, escludendo il primo numero della serie e includendo il valore successivo nel sottoinsieme [18].

Da un punto di vista matematico se consideriamo Y la serie originale e \hat{Y} la serie generata dopo l’applicazione della funzione di rolling mean, e definiamo k dimensione della finestra, numero intero positivo, le osservazioni per ogni punto della rolling mean sono definite da

$$\hat{y}_t = \frac{1}{k} \sum_{n=1}^k y_{t-n}$$

con y_t e \hat{y}_t rispettivamente osservazioni della serie originale ed osservazioni della serie dopo l’applicazione della funzione di rolling mean, all’istante di tempo t .

Snippet

```
def rolling_mean(serie: list | pd.Series, window: int):

    # controllo della finestra
    if window <= 0 and not isinstance(window, int):
        raise Exception('la finestra deve essere un \
                        numero intero > 0')

    # serie rolling mean
    rolling_serie: list = []

    # calcolo della rolling mean
    for i in range(len(serie) - (window-1)):

        # calcolo singolo y hat
        rolling_serie.append(np.mean(serie[i: i + (window)]))

    return rolling_serie \
        if not isinstance(serie, pd.Series) \
        else pd.Series(
```

```

rolling_serie,
index=serie.index[(window-1)//2 : -(window-1)//2])

```

Esempio (Diversi valori per la finestra). Consideriamo come esempio la serie che descrive la temperatura giornaliera media nella città di Beijing. Se applichiamo la funzione di rolling mean con diversi valori per la finestra, otterremmo dei grafici come quelli mostrati in figura 21.

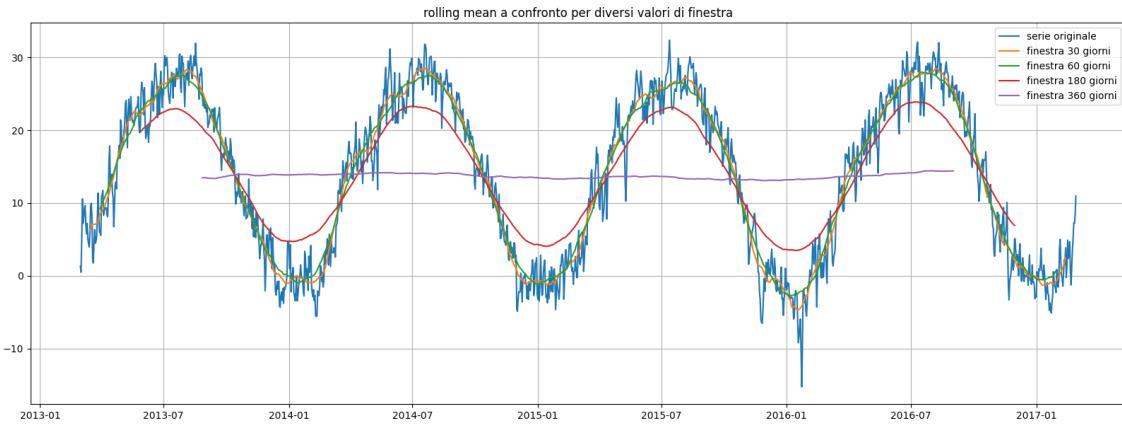


Figure 21: Funzione di rolling mean applicata a diversi valori per la finestra.

2.5.2 Exponential

Lo smoothing esponenziale è una tecnica di regola empirica per “levigare” i dati delle serie temporali utilizzando la funzione finestra esponenziale. Mentre nella rolling mean semplice le osservazioni passate vengono ponderate in modo uguale, le funzioni esponenziali vengono utilizzate per assegnare pesi esponenzialmente decrescenti nel tempo. Si tratta di una procedura di facile apprendimento e di facile applicazione per effettuare alcune determinazioni basate su ipotesi precedenti dell’utente, come la stagionalità [17].

Da un punto di vista matematico se consideriamo Y la serie originale e \hat{Y} la sua exponential smoothing serie, dove $\hat{y}_0 = y_0$, cioè la prima osservazione della exponential smoothing serie è inizializzata con il primo valore della serie originale, allora le osservazioni successive per ogni punto della exponential smoothing serie sono definite come

$$\hat{y}_t = \alpha y_t + (1 - \alpha)\hat{y}_{t-1}$$

con α numero intero positivo definito nell’intervallo $0 < \alpha < 1$, \hat{y}_t e y_t rispettivamente osservazioni della exponential smoothing serie e della serie originale all’istante di tempo t definito in $t \in [1, \dots, N]$ con N numero totale delle osservazioni della serie originale.

Snippet

```

def smooth_exponential(serie: list | pd.Series, alpha: float):

```

```

# controllo per alpha
if not isinstance(alpha, float) or alpha < 0 or alpha > 1:
    raise Exception("alpha deve essere \
                    nell'intervallo 0 < alpha < 1")

# calcolo della exponential serie
result = [serie[0]]
for n in range(1, len(serie)):
    result.append(
        alpha * serie[n] + (1 - alpha) * result[n - 1])

return result \
    if not isinstance(serie, pd.Series) \
    else pd.Series(result, index=serie.index)

```

Esempio (Diversi valori per alpha). Consideriamo come esempio la serie che descrive la temperatura giornaliera media nella città di Beijing. Se applichiamo la funzione di exponential smoothing per diversi valori di alpha, otterremmo dei grafici come quelli mostrati in figura 22.

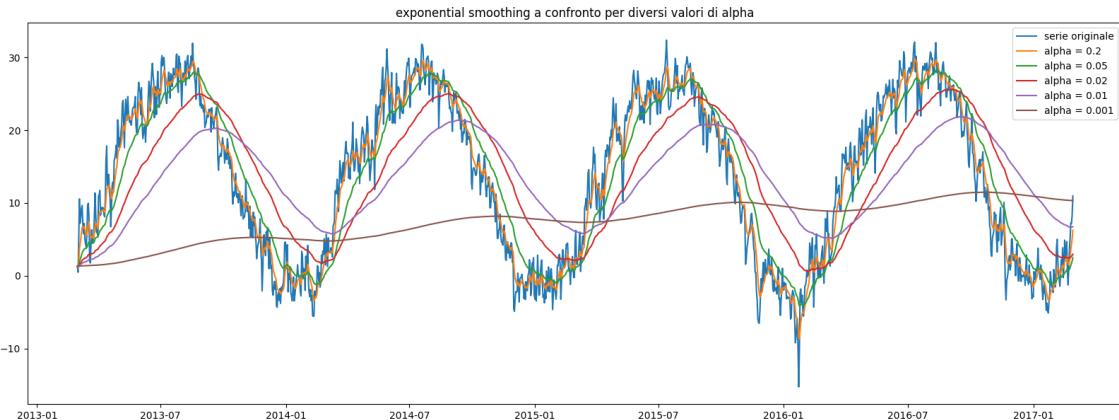


Figure 22: Funzione di exponential smoothing applicata a diversi valori di alpha.

2.5.3 Double Exponential

La tecnica di double exponential smoothing viene utilizzata nella previsione delle serie temporali quando i dati hanno una tendenza lineare (trend) ma non un andamento stagionale [11]. Questo tipo di smoothing utilizza lo stesso parametro alpha (α) della tecnica di exponential smoothing vista precedentemente più un ulteriore parametro beta (β) che regola l'ammontare della componente di trend.

La decomposizione della serie ci aiuterà, quindi otteniamo così due componenti: il livello ed il trend. Con la funzione di exponential smoothing precedente abbiamo imparato a prevedere il livello; ora applicheremo lo stesso exponential smoothing al trend, assumendo che la direzione futura delle variazioni della serie temporale dipenda dalle variazioni precedenti. Di conseguenza, otterremmo le seguenti serie di

funzioni [10]:

$$\begin{aligned}\hat{y}_0 &= y_0 \\ \ell_0 &= y_0 \\ b_0 &= y_1 - y_0\end{aligned}$$

$$\begin{aligned}\ell_t &= \alpha y_t + (1 - \alpha)(\ell_{t-1} + b_{t-1}) \\ b_t &= \beta(\ell_t - \ell_{t-1}) + (1 - \beta)b_{t-1} \\ \hat{y}_{t+1} &= \ell_t + b_t\end{aligned}$$

dove alpha, beta sono interi positivi nell'intervallo $(0, 1)$, ℓ_t componente che regola il livello, b_t componente che regola il trend, ed infine \hat{y}_t , y_t rispettivamente osservazioni all'istante di tempo t della double exponential serie \hat{Y} e della serie originale Y .

Esempio (Diversi valori per alpha). Consideriamo come esempio la serie che descrive la temperatura giornaliera media nella città di Beijing. Se applichiamo la funzione di double exponential smoothing per diversi valori di alpha e beta, otterremmo dei grafici come quelli mostrati in figura 23 e 24.

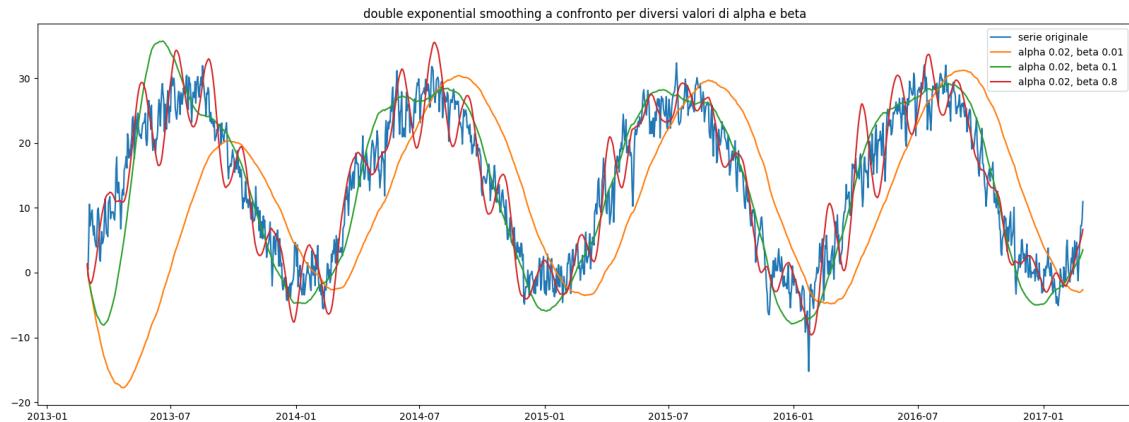


Figure 23: Funzione di double exponential smoothing applicata a diversi valori di alpha e beta (alpha costante).

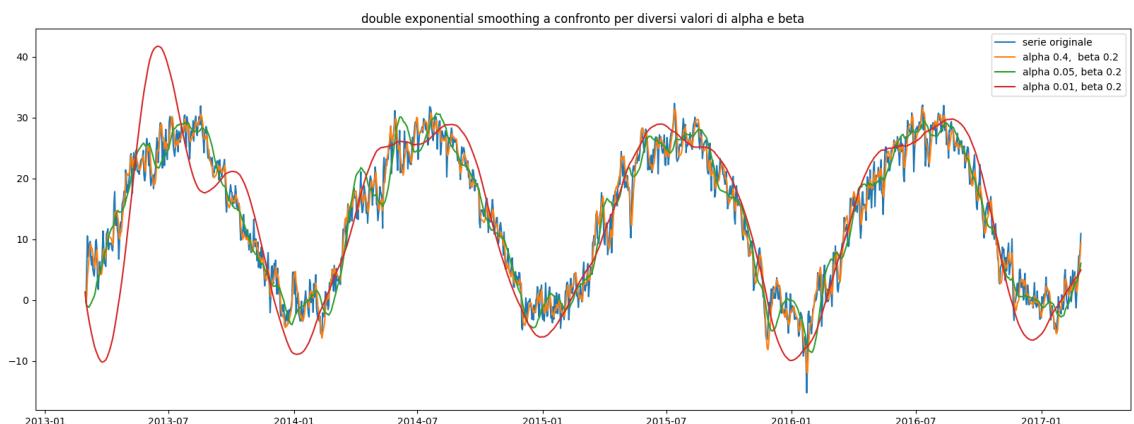


Figure 24: Funzione di double exponential smoothing applicata a diversi valori di alpha e beta (beta costante).

2.6 Autocorrelazione ed Autocorrelazione parziale

In questa sezione verranno presentate le funzioni di autocorrelazione ed autocorrelazione parziale, utilizzate nell'analisi di serie temporali per poter trovare pattern e la correlazione, diretta o indiretta, della serie con una sua versione spostata lungo l'asse temporale.

2.6.1 Funzione di Autocorrelazione

L'autocorrelazione definisce il grado di dipendenza tra i valori assunti da una funzione campionata nel suo dominio in ascissa (nel nostro caso il tempo). Se è dimostrata, l'autocorrelazione tra due valori, al cambiare delle peculiarità di uno di essi varierà anche l'altro.

L'autocorrelazione è uno strumento matematico usato frequentemente nella teoria dei segnali per l'analisi di funzioni o di serie di valori. Essa è la correlazione del segnale (o più in generale del valore di una variabile) con se stesso; in altre parole il segnale all'istante di tempo t viene confrontato con un altro valore di se stesso ritardato di una quantità τ (senza tale ritardo il segnale è logicamente sempre uguale) per verificare quanto si somigli (più precisamente quanto si corрeli) all'avanzare del tempo. Possiamo dedurre che se un segnale varia lentamente nel tempo, il valore degli istanti $y(t)$ e $y(t + \tau)$ sarà pressoché simile (l'autocorrelazione avrà segno positivo), mentre se varia rapidamente, il valore di tali istanti sarà molto diverso e l'autocorrelazione assume un valore prossimo allo zero. L'autocorrelazione si utilizza spesso per cercare porzioni periodiche che si ripetono all'interno di un segnale, in modo tale da determinare la presenza di un segnale periodico che è stato sepolto da un rumore, o identificare la frequenza fondamentale di un segnale [15].

Informalmente, è la somiglianza tra le osservazioni di una variabile casuale in funzione dell'intervallo di tempo che le separa [14].

Definizione matematica

Caso continuo Dato un segnale $f(t)$ continuo ed indicizzato nel tempo, l'autocorrelazione $R_{ff}(\tau)$ è definita come la cross-correlazione di $f(t)$ con se stesso avente un ritardo di τ

$$R_{ff}(\tau) = \int_{-\infty}^{\infty} f(t + \tau) \overline{f(t)} dt = \int_{-\infty}^{\infty} f(t) \overline{f(t - \tau)} dt$$

dove $\overline{f(t)}$ indica il complesso coniugato di $f(t)$. Nota come il parametro t nell'integrale è una variabile fittizia ed è solo necessaria a calcolare l'integrale. Non ha un significato specifico [14].

Caso discreto Nel caso discreto la funzione di autocorrelazione è definita come:

$$R_{ff}(\tau) = \mathbb{E} \left[f(t) \overline{f(t - \tau)} \right] = \mathbb{E} \left[f(t + \tau) \overline{f(t)} \right]$$

Normalizzazione del caso discreto La normalizzazione della funzione di autocorrelazione nel caso discreto ci fornisce la possibilità di avere una visualizzazione con valori compresi tra $[-1, 1]$. Sottraendo la media prima della moltiplicazione si ottiene la funzione di autocovarianza

$$K_{ff}(\tau) = \mathbb{E} \left[(f(t + \tau) - \mu) \overline{(f(t) - \mu)} \right] = \mathbb{E} \left[f(t + \tau) \overline{f(t)} \right] - \mu \bar{\mu}$$

dove μ è la media.

La funzione di autocorrelazione normalizzata viene definita come

$$\rho_{ff}(\tau) = \frac{K_{ff}(\tau)}{\sigma^2} = \frac{\mathbb{E} \left[(f(t + \tau) - \mu) \overline{(f(t) - \mu)} \right]}{\sigma^2}$$

Implementazione della funzione di autocorrelazione ed esempi Vediamo ora come poter implementare la funzione di autocorrelazione (normalizzata).

Snippet (*shift di una funzione*)

```
def shift(X, n):
    """ Shifta di n periodi la funzione X
    """
    # controllo per n
    if not isinstance(n, int):
        raise Exception('n deve essere un valore intero')

    X_copy = X.copy() # copia e rendi numpy array
    if not isinstance(X_copy, np.ndarray):
        X_copy = np.array(X_copy)

    # se si shifta troppo impostiamo n
    # alla lunghezza della funzione X
    if np.abs(n) > len(X):
        n = np.sign(n) * len(X)

    # dato n shiftiamo la funzione verso sinistra
    # dato -n shiftiamo la funzione verso destra
    for i in range(np.abs(n)):
        # togliamo il primo o l'ultimo valore
        # in base a dove vogliamo shiftare
        X_copy = X_copy[1:] if np.sign(n) == 1 else X_copy[:-1]

    # shifta la funzione aggiungendo uno
    # 0 in testa o in coda in base a dove
    # vogliamo shiftare
```

```

X_copy = np.insert(X_copy, 0 if np.sign(n) == -1
                   else len(X_copy), 0)

return X_copy.tolist() if not isinstance(X, np.ndarray) \
else np.array(X_copy)

```

Snippet (*singola autocorrelazione*)

```

def singola_autocorrelazione(X: list | np.ndarray,
tau: int, normalizzazione = True) -> float:

    """ Esegue una singola autocorrelazione data la latenza
    tau
    """
    # controllo per tau
    if not isinstance(tau, int) and tau < 0:
        raise Exception('tau deve essere un intero \
maggiori di 0')

    # controllo che X sia un array numpy
    if not isinstance(X, np.ndarray):
        X = np.array(X)

    if normalizzazione:
        mean = X.mean()
        var = X.var()

        #  $f(t-tau) - \mu$ 
        primo_termine = shift(X - mean, -tau)

        # coniugato( $f(t) - \mu$ )
        secondo_termine = np.conjugate(X - mean)

        #
        ... / sigma^2
    return np.mean(primo_termine * secondo_termine) / var

# espettazione[  $f(t-tau)$  * coniugato( $f(t)$ ) ]
return np.mean(shift(X, -tau) * np.conjugate(X) )

```

Snippet (*funzione di autocorrelazione*)

```

def funzione_autocorrelazione(
    X: list | np.ndarray,
    norm = True
) -> list | np.ndarray:

```

```

""" calcola la funzione di autocorrelazione
"""
autocorrelazione: list = []
# calcola la autocorrelazione singola per ogni
# possibile tau
for i in range(len(X)):
    autocorrelazione.append(
        singola_autocorrelazione(X, i,
                                  normalizzazione=norm)
    )

return autocorrelazione if not isinstance(X, np.ndarray) \
else np.array(autocorrelazione)

```

Esempio (Autocorrelazione). Prendiamo come esempio un segnale la cui velocità cambia di “poco” nel tempo.

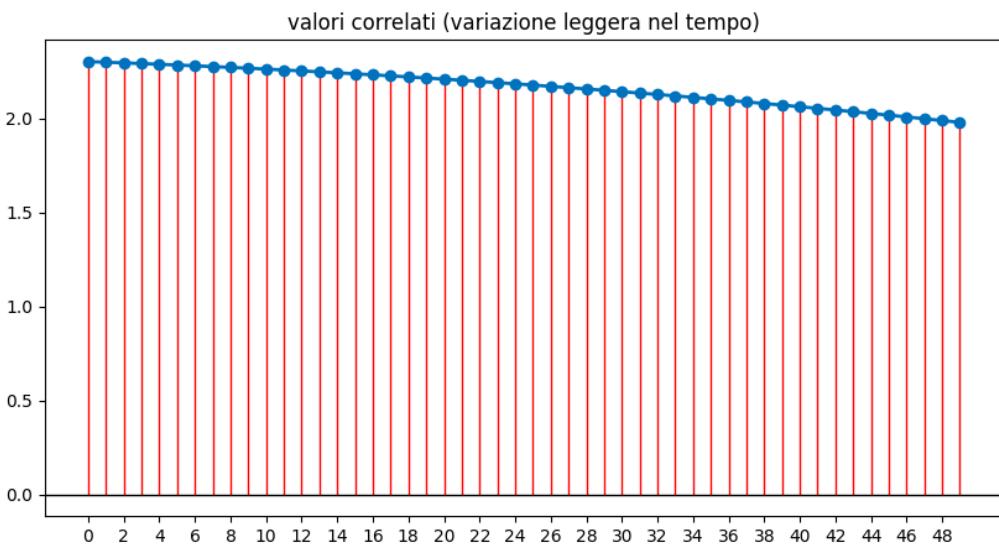


Figure 25: Autocorrelazione del segnale.

In figura 25 possiamo notare come l'autocorrelazione non normalizzata del segnale varia di poco nel tempo per segnali che variano lentamente.

Consideriamo ora più segnali la cui velocità cresce velocemente nel tempo.

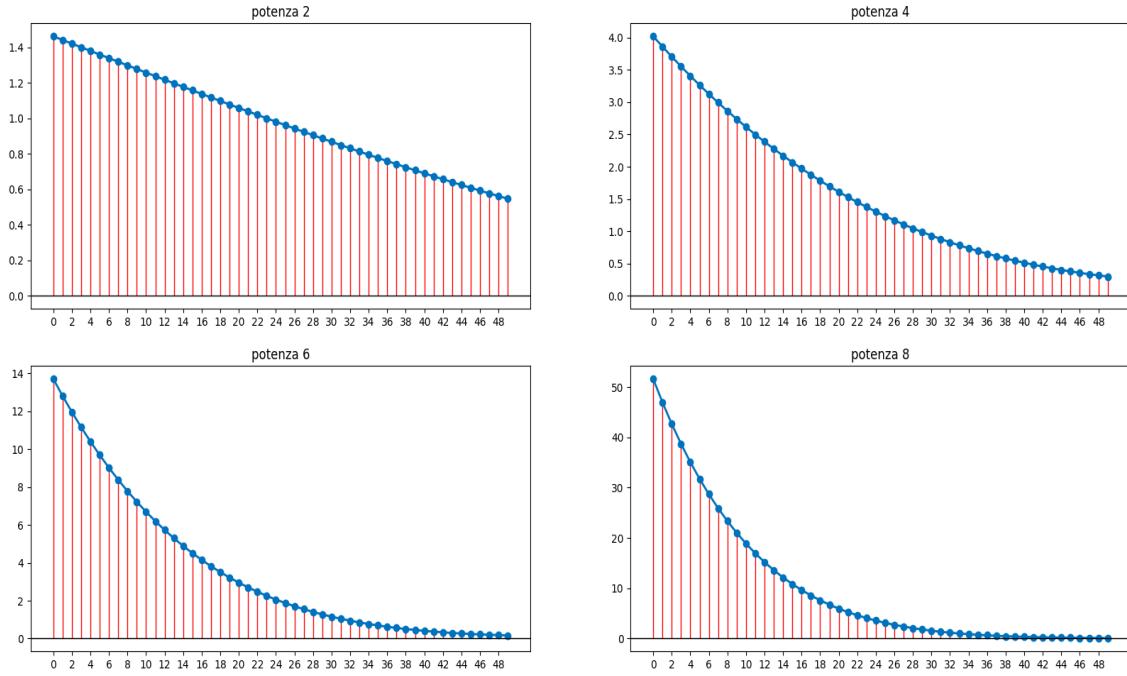


Figure 26: Autocorrelazione dei segnali.

In figura 26 possiamo notare come l'autocorrelazione non normalizzata dei segnali varia maggiormente nel tempo per segnali che variano velocemente.

Possiamo quindi osservare come segnali con minima variazione nel tempo avranno anche una minima variazione nell'autocorrelazione non normalizzata, mentre segnali che hanno un'ampia variazione tra un'osservazione e quella successiva avranno una maggiore variazione anche nell'autocorrelazione non normalizzata.

Un'ulteriore osservazione che nasce dagli esempi sopra è che i segnali più correlati tra loro sono i segnali con una variazione minore nel tempo, mentre i segnali meno correlati saranno i segnali con una maggior variazione.

2.6.2 Funzione di Autocorrelazione Parziale

L'autocorrelazione parziale, riassume la relazione tra un'osservazione di una serie temporale e le osservazioni di fasi temporali precedenti, come la normale autocorrelazione con la diversità che le relazioni delle osservazioni intermedie vengono rimosse. In sostanza, le correlazioni indirette vengono eliminate lasciando visibile solamente l'effetto diretto [1].

Potreste, ad esempio, essere interessati alla relazione diretta tra i consumi di oggi e quelli di un anno fa. Non vi importa nulla di ciò che accade nel mezzo.

Il consumo dei 12 mesi precedenti ha un effetto sul consumo degli 11 mesi precedenti e il ciclo continua fino al periodo più recente. Nelle stime di autocorrelazione parziale, questi effetti indiretti vengono ignorati [2].

Calcolo della funzione di autocorrelazione parziale La funzione di autocorrelazione parziale teorica di una serie temporale stazionaria può essere calcolata utilizzando l'algoritmo di Durbin-Levinson:

$$\phi_{n,n} = \frac{\rho(n) - \sum_{k=1}^{n-1} \phi_{n-1,k} \rho(n-k)}{1 - \sum_{k=1}^{n-1} \phi_{n-1,k} \rho(k)}$$

dove $\phi_{n,k} = \phi_{n-1,k} - \phi_{n,n} \phi_{n-1,n-k}$ per $1 \leq k \leq n-1$ and $\rho(n)$ è la funzione di autocorrelazione [19].

Quando è utile l'autocorrelazione parziale La funzione di autocorrelazione parziale gioca un ruolo molto importante, nell'analisi di serie temporali, per l'identificazione dei lag "importanti" per un modello autoregressivo (AR). Essendo che lo scopo del tirocinio è analizzare serie temporali senza eseguire nessuna predizione sui dati con modelli autoregressivi, non verrà data un'implementazione della formula presente sopra.

Pacchetto per l'utilizzo La funzione `pacf`, del modulo `statsmodels.tsa.stattools`, e la funzione `plot_pacf`, del modulo `statsmodels.graphics.tsaplots`, forniscono rispettivamente un'implementazione della funzione di autocorrelazione parziale ed il grafico di essa.

3 Gestione dei dataset forniti

Da questa sezione in poi il tirocinio entrò nella sua seconda fase di durata 1 mese e le successive sezioni e sottosezioni, inclusa la medesima, saranno in ordine cronologico di lavoro.

L'obbiettivo di questa sezione è spiegare le scelte applicative che sono state attuate per l'elaborazione dei dataset e descriverne il loro contenuto, per riuscirne a capire le analisi, le soluzioni ed i risultati ottenuti nelle seguenti sezioni.

3.1 Descrizione dei dataset

Durante il tirocinio sono stati forniti più dataset. Ogni dataset era relativo ad un soggetto, sano o patologico, con una specifica camminata, più nel dettaglio i dataset erano relativi a camminate normali, sulle punte o tacco punta. Ogni dataset era formato da più serie temporali, quindi indicizzate nel tempo, relative alla posizione di un particolare giunto (esempio: piede destro, piede sinistro, naso ...)

Campionamento dei dati I dati relativi ad ogni dataset sono stati ottenuti girando un video del soggetto in posizione laterale e lasciato camminare, avanti ed indietro per un corridoio, davanti all'obiettivo della telecamera. Successivamente, i video registrati, sono stati analizzati utilizzando una libreria open source che, con un algoritmo di machine learning, riconosce i punti relativi alle giunture interessate. Per ogni frame del video è stato quindi possibile fornire la posizione dei giunti in pixel, sia sull'asse delle ascisse che delle ordinate, in riferimento al pixel $(0, 0)$ del frame analizzato.

Il campionamento dati e la successiva trasformazione di essi in dataset, contenenti le posizioni dei giunti di ogni soggetto, è stata svolta dal gruppo di ricerca del dipartimento che successivamente ci ha fornito i dataset elaborati.

Esempio (Giuntura del naso). Se per esempio prendiamo la giuntura del naso relativa ad uno dei dataset, indifferentemente dal fatto che esso sia relativo ad un soggetto sano o patologico, essa presenta nel dataset due serie chiamate `x_naso` e `y_naso` che rappresentano rispettivamente la posizione x ed y ad ogni frame di acquisizione.

Frequenza di campionamento La fotocamera utilizzata per registrare i video ha una frequenza di campionamento di 30Hz , quindi ogni misurazione è distante dalla successiva circa $33,3\text{ms}$. Detto ciò la fotocamera riuscirà a captare periodicità che avvengono al massimo ogni $66,6\text{ms}$ (15Hz), logicamente per sapere quando un certo evento inizia e finisce abbiamo bisogno di almeno 2 misurazioni, più che sufficienti a captare periodicità che risultino significative all'analisi del movimento di un soggetto.

Descrizione delle serie relative ad un dataset I giunti forniti dai dataset sono: naso, torace, spalla destra, gomito destro, polso destro, spalla sinistra, gomito sinistro, polso sinistro, cresta iliaca,anca destra, ginocchio destro, caviglia destra,anca sinistra, ginocchio sinistro, caviglia sinistra, occhio destro, occhio sinistro, zigomo

destro, zigomo sinistro, piede sinistro (3 posizioni), piede destro (3 posizioni). Per ognuno di questi giunti è presente una misura della posizione sull'asse delle ascisse, una misura della posizione sull'asse delle ordinate ed una misura di likelihood cioè un numero tra 0 ed 1 che esprime quanto siamo sicuri di aver trovato il giunto nella posizione corretta (questa misurazione è stata ignorata).

3.2 Rinomina delle colonne dei dataset

I dataset sono stati forniti sprovvisti di una rappresentazione significativa per ogni colonna quindi, come prima operazione, sono state rinominate tutte le colonne.

Senza scendere troppo nei dettagli implementativi di come questa operazione è stata eseguita, poiché una tecnica per la rinomina delle colonne è stato fornita nella sezione precedente, vediamo come i dataset si presentano prima e dopo l'applicazione di essa.

	NaN	NaN.1	NaN.2	NaN.3	NaN.4	NaN.5	NaN.6	NaN.7	NaN.8	NaN.9	...
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...

Figure 27: Dataset prima della rinomina delle colonne.

	x_naso	y_naso	l_naso	x_torace	y_torace	l_torace	x_spalla_dx	y_spalla_dx	l_spalla_dx	x_gomito_dx	...
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...

Figure 28: Dataset dopo la rinomina delle colonne.

Come possiamo notare dalle figure 27 e 28 i nomi delle colonne dei dataset sono state rinominate, esse ora rappresentano meglio la realtà ed il loro accesso su python è facilitato, poiché è possibile accedervi utilizzando il comando `dataset_name.nome_giunto` invece che `dataset_name['nome_giunto']`.

3.3 Gestione dei valori nulli

Nei dataset forniti erano presenti dei valori nulli causati dall'uscita del soggetto dall'obbiettivo e quindi l'impossibilità, da parte dell'algoritmo di machine learning, di ottenere una posizione per i giunti interessati.

Essendo che l'obbiettivo finale del tirocinio non è quello di eseguire delle predizioni sulle serie temporali ma quello di inferire sui dati utilizzando le tecniche dell'analisi di serie temporali, i valori nulli sono stati semplicemente eliminati. In questa maniera consideriamo consecutivi dati che effettivamente non lo sarebbero, questo non è

stato un problema nelle successive analisi poiché la maggior parte dei valori nulli era presente nei momenti in cui il soggetto si girava per eseguire un’ulteriore camminata davanti all’obiettivo.

Guardiamo ora una esempio di serie prima e dopo l’eliminazione dei valori nulli così da avere un’idea di come i dati si presentano.

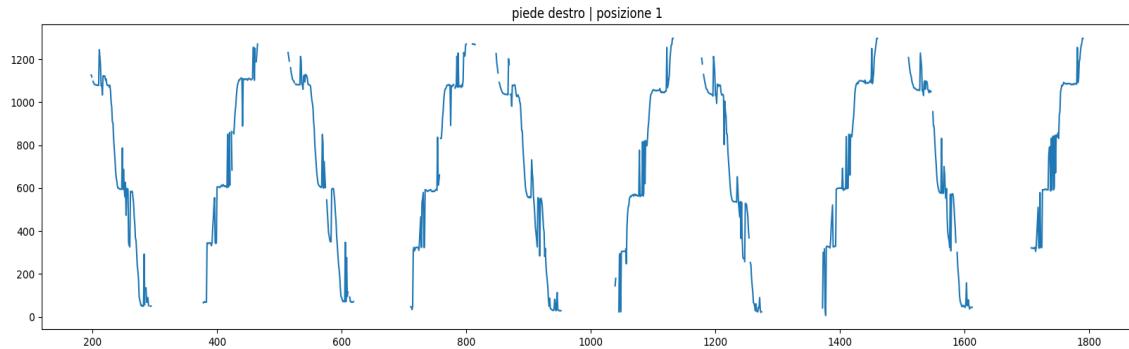


Figure 29: Piede destro posizione 1 con valori nulli.

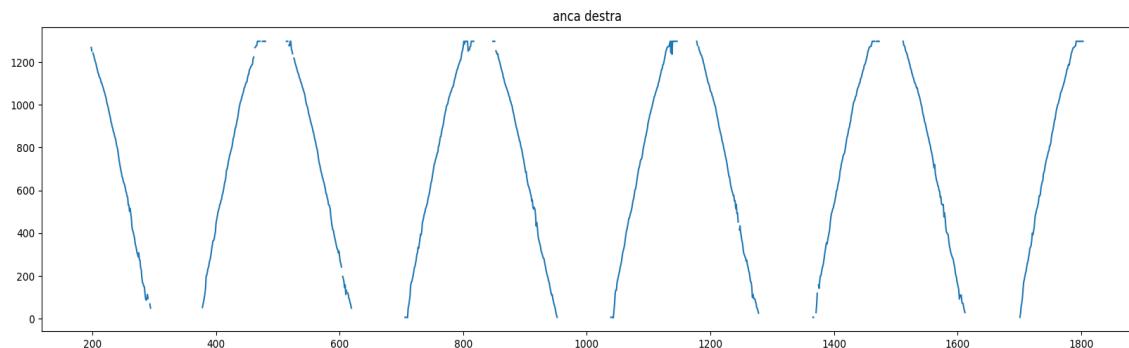


Figure 30: Anca destra con valori nulli.

Come possiamo notare dai grafici in figura 29 e 30 le serie contengono valori nulli nei punti in cui il soggetto esce dall’obiettivo girandosi per un’altra camminata davanti all’obiettivo.

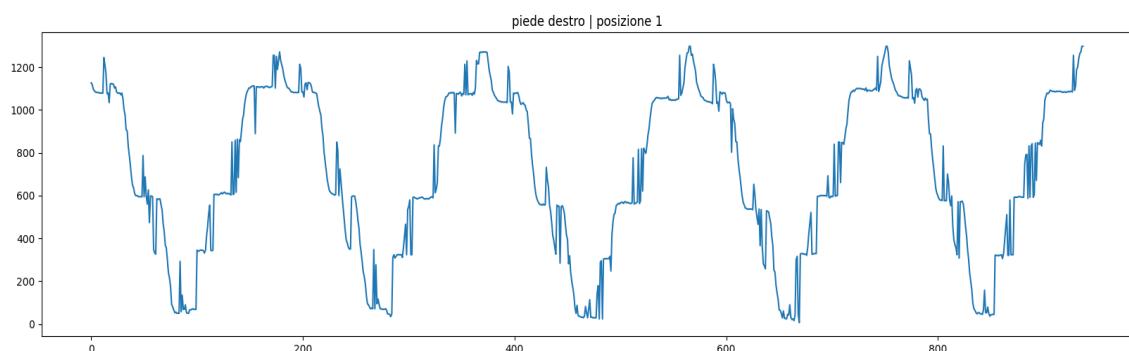


Figure 31: Piede destro posizione 1 senza valori nulli.

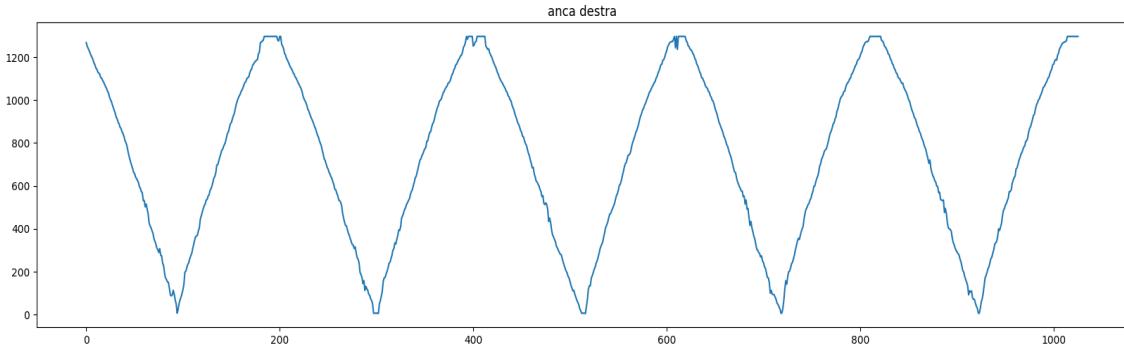


Figure 32: Anca destra senza valori nulli.

Guardando i grafici rappresentati in figura 31 e 32 possiamo notare come i valori nulli sono stati eliminati considerando così ogni camminata davanti all’obiettivo continua.

3.4 Scelta dell’indice di tabella per ogni dataset

Un’osservazione che nasce spontanea dall’osservazione dei grafici in figura 29, 30, 31 e 32 e dalle tabelle in figura 27 e 28 è che i dataset non sono indicizzati nel tempo utilizzando la classica indicizzazione come `anno/mese/giorno ora:minuti:secondi` ma sono indicizzati in base al frame di acquisizione quindi al numero di riga della tabella. Questo non è un problema dal punto di vista dell’analisi di ogni serie poiché noto il frame di acquisizione e la frequenza di campionamento si riuscirà facilmente a risalire ai secondi.

Nel nostro caso, sapendo che i dati sono stati acquisiti con una frequenza di campionamento di 30Hz, presa una qualsiasi misurazione ad un determinato frame, risaliremo al secondo di tempo con la seguente formula

$$\frac{\text{frame}}{30}$$

3.5 Filtraggio dei dataset

Come si può notare dai grafici in figura 29, 30, 31 e 32 le serie dei dataset contengono molto rumore causato da una sbagliata lettura della posizione, di ogni giunto, da parte dell’algoritmo di machine learning.

Come spiegato nella precedente sottosezione sul filtraggio, il rumore risiede principalmente nelle alte frequenze di un segnale.

In linea teorica si è pensato che tutte le frequenze più alte di 2Hz possano essere rumore, quindi in sostanza si andranno ad eliminare tutte le periodicità e movimenti periodici che avvengono sotto 0.5secondi.

Una soluzione migliore sarebbe stata quella di analizzare ogni serie e, per ciascuna di esse, applicare un filtro con una frequenza di taglio ottimale così da non eliminare informazioni importanti. Per lo scopo di questo tirocinio, come verrà spiegato in seguito, ci soffermeremo sull’analisi dei giunti relativi al piede e quindi una frequenza

di taglio di 2Hz potrebbe essere sufficiente a non perdere nessuna informazione importante.

Per fare un esempio pratico supponiamo che l'intervallo di tempo tra due successivi istanti di contatto con il terreno dello stesso piede (stride), per un soggetto sano, avviene in media ogni 0.8secondi, allora la frequenza di taglio scelta non elimina la periodicità interessata e riusciremo ad individuarla nel grafico.

Nella pratica se applichiamo una frequenza di taglio di 2Hz il segnale rimane con del rumore e delle periodicità indesiderate, questo è dovuto al cruciale ruolo che svolge l'ordine del filtro. Dopo svariati tentativi si è riuscito ad ottenere un risultato soddisfacente utilizzando una frequenza di taglio di 1.2Hz con un ordine di filtro 10.

Guardiamo ora delle serie prima e dopo l'applicazione del filtro con i parametri descritti precedentemente.

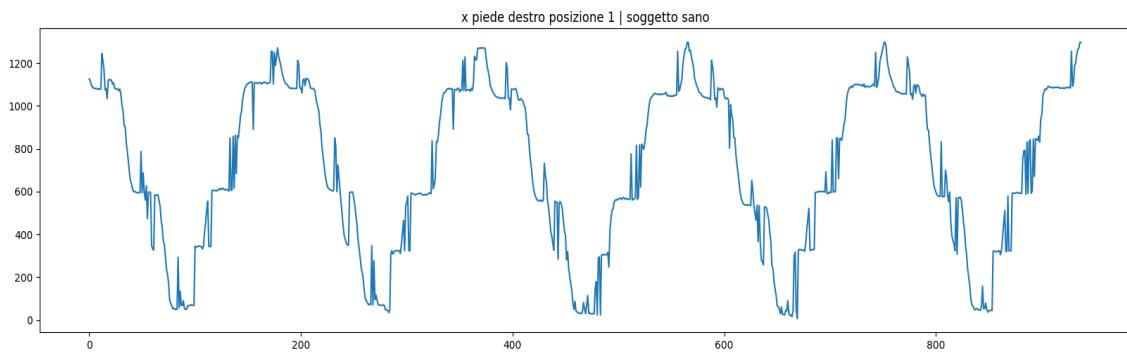


Figure 33: Piede destro posizione 1 di un soggetto sano prima dell'applicazione del filtro.

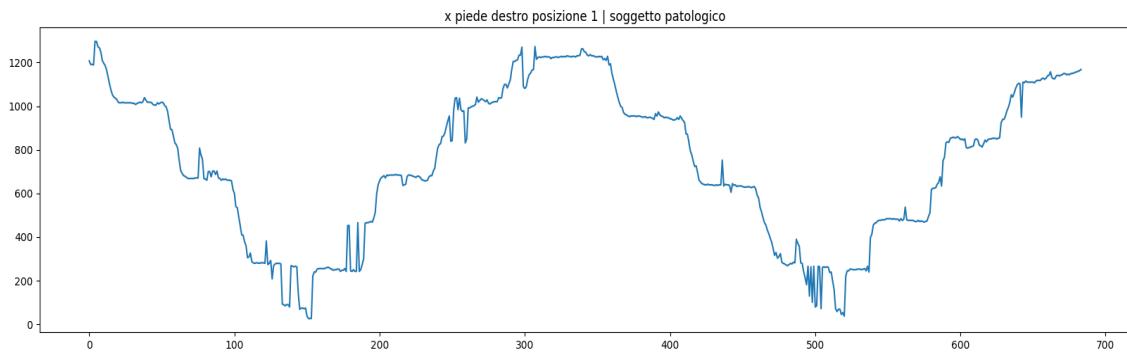


Figure 34: Piede destro posizione 1 di un soggetto patologico prima dell'applicazione del filtro.

Come possiamo notare dai grafici in figura 33 e 34 le serie non sono filtrate e contengono molto rumore.



Figure 35: Piede destro posizione 1 di un soggetto sano dopo l'applicazione del filtro.

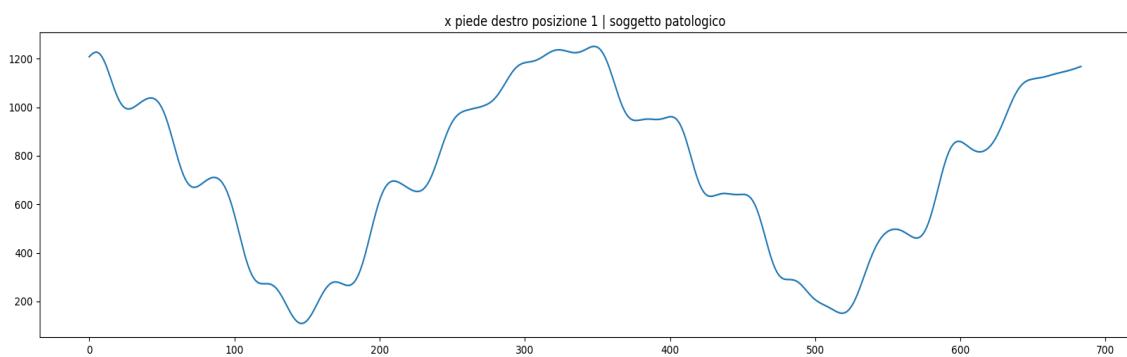


Figure 36: Piede destro posizione 1 di un soggetto patologico dopo l'applicazione del filtro.

Dai grafici in figura 35 e 36 possiamo notare come il filtro ha rimosso il rumore presente dalle serie e come l'effetto sinusoidale rimasto rappresenta l'andamento del passo di un soggetto. Nello specifico se consideriamo il coefficiente angolare delle rette tangenti ai punti del grafico, quindi la sua derivata prima, e consideriamo come inizio del passo il primo punto in cui la derivata è uguale a 0 la fine di un passo sarà il successivo punto in cui la sua derivata prima è uguale a 0.

Detto ciò un'ulteriore osservazione possibile sui grafici in figura 35 e 36 è che si può notare la presenza della periodicità relativa ad un'andata e ritorno del soggetto, essendo che sicuramente avviene più lentamente della periodicità di uno stride, essa risiede nelle frequenze “molto basse”. Per cercare di isolare maggiormente la caratteristica del passo si è deciso di eliminare questa periodicità andando a filtrare con un filtro passa alte le frequenze “bassissime” o almeno abbastanza basse da rimuovere solamente quella periodicità senza incidere sulle componenti che caratterizzano lo stride di un soggetto. Dopo qualche tentativo è stato deciso di utilizzare un filtro passa alte con frequenza di taglio 0.5Hz (periodicità che si ripetono ogni 2secondi e superiori) con un ordine di filtro di 10.

Vediamo ora i grafici delle due serie rappresentate in figura 35 e 36 a confronto prima e dopo l'ultima fase di filtraggio.

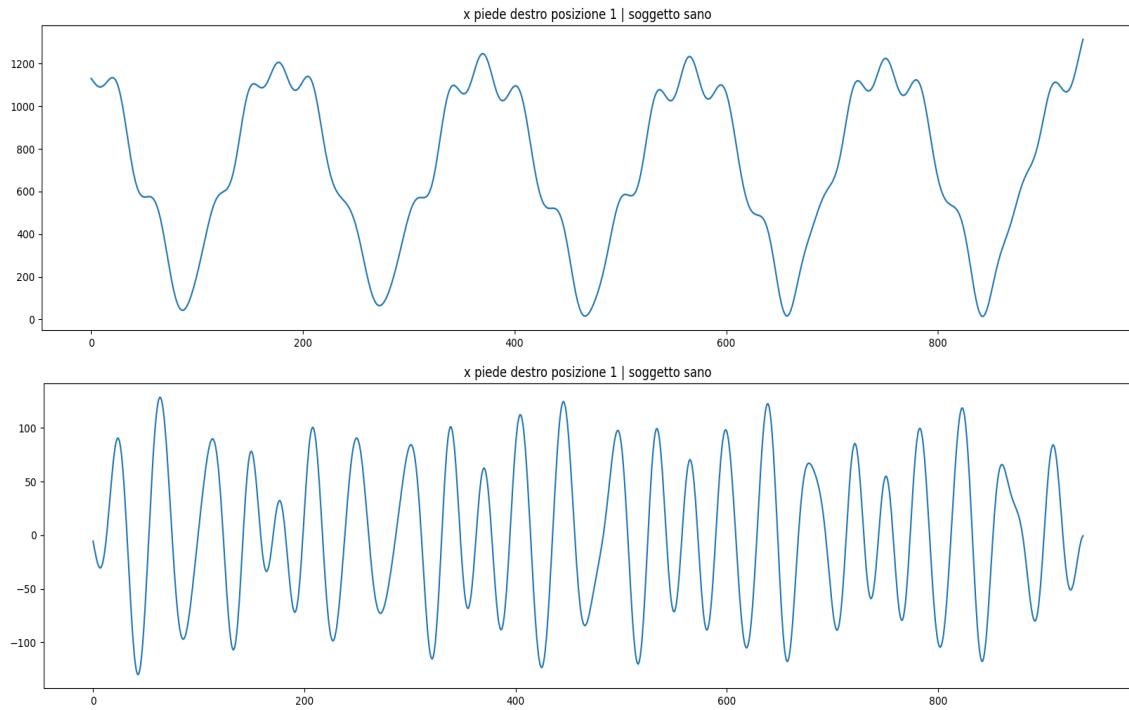


Figure 37: Piede destro posizione 1 di un soggetto sano prima e dopo l'applicazione dell'ultima fase di filtraggio.

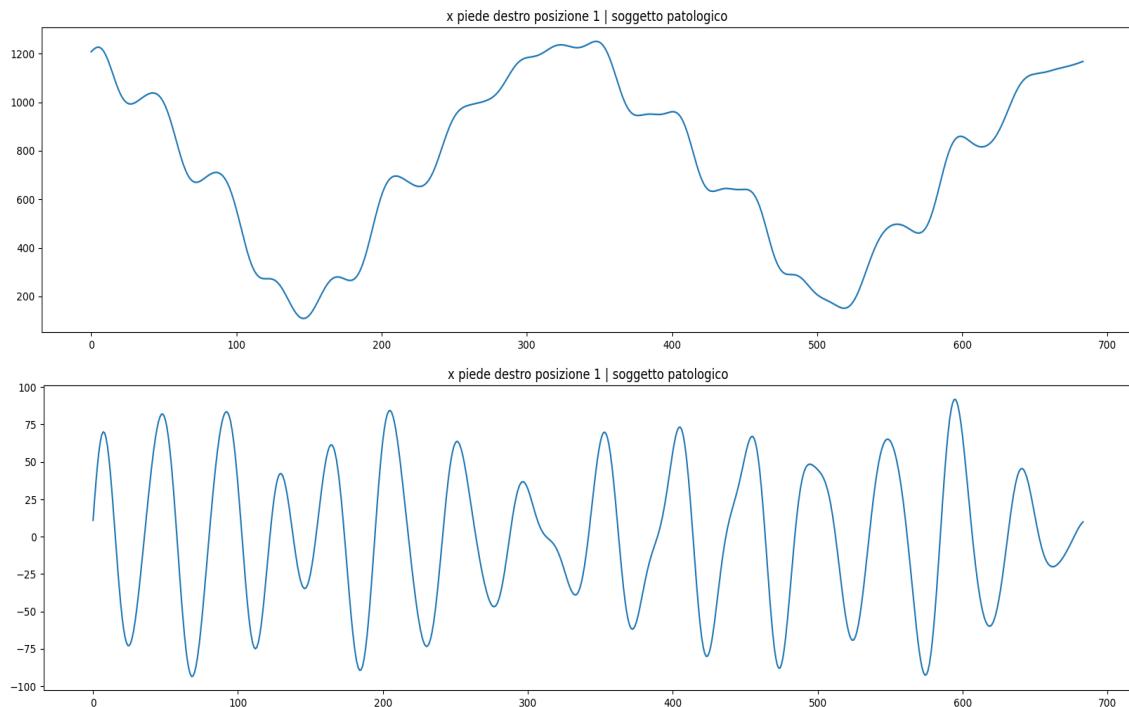


Figure 38: Piede destro posizione 1 di un soggetto patologico prima e dopo l'applicazione dell'ultima fase di filtraggio.

Come possiamo notare dai grafici in figura 37 e 38 la componente periodica di andata

e ritorno è stata eliminata e consideriamo il secondo grafico di ogni figura possiamo notare come i massimi locali delle componenti sinusoidali rappresentino l'inizio di ogni passo del soggetto, quindi i frame che intercorrono tra un punto di massimo locale e quello successivo rappresentano uno stride di un soggetto.

4 Analisi del problema e Soluzioni

In questa sezione verrà descritto il problema affrontato, le soluzioni scelte ed i rispettivi risultati ottenuti.

Problema affrontato Il problema affrontato chiedeva di trovare pattern, anomalie e/o inferire sui dati forniti, per ottenere, come risultato, informazioni su di essi mediante l'utilizzo di tecniche dell'analisi di serie temporali, e quindi successivamente capire se quest'ultime possano essere utilizzate come tecniche ausiliarie per l'analisi a problemi di questa tipologia.

Per riuscire risolvere il problema posto e per questioni di tempo, il problema è stato ridotto all'analisi dei soli giunti del piede, più in particolare, il tirocinio si è basato sulla ricerca di uno o più metodi utili a trovare la durata del passo (stride) di un soggetto con una successiva analisi dei risultati.

Risultati attesi Come risultato atteso è stata presa in considerazione l'eventualità della non riuscita dell'obiettivo finale, quindi l'impossibilità di trovare, tramite le tecniche dell'analisi di serie temporali, una possibile soluzione al problema.

In caso positivo, quindi di una possibile soluzione al problema posto, i risultati attesi potrebbero essere:

- Capacità da parte del metodo sviluppato di rilevare la durata del passo.
- Capacità da parte del metodo sviluppato di rilevare incorrettamente la durata del passo.
- Impossibilità da parte del metodo sviluppato di rilevare un passo.

4.1 Prima soluzione

In questa sezione verrà mostrata la prima soluzione al problema posto, nello specifico verrà spiegata in breve l'idea generale, l'implementazione di alcuni dei metodi utilizzati, l'analisi dei risultati, l'analisi della complessità ed infine le conclusioni.

4.1.1 Idea generale

L'idea generale, fulcro di questa soluzione, ruota intorno all'analisi della funzione di autocorrelazione. Quest'ultima viene utilizzata per riconoscere pattern o una sorta di correlazione tra i dati non visibile direttamente.

4.1.2 Implementazione

Come primo step implementativo è stato necessario scrivere una funzione che rendesse ogni serie stazionaria mediante l'ausilio del test di Dickey-Fuller ed il calcolo della prima differenza. Ovviamente per rendere una serie stazionaria esistono anche altri metodi diversi dalla prima differenza ma si è deciso di utilizzarle quest'ultima per la sua semplicità.

Snippet (*Prima differenza*)

```
def compute_first_diff(series: list | np.ndarray):
    """ Calcola la prima differenza
    """

    if not isinstance(series, np.ndarray):
        series = np.array(series)

    return series[1:] - series[:len(series)-1]
```

Snippet (*Metodo per rendere la serie stazionaria*)

```
def make_stationary(series, max_steps = 30):
    """ Rende la serie stazionaria mediante il calcolo
        della prima differenza
    """

    step = 0      # numero di step
    s_copy = series.copy()  # copia della serie

    # fino a quando il test ritorna che la serie
    # non è stazionaria oppure sono stati superati
    # il massimo di step
    while(sts.adfuller(series)[1] > 0.05
          and step < max_steps):

        # calcola la prima differenza
        serie = compute_first_diff(series)
        step += 1

        # se max_step è superato, non si è riusciti
        # a rendere la serie stazionaria
        if step > 30:
            series = s_copy

    return series
```

Dopo aver reso ogni serie stazionaria è stata applicata la funzione di autocorrelazione ad ognuna di esse, per capire meglio i risultati di questa operazione prendiamo in considerazione alcuni dei grafici relativi all'autocorrelazione di serie appartenenti a dataset dei soggetti 6 ed 12.

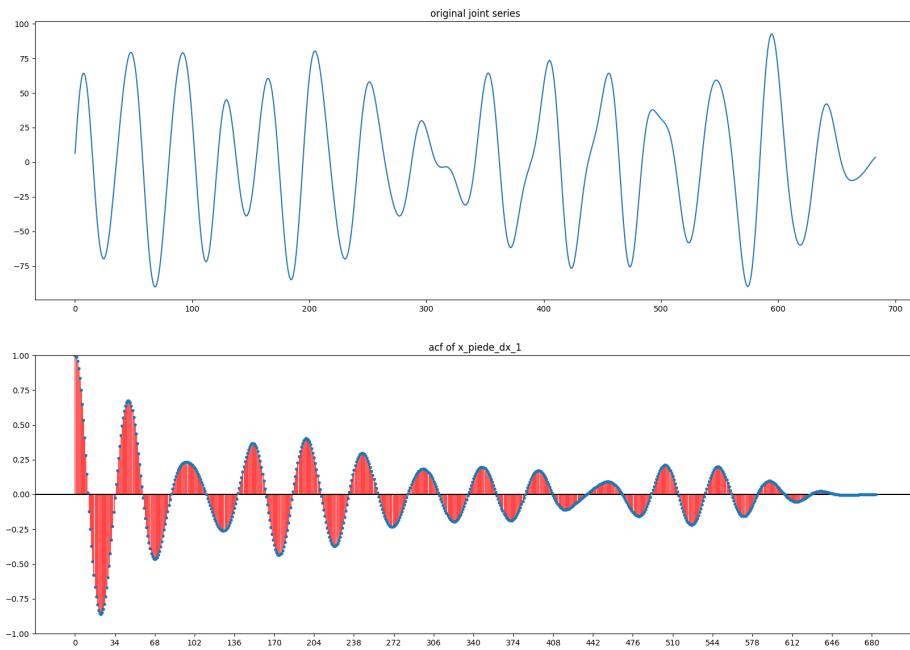


Figure 39: Soggetto 6 autocorrelazione x piede destro posizione 1.

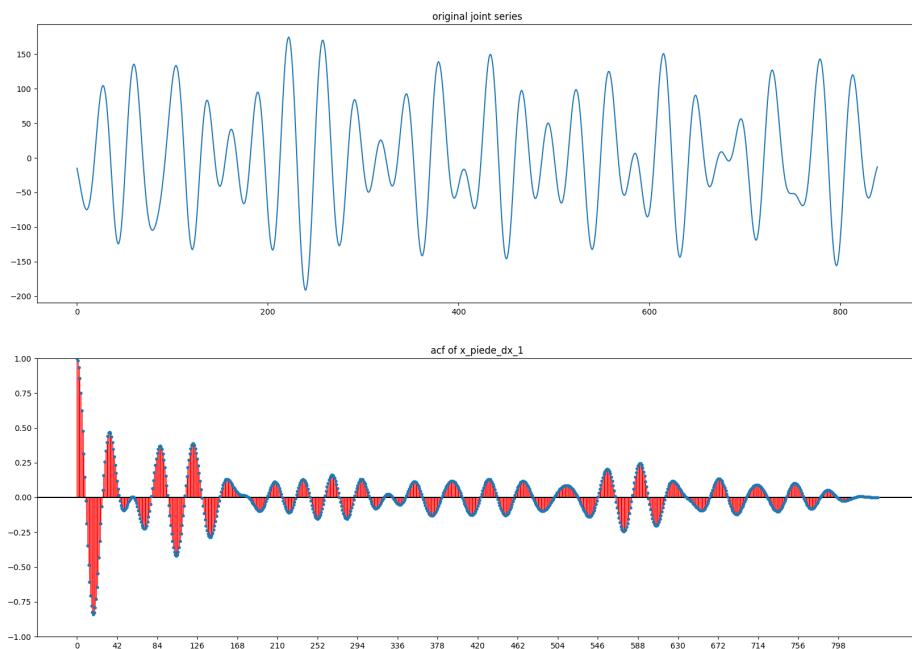


Figure 40: Soggetto 12 autocorrelazione x piede destro posizione 1.

In figura 39 e 40 possiamo vedere la serie originale in comparazione con il proprio grafico di autocorrelazione. Per entrambi i grafici troviamo i periodi/frame sull'asse delle ascisse mentre sull'asse delle ordinate troviamo l'ampiezza delle sinusoidi, per il grafico rappresentante i dati, ed il livello di correlazione, per il grafico dell'autocorrelazione (quanto la serie dei dati è correlata con una sua versione spostata di x frame/lag).

Nella maggior parte dei casi possiamo inoltre notare come i punti di massimo locale del grafico dell'autocorrelazione corrispondono all'inizio di un passo, più in precisione corrispondono alla ripetizione/pattern relativo ad un passo.

Per capire meglio come interpretare il grafico dell'autocorrelazione vediamo come essa si comporta con una normale sinusode (coseno e seno).

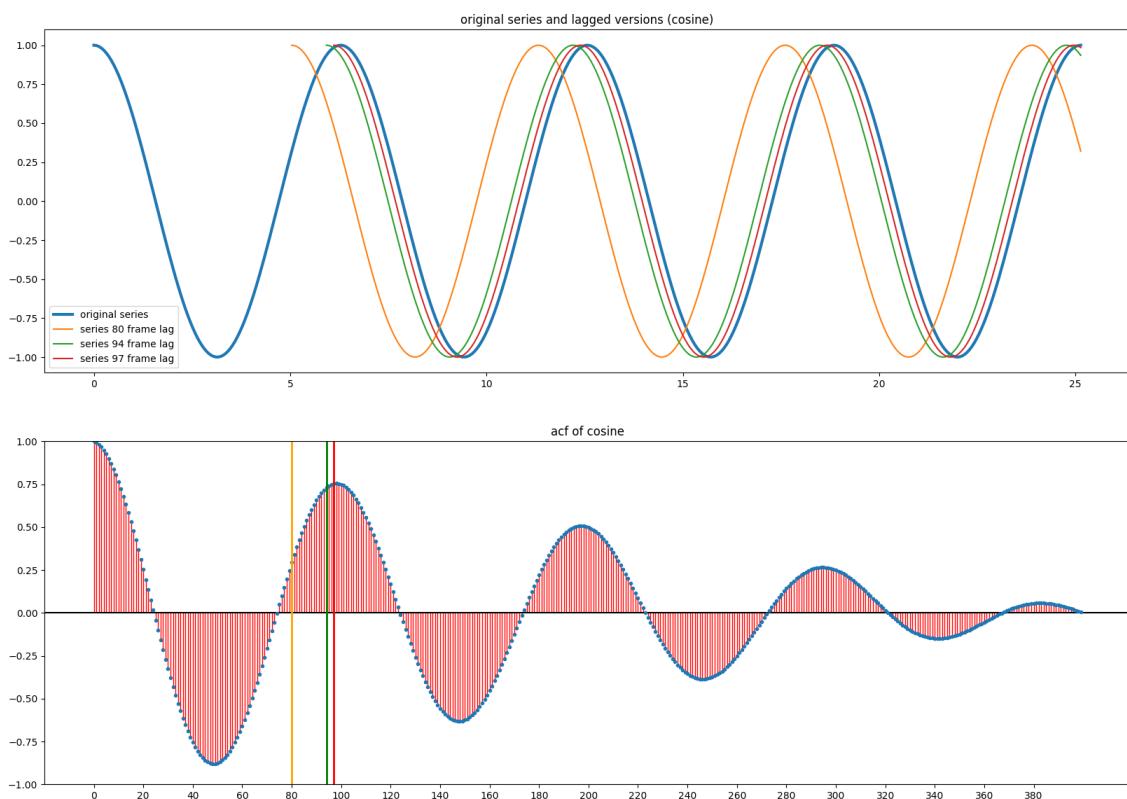


Figure 41: Grafico di un coseno con sue versioni shiftate ed autocorrelazione.

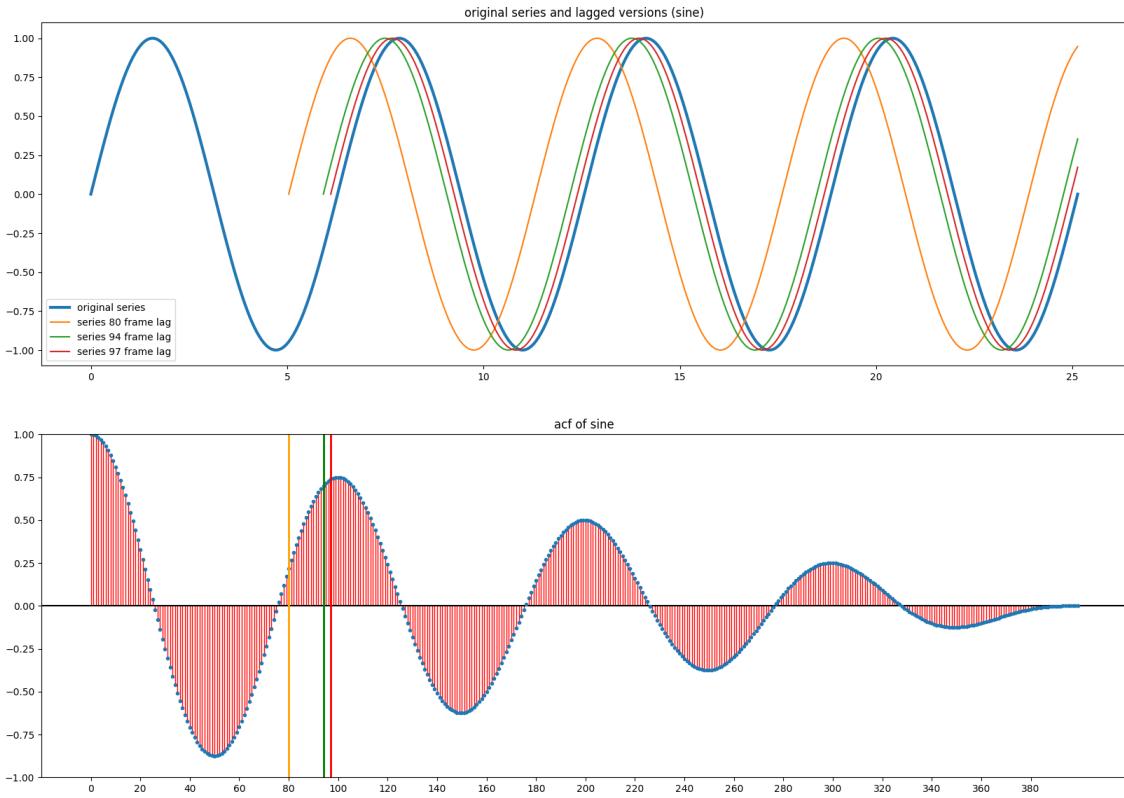
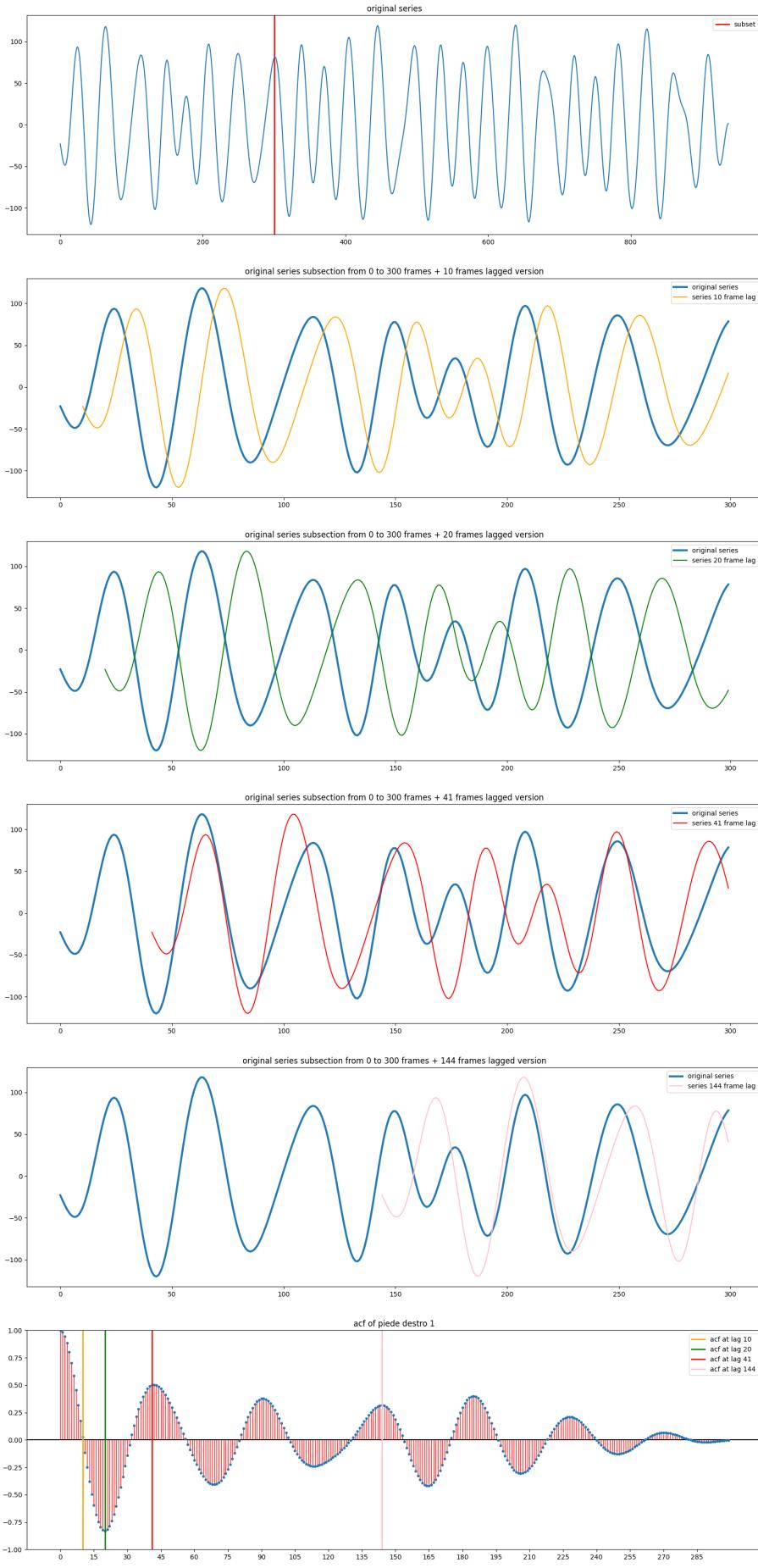


Figure 42: Grafico di un seno con sue versioni shiftate ed autocorrelazione.

Come possiamo vedere dai grafici in figura 41 e 42 possiamo pensiamo al passo di una persona come un seno oppre ad un coseno. L’idea di utilizzare la funzione di autocorrelazione nasce dal fatto che quando iniziamo a shiftare la serie ed iniziamo a sovrapporre i passi, cioè quando le sinusoidi inizieranno ad allinearsi, la correlazione aumenterà trovando così il pattern di camminata e la durata di uno stride.

L’effetto di tendenza a 0, all’aumentare del numero di frame, nel grafico dell’autocorrelazione, è dovuto alla funzione di shift, quest’ultima per poter “spostare” la serie aggiunge degli 0 a destra o a sinistra dipendentemente dal verso in cui si vuole eseguire lo shift.

Per dare un esempio più pratico prendiamo un sottoinsieme della serie relativa al piede destro posizione 1 del soggetto sano 8.



49

Figure 43: Autocorrelazione con esempi di segnale shiftato.

In figura 43 possiamo trovare diversi grafici relativi alla serie del paziente indicato sopra. Spieghiamo ora il loro significato:

1. **Grafico in 1° posizione:** rappresenta la serie originale con una linea rossa (frame 300) che indica il frame finale del sottoinsieme considerato dai grafici sottostanti.
2. **Grafico in 2° posizione:** in blu viene rappresentato il sottoinsieme preso in considerazione, cioè dal frame 0 al 300, ed in arancione lo stesso sottoinsieme ma shiftato di 10 frame/lag.
3. **Grafico in 3° posizione:** in blu viene rappresentato il sottoinsieme preso in considerazione, cioè dal frame 0 al 300, ed in verde lo stesso sottoinsieme ma shiftato di 20 frame/lag.
4. **Grafico in 4° posizione:** in blu viene rappresentato il sottoinsieme preso in considerazione, cioè dal frame 0 al 300, ed in rosso lo stesso sottoinsieme ma shiftato di 41 frame/lag.
5. **Grafico in 5° posizione:** in blu viene rappresentato il sottoinsieme preso in considerazione, cioè dal frame 0 al 300, ed in rosa lo stesso sottoinsieme ma shiftato di 144 frame/lag.
6. **Grafico in 6° posizione:** la funzione di autocorrelazione calcolata su tutti i lag possibili con linee di colore arancione, verde, rosso e rosa in prossimità dei valori relativi alla correlazione tra la serie originale e la serie shiftata di 10, 20, 41 e 144 lag (in sostanza sono i valori delle correlazioni relative ai grafici 2, 3, 4 e 5).

Possiamo notare che il primo valore della funzione di autocorrelazione, cioè il valore della correlazione tra la serie originale e se stessa, è uguale ad 1, di logica la massima correlazione possibile si può avere quando la serie è perfettamente allineata con se stessa.

La correlazione tra la serie originale e la sua versione shiftata di 10 lag invece ha valore quasi uguale a 0, questo perché se si guarda il grafico numero 2 in figura, la serie originale e la sua versione shiftata in colore arancione si “annullano” avendo così correlazione nulla. In altre parole quando la correlazione è vicina a 0 la serie con la sua versione shiftata sono “molto” diverse.

La correlazione tra la serie originale e la sua versione shiftata di 20 lag ha un’alta correlazione ma negativa, infatti se guardiamo il grafico numero 3 possiamo vedere come la serie originale in blu e la sua versione shiftata in verde sono una “l’opposto” dell’altra trovando così un’alta correlazione ma negativa. Per poter immaginare una correlazione negativa possiamo pensare alla correlazione della serie originale con una sua versione ribaltata sull’asse delle ascisse (in sostanza se y è la nostra serie consideriamo $-y$), il valore della correlazione a lag 0 varrà -1 .

Infine se consideriamo la correlazione tra la serie originale e la sua versione shiftata di 41 lag essa avrà un valore positivo ma ovviamente non 1, se guardiamo il grafico numero 4 possiamo notare come la serie originale in blu e la serie shiftata in rosso

tendano ad allinearsi, avendo così come risultato una correlazione positiva.

Capito ora come poter interpretare la funzione di autocorrelazione relativa ad una serie continuiamo con l'implementazione della soluzione proposta.

Utilizzando la funzione `argrelextrema` del modulo `statsmodels.tsa.stattools` all'output della funzione di autocorrelazione è stato possibile recuperare gli argomenti (indici della lista) dei massimi locali. Per esempio se prendiamo in considerazione il grafico 39 gli argomenti dei massimi locali sono

```
[45, 95, 152, 197, 245, 297, 347, 395, 454, 504, 549, 592, 636, 678]
```

dove ogni argomento fa riferimento all'istante di tempo t , quindi al frame, in cui uno stride si ripete.

Questo ovviamente succede in linea teorica, non sempre è detto che il grafico dell'autocorrelazione riesca ad ottenere con precisione quest'informazione, soprattutto se i dati non sono filtrati correttamente.

Continuando con l'implementazione del metodo, calcolando la distanza tra un argomento e l'altro troveremo la durata di quel determinato passo. Prendendo sempre in considerazione la lista di argomenti sopra e calcolandone la prima differenza otterremo

```
[50, 57, 45, 48, 52, 50, 48, 59, 50, 45, 43, 44, 42]
```

dove ogni elemento rappresenta la durata di un singolo stride.

Se calcoliamo la media otterremo la durata media di uno stride, che per la lista sopra sarà di 48.69frame cioè 1.623secondi.

Snippet (*Metodo per il calcolo dello stride*)

```
def stride(serie: list | np.ndarray):
    """ Calcola la durata di uno stride
        assumendo che la serie sia stazionaria
    """

    # redi la serie un array numpy
    if not isinstance(serie, np.ndarray):
        serie = np.array(serie)

    # funzione di autocorrelazione
    acf = funzione_autocorrelazione(serie)

    # prende gli argomenti massimi locali della acf
    arg_max_locali = argrelextrema(acf, np.greater)[0]

    # differenza tra gli argomenti massimi locali
    arg_max_locali_diff = compute_first_diff(arg_max_locali)
```

```

# calcola durata media di uno stride
media_stride = arg_max_locali_diff.mean()

if not isinstance(serie, np.ndarray):
    arg_max_locali_diff = arg_max_locali_diff.tolist()

return arg_max_locali_diff, media_stride

```

Questa serie di metodi è stata successivamente applicata ad ogni dataset ottenendo così la durata media di uno stride per ogni posizione del piede destro e sinistro. Per fare ciò è stato creato un “mini” programma che esegue automaticamente queste istruzioni ad ogni serie.

4.1.3 Analisi di complessità

Assumendo che la serie fornita sia già stazionaria la complessità di questo algoritmo deriva principalmente dalla complessità della funzione `funzione_autocorrelazione` e dalla complessità della funzione `singola_autocorrelazione` utilizzata dalla funzione di autocorrelazione.

Se consideriamo n come la lunghezza della serie, la funzione `autocorrelazione_singola` ha complessità temporale e spaziale $O(n)$ quindi la funzione `funzione_autocorrelazione` avrà complessità temporale $O(n^2)$ e complessità spaziale $O(n)$.

4.1.4 Analisi dei risultati e Conclusioni

Per semplicità, i risultati verranno analizzati considerando un passo completo, cioè la somma degli stride del piede destro e sinistro. Per fare chiarezza mostriamo un’immagine.

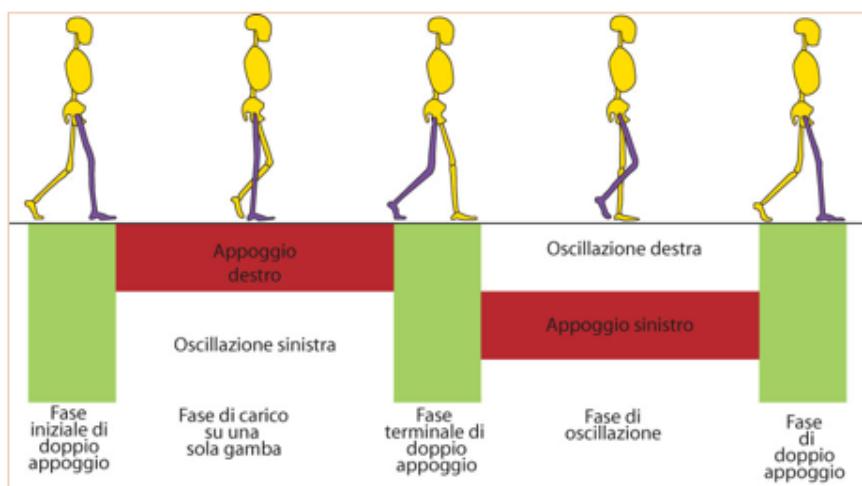


Figure 44: Camminata di un soggetto [7].

Considerando la figura 44 stiamo valutando il periodo (durata) che parte dalla prima striscia verde fino all'ultima striscia verde, quindi dall'inizio di una fase iniziale di doppio appoggio fino all'inizio di un'altra fase iniziale di doppio appoggio.

Detto ciò, compariamo ora la durata media dei passi completi dei soggetti sani e patologici.

Si è trovato che la durata media di un passo completo con camminata normale è di rispettivamente 68,84 **frame** (2,294 **secondi**) per i soggetti sani e di 81,71 **frame** (2,723 **secondi**) per i soggetti patologici. I soggetti patologici, con camminata normale, hanno quindi una durata di passo completo maggiore del 19,84% rispetto alla durata di un passo completo, con camminata normale, dei soggetti sani.

Applicando lo stesso ragionamento alle camminata tacco-punta e punta si è riscontrato che i soggetti patologici, con camminata tacco-punta, hanno una durata di passo completo maggiore del 8,28% rispetto alla durata di un passo completo, con camminata tacco-punta, dei soggetti sani, mentre nella camminata punta si è riscontrata una durata maggiore del 43,04%.

I risultati ottenuti analizzando i dati delle camminate tacco-punta e punta non possono ritenersi affidabili in quanto le percentuali e le medie sono state calcolate con troppe poche osservazioni.

Premessa sui risultati I risultati sopra ottenuti non sono da prendere come riferimento in quanto essi sono ottenuti mediante l'ausilio di un metodo non totalmente sicuro. Il metodo utilizzato è sperimentale e tantomeno fine allo scopo del tirocinio quindi potrebbe giungere a risultati non corretti poiché la funzione di autocorrelazione potrebbe rilevare pattern e correlazioni diverse da quelle aspettate ed anche perché questa tecnica è stata applicata a serie filtrate in un certo modo.

Problemi non totalmente risolti o ancora incogniti Una incognita rimasta aperta dopo l'analisi dei grafici dell'autocorrelazione e delle serie originali è l'effettiva importanza in questa soluzione di rendere le serie stazionarie. La funzione di test Dickey-Fuller per l'analisi della stazionarietà fornita dal pacchetto **statsmodels** ha la possibilità di capire automaticamente il numero di lag su cui eseguire il test e, per come è stato implementata la soluzione, l'algoritmo rileva un numero di lag non sufficiente per ottenere un risultato di test veritiero.

Considerato il fatto che la funzione della soluzione che rende le serie stazionarie non rileva propriamente quando una serie è stazionaria, e quindi tutte le analisi successive sono state eseguite su serie non propriamente stazionarie, i risultati ottenuti non sono discostano molto dalla realtà. Questo potrebbe essere dovuto a come sono stati filtrati i dati in partenza, e quindi, per questa determinata applicazione, non è stato necessario avere delle serie propriamente stazionarie.

In breve con lo sviluppo di questa soluzione e l'interpretazione dei risultati ottenuti non si è riuscita a cogliere l'importanza di avere una serie stazionaria non trovando molta differenza tra una serie stazionaria ed una non stazionaria (sempre rimanendo nei limiti di quest'applicazione).

4.2 Seconda Soluzione

In questa sezione verrà mostrata la seconda soluzione al problema posto, nello specifico verrà spiegata in breve l'idea generale, l'implementazione di alcune parti relative all'algoritmo, l'analisi dei risultati, l'analisi della complessità ed infine le conclusioni.

4.2.1 Idea generale

In questa soluzione al problema si è pensato di considerare la scoposizione delle serie nelle loro componenti quali trend, stagionalità e residui. Nello specifico l'idea è nata pensando che possiamo considerare ad un passo (stride) di un soggetto come ad un evento periodico che si ripete più o meno in ugual maniera nel tempo ad uno specifico periodo. Considerando quindi la scoposizione nelle tre componenti di una serie, la stagionalità ad un determinato periodo potrebbe rappresentare l'informazione che stiamo cercando di ottenere. Questo avviene solamente il linea teorica in quanto non è detto che il passo di un soggetto avvenga sempre ad intervalli precisi ma l'idea è che ad un certo periodo la stagionalità raccolga la maggior parte dell'informazione cercata.

Essendo che questo processo si può provare attuare solamente guardando il grafico della stagionalità si è pensato di automatizzarlo cercando un algoritmo che, data una serie temporale relativa ad un giunto, esso dia come risultato un periodo la cui stagionalità raccolga la maggior parte dell'informazione di un passo. Per fare ciò è stato necessario comparare due serie, o meglio detto, è stato necessario comparare la serie relativa ad un giunto divisa a metà considerando le due metà come serie separate. Successivamente si è calcolata la stagionalità su più periodi per entrambe le serie. In linea teorica i periodi con stagionalità simili sarebbero dovute essere i pattern che occorrono in entrambe le serie, mentre le stagionalità “diverse” sarebbero dovute essere pattern che non rientrano in entrambe probabilmente dovuto ad altre motivazioni quali, ad esempio, del rumore ancora presente.

4.2.2 Implementazione

Per prima cosa è necessario dividere una serie relativa ad un giunto di un soggetto in due e decidere su quali periodi calcolare la stagionalità.

Successivamente per ogni periodo si è calcolata la stagionalità delle due parti calcolandone la differenza quindi l'errore tra le due stagionalità. Per calcolare la stagionalità è stata utilizzata la funzione `seasonal_decompose` del modulo `statsmodels.tsa.seasonal` relativo al pacchetto `statsmodels`, mentre per il calcolo dell'errore le due stagionalità sono state convertite nel dominio delle frequenze, utilizzando la trasformata di Fourier, e calcolata la differenza di ampiezza tra frequenze. L'algoritmo è stato creato per poter accettare anche serie divise in più di due parti.

Vediamo ora come è stata implementata la funzione che esegue il calcolo dell'errore.

Snippet (*Calcolo dell'errore tra stagionalità*)

```

def freq_error(season_1: list, season_2: list):

    # prendi la lunghezza minima
    min_len = 0
    if len(season_1) < len(season_2):
        min_len = len(season_1)
    else:
        min_len = len(season_2)

    # calcolo della trasformata di
    # Fourier tra stagionalità
    fft_season_1 = sft.fft(season_1, min_len)
    fft_season_2 = sft.fft(season_2, min_len)

    # calcolo del magnitudo per ogni trasformata
    fft_season_1_m = np.abs(fft_season_1) / len(season_1)
    fft_season_2_m = np.abs(fft_season_2) / len(season_2)

    # differenza tra frequenze
    fft_diff = []
    for idx in range(min_len//2):
        fft_diff.append(fft_season_1_m[idx] - fft_season_2_m[idx])

    # calcolo dell'errore quadratico medio
    return np.power(fft_diff, 2).mean()

```

Vediamo ora il metodo principale.

Snippet (*Metodo per la soluzione al problema (2)*)

```

def soluzione_2(seriesList, periods):
    """
    seriesList =
        lista che contiene la serie divisa in due parti
        quindi due liste

    periods =
        lista di interi, è la lista che
        contiene i periodi su cui calcolare
        la stagionalità
    """

    # per ogni periodo
    errors = []
    for j, period in enumerate(periods):

        # non è possibile calcolare
        # la stagionalità per il periodo 0

```

```

if period == 0:
    raise Exception('stagionalità non calcolabile \
con periodo 0')

# calcola l'errore le parti della serie
aux_errors = []
for idx, series in enumerate(seriesList):

    # stagionalità della prima parte
    season = seasonal_decompose(
        series, period=period).seasonal

    for idx in range(idx+1, len(seriesList)):

        # stagionalità della seconda parte
        # (e altre parti)
        aux_season = seasonal_decompose(
            seriesList[idx], period=period).seasonal

        # calcolo dell'errore tra stagionalità
        aux_errors.append(
            freq_error(season, aux_season)
        )

errors.append(np.mean(aux_errors))

return np.min(errors), periods[np.argmin(errors)], errors

```

Come possiamo vedere dall’implementazione come risultato otteniamo l’errore minimo, il periodo associato all’errore minimo e la lista contenente, per ogni periodo, l’errore associato.

In una fase successiva è stata creata una funzione che permette di analizzare la lista relativa agli errori per ogni periodo creando dei grafici che indicano i periodi “salienti”. L’implementazione di quet’ultima non verrà data ma il concetto è che per periodi piccoli l’errore è sempre il minimo e crescente e, dopo un certo periodo, l’errore inizia ad aumentare drasticamente per poi diminuire successivamente dopo aver trovato una stagionalità nuovamente simile tra le due serie. Per capire meglio consideriamo la lista degli errori tra 1 e 30, in questo range l’errore è crescente per ogni periodo, dal periodo 31 in poi l’errore aumenta drasticamente fino al periodo 80 per poi diminuire nuovamente ed ad un certo punto riaumentare. Questo è dovuto appunto al fatto che quando le stagionalità non si “assomigliano” l’errore sarà alto mentre per stagionalità simili l’errore è minimo o comunque molto basso.

4.2.3 Analisi di complessità

Consideriamo come m il numero di sottoinsiemi, di ugual lunghezza, creati a partire da una serie (negli esempi sopra abbiamo sempre considerato di dividere la serie in due parti), n la lunghezza di un sottoinsieme, S la complessità della funzione `seasonal_decompose` e tenuto conto che il periodo massimo su cui è possibile calcolare quest'ultima è di $(m/2)$, la complessità temporale della seconda soluzione è $O((n/2)m \log(m)Sn \log(n))$ mentre la complessità spaziale sarà $O(n + 2n + (n/2))$ quindi $O(n)$.

4.2.4 Analisi dei risultati e Conclusioni

Analizziamo ora qualche grafico cercando di capire meglio come rappresentare i risultati del metodo spiegato nella sezione precedente.

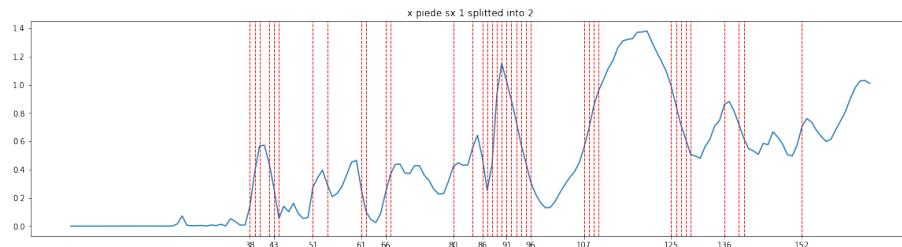


Figure 45: Grafico degli errori per ogni periodo relativo al piede sinitro posizione 1 del soggetto patologico 6.

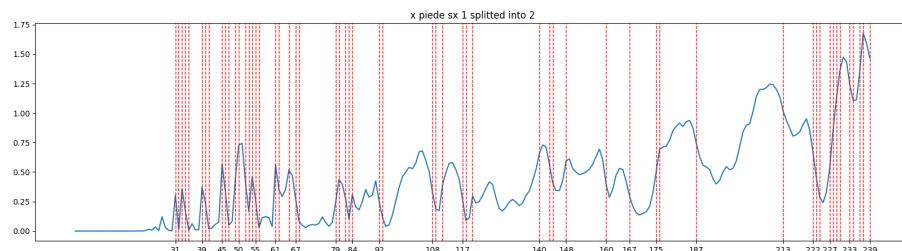


Figure 46: Grafico degli errori per ogni periodo relativo al piede sinitro posizione 1 del soggetto sano 8.

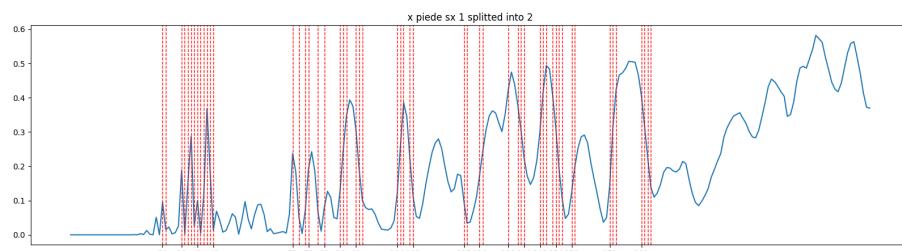


Figure 47: Grafico degli errori per ogni periodo relativo al piede sinitro posizione 1 del soggetto patologico 21.

Nei grafici in figura 45, 46 e 47 sono rappresentati gli errori sull'asse delle ordinate ed i periodi, a cui ogni errore è associato, sull'asse delle ascisse. Le linee rosse verticali rappresentano i punti salienti, calcolate mediante l'asilio di una funzione che utilizza la media e deviazione standard della differenza tra un errore e quello successivo.

Controllando diversi grafici si è potuto notato che, per molte serie, in prossimità del periodo relativo ad un passo (stride) l'errore inizia ad alzarsi, in questo modo dovrebbe essere possibile calcolare la durata di un passo. Questo però non succede sempre, con alcune serie questo fenomeno non si manifesta o comunque si manifesta ma non in corrispondenza del periodo legato ad un passo, rendendo quindi impossibile reperire l'informazione necessaria.

Questo potrebbe essere causato dal fatto che la funzione di seasonal decompose “aggiusta” la componente di residui per riuscire ad ottenere una periodicità nella componente di stagionalità.

Tenendo a mente le considerazioni fatte fino questo punto possiamo quindi considerare i risultati ottenuti, e di conseguenza questa soluzione, come non validi, essendo che non si ha la sicurezza di poter ottenere risultati coerenti per la ricerca dell'informazione interessata.

4.3 Conclusioni Finali

Prendendo in considerazione la prima soluzione, spiegata nella sezione precedente, possiamo concludere che la funzione di autocorrelazione può essere utilizzata per la soluzione a problemi di questa tipologia, più in particolare essa può essere utilizzata per trovare pattern e/o caratteristiche presenti nei dati, che dal grafico potrebbero non essere evidenti.

La seconda soluzione proposta invece non è stata utile allo scopo di trovare un singolo passo relativo ad un soggetto. Normalmente consideriamo azioni abituali, come ad esempio il passo, come pattern, di un determinato periodo, che si ripetono nel tempo quando invece nella realtà non è così. Tuttavia, l'algoritmo presentato nella seconda soluzione, potrebbe essere utilizzato in altre applicazioni la cui necessità ricade sul trovare periodi in cui occorrono stagionalità simili appartenenti a serie temporali diverse, oppure esso potrebbe essere preso come spunto implementativo per la creazione di un ulteriore algoritmo.

In conclusione, possiamo constatare che le tecniche utilizzate per l'analisi di serie temporali possono essere d'aiuto per una prima analisi non accurata del problema, non sostituendo però le tecniche tuttora utilizzate per analisi a problemi di questa tipologia.

5 Bibliografia

Bibliografia

- [1] medium - by Marv. *What's The Difference Between Autocorrelation and Partial Autocorrelation For Time Series Analysis?* [Online; controllata il 8-marzo-2021]. 2021. URL: <https://emel333.medium.com/interpreting-autocorrelation-partial-autocorrelation-plots-for-time-series-analysis-23f87b102c64>.
- [2] analyticsindiamag - BY VIJAYSINH LENDAVE. *What are autocorrelation and partial autocorrelation in time series data?* [Online; controllata il 6-febbraio-2022]. 2022. URL: <https://analyticsindiamag.com/what-are-autocorrelation-and-partial-autocorrelation-in-time-series-data/>.
- [3] medium - towardsdatascience - by Egor Howell. *Seasonality of Time Series.* [Online; controllata il 26-ottobre-2022]. 2022. URL: <https://towardsdatascience.com/seasonality-of-time-series-5b45b4809acd>.
- [4] medium - towardsdatascience - by Spencer Hayes. *Finding Seasonal Trends in Time-Series Data with Python.* [Online; controllata il 8-giugno-2021]. 2021. URL: <https://towardsdatascience.com/finding-seasonal-trends-in-time-series-data-with-python-ce10c37aa861>.
- [5] Sean Abu. *Seasonal ARIMA with Python.* [Online; controllata il 22-marzo-2016]. 2016. URL: <http://www.seanabu.com/2016/03/22/time-series-seasonal-ARIMA-model-in-python/>.
- [6] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository.* 2017. URL: <http://archive.ics.uci.edu/ml>.
- [7] Dott. Cristian Francavilla. *Analisi del cammino.* [Online; controllata il 19-gennaio]. URL: <https://www.cristianfrancavilla.it/analisi-del-cammino/>.
- [8] geeksforgeeks. *What is a trend in time series?* [Online; controllata il 30-maggio-2022]. 2022. URL: <https://www.geeksforgeeks.org/what-is-a-trend-in-time-series/>.
- [9] Zhang S. Guo B. Dong A. He J. Xu Z. Chen S.X. *Cautionary Tales on Air-Quality Improvement in Beijing.* 2017. URL: <https://archive.ics.uci.edu/ml/datasets/Beijing+Multi-Site+Air-Quality+Data>.
- [10] Dmitriy Sergeyev. *Topic 9. Part 1. Time series analysis in Python.* [Online]. 2022. URL: https://mlcourse.ai/book/topic09/topic9_part1_time_series_python.html#double-exponential-smoothing.
- [11] Simplilearn. *An Introduction to Exponential Smoothing for Time Series Forecasting in Python.* [Online; controllata il 30-settembre-2022]. 2022. URL: <https://www.simplilearn.com/exponential-smoothing-for-time-series-forecasting-in-python-article>.
- [12] StatLect. *Covariance stationary.* [Online]. URL: <https://www.statlect.com/glossary/covariance-stationary#:~:text=A%20sequence%20of%20random%20variables,not%20on%20their%20absolute%20position..>
- [13] Wikipedia. *Augmented Dickey–Fuller test — Wikipedia, L'encyclopédia libera.* [Online; controllata il 30-novembre-2022]. 2022. URL: https://en.wikipedia.org/wiki/Augmented_Dickey%20Fuller_test.

- [14] Wikipedia. *Autocorrelation* — Wikipedia, L'enciclopedia libera. [Online; controllata il 3-gennaio-2023]. 2023. URL: <https://en.wikipedia.org/wiki/Autocorrelation>.
- [15] Wikipedia. *Autocorrelazione* — Wikipedia, L'enciclopedia libera. [Online; controllata il 17-gennaio-2023]. 2023. URL: <https://it.wikipedia.org/wiki/Autocorrelazione>.
- [16] Wikipedia. *Dickey–Fuller test* — Wikipedia, L'enciclopedia libera. [Online; controllata il 30-marzo-2022]. 2022. URL: https://en.wikipedia.org/wiki/Dickey%E2%80%93Fuller_test.
- [17] Wikipedia. *Exponential smoothing* — Wikipedia, L'enciclopedia libera. [Online; controllata il 4-febbraio-2023]. 2023. URL: https://en.wikipedia.org/wiki/Exponential_smoothing.
- [18] Wikipedia. *Moving average* — Wikipedia, L'enciclopedia libera. [Online; controllata il 16-dicembre-2022]. 2022. URL: https://en.wikipedia.org/wiki/Moving_average.
- [19] Wikipedia. *Partial autocorrelation function* — Wikipedia, L'enciclopedia libera. [Online; controllata il 31-gennaio-2023]. 2023. URL: https://en.wikipedia.org/wiki/Partial_autocorrelation_function.
- [20] Wikipedia. *Serie Storica* — Wikipedia, L'enciclopedia libera. [Online; controllata il 30-gennaio-2022]. 2022. URL: https://it.wikipedia.org/wiki/Serie_storica.
- [21] Wikipedia. *Unit root* — Wikipedia, L'enciclopedia libera. [Online; controllata il 14-giugno-2022]. 2022. URL: https://en.wikipedia.org/wiki/Unit_root.