SCHOOL OF ELECTRICAL & COMPUTER ENGINEERING

TECHNICAL UNIVERSITY OF CRETE

DIPLOMA THESIS

TIME-SERIES ANALYSIS USING MACHINE LEARNING METHODS

AUTHOR: NIKOLAOS PARASKAKIS

THESIS COMMITTEE: PROF. DIONISSIOS HRISTOPULOS
PROF. ATHANASIOS LIAVAS
PROF. MICHAEL ZERVAKIS

SEPTEMBER, 2023

A thesis submitted in fulfillment of the requirements
for the diploma of Electrical and Computer Engineer

Abstract

This diploma thesis explores the application of machine learning techniques to time-series analysis, focusing on the yearly number of sunspots dataset. The introduction begins with a presentation of fundamental concepts in time-series analysis, encompassing stochastic processes, correlation, stationarity, heteroscedasticity, and time-series decomposition methods. The thesis then delves into crucial aspects of time-series forecasting including dataset splitting, cross-validation, evaluation metrics, and various forecasting strategies, emphasizing both one-step and multi-step forecasting.

A key focus of this research is the examination of non-linear data transformations and their role in enhancing model predictive performance by achieving desirable properties of the transformed dataset, such as normality and stationarity. The study also investigates advanced machine learning methods, such as Gaussian Processes (GPs), Gradient Boosting Decision Trees (GBDT), and Long Short-Term Memory (LSTM) neural networks in the context of time-series forecasting.

This thesis contains a case study which involves the analysis and forecasting of the yearly number of sunspots. First, we take advantage of GPs, which constitute a probabilistic non-parametric regression framework. We use a constant mean function and an exponential multiplied by a periodic covariance kernel, while assuming i.i.d. Gaussian noise, and Gaussian likelihood of the data. To square with these assumptions, we apply the κ-logarithmic transformation (Kaniadakis G., 2009), that accounts for the skewness, heteroscedasticity, and non-negativity of the sunspot data. Then, we train the model on the transformed data. We optimized the model's hyperparameters using maximum likelihood estimation (MLE). Next, we utilize the algorithm of LightGBM (Light Gradient Boosting Machine), which is a gradient-boosting framework of regression trees, that is well-known for its efficiency and accuracy in regression tasks. The tuning of hyperparameters is carried out using Bayesian optimization with the goal to minimize the validation loss. Finally, we use an especial form of recurrent neural network (RNN), the LSTM, which comprise a deep learning architecture, capable of capturing long-term dependencies and complex patterns. It consists of cells, each of which is connected to three gates (input, forget, and output) responsible for information flow. We implemented an LSTM model with multiple layers capable of forecasting the yearly number of sunspots, and optimized its hyperparameters using grid search with the objective of minimizing the validation loss. All in all, this real-world example illustrates the effectiveness of the discussed machine learning techniques in modeling time-series data and producing competitive predictions.

A comparative analysis which examines the strengths and weaknesses of each of these methods is presented. GP regression excels in interpretability, delivers uncertainty estimates along with point estimates, and can capture complex patterns using different kernels, but it requires the computationally intensive inversion of large covariance matrices (large dataset). LSTM performs well in capturing long-term dependencies, but it needs large amounts of data, time, and resources for tuning and training, and it suffers from error accumulation on long-term predictions. LightGBM can capture complex patterns as well, and it is more computationally efficient, making its training faster.

All in all, this thesis provides insights into the performance and characteristics of three powerful machine learning methods for sunspot number prediction. Our findings collectively mark a significant stride in the application of advanced machine learning techniques to forecast and analyze time-series data across diverse disciplines.

Acknowledgements

I extend my heartfelt gratitude to all who have played a pivotal role in the successful culmination of this thesis. First and foremost, I am deeply grateful to my thesis supervisor, Dionissios Hristopulos, whose unwavering support, expert guidance, and insightful feedback have been instrumental throughout this research endeavor. I would also like to thank my thesis committee members, Athanasios Liavas and Michael Zervakis, for their valuable input and scholarly insights, and the Technical University of Crete as a whole, for being a conducive academic environment providing me with all the necessary knowledge and supplies that an engineer needs nowadays. My family's encouragement and understanding during the challenges and triumphs of this academic journey have been a constant source of motivation, and I express my profound thanks to them. Last but not least, I appreciate the support and camaraderie of my friends and fellow students, especially Rania and Evi, who have made this journey more enjoyable, provided daily encouragement, and together we have created priceless memories. This thesis would not have been possible without the collective support, mentorship, and contributions of all those mentioned above.

Nikolaos Paraskakis
September, 2023

Contents

List of Acronyms

R^2 R-Squared

ACF Auto-Correlation Function

AdaBoost Adaptive Boosting

ADF Augmented Dickey-Fuller

AIC Akaike Information Criterion

ANN Artificial Neural Network

AR Auto-Regressive

ARD Automatic Relevance Determination

ARIMA Auto-Regressive Integrated Moving Average

ARMA Auto-Regressive Moving Average

BIC Bayesian Information Criterion

BPTT Backpropagation Through Time

CART Classification And Regression Tree

CDF Cumulative Distribution function

CEC Constant Error Carousel

CH HSS Coronal Hole High Speed Streams

CIR Co-rotating Interaction Region

CMA Cumulative Moving Average

CME Coronal Mass Ejection

CNN Convolutional Neural Network

CPU Central Processing Unit

DBN Deep Belief Network

DF Dickey-Fuller

DF-GLS Dickey-Fuller Generalized Least Squares

DFT Discrete Fourier Transform

DNN Deep Neural Network

DSCOVR Deep Space Climate Observatory

ECDF Empirical Cumulative Distribution Function

EFB Exclusive Feature Bundling

EI Expected Improvement

EMA Exponential Moving Average

ESA European Space Agency

FFNN Feed-Forward Neural Network

FFT Fast Fourier Transform

FRNN Fully Recurrent Neural Network

G-causality Granger-Causality

GBDT Gradient Boosting Decision Tree

GLS Generalized Least Squares

GOSS Gradient-based One Side Sampling

GP Gaussian Process

GPR Gaussian Process Regression

GPU Graphics Processing Unit

GSFC Goddard Space Flight Center

HSO Heliophysics System Observatory

i.i.d. independent and identically distributed

IMF Interplanetary Magnetic Field

KPSS Kwiatkowski–Phillips–Schmidt–Shin

LASCO Large Angle and Spectrometric Coronagraph

LCB Lower Confidence Bound

LightGBM Light Gradient Boosting Machine

LSTM Long Short-Term Memory

MA Moving Average

MAE Mean Absolute Error

MAP Maximum A Posteriori

MK Mann-Kendall

MLE Maximum Likelihood Estimation

MSE Mean Squared Error

MVN	Multi-Variate Normal
NASA	National Aeronautics and Space Administration
NOAA	National Oceanic and Atmospheric Administration
NP-Hard	Non-deterministic Polynomial-time Hard
NWS	National Weather Service
OLS	Ordinary Least Squares
PACF	Partial Auto-Correlation Function
PCC	Pearson Correlation Coefficient
PDF	Probability Density Function
PFSS	Potential Field Source Surface
PI	Probability Improvement
PMF	Probability Mass Function
PP	Phillips-Perron
RAM	Random Access Memory
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RRMSE	Relative Root Mean Square Error
RTRL	Real-Time Recurrent Learning
SA	Steps Ahead
SDO	Solar Dynamics Observatory
SIDC	Solar Influences Data Analysis Center
SILSO	Sunspot Index and Long-term Solar Observations
SMA	Simple Moving Average
SMBO	Sequential Model Based Optimization
SOHO	Solar and Heliospheric Observatory
SSD	Solid State Drive
STEREO	Solar Terrestrial Relations Observatory
STL	Seasonal and Trend Decomposition
SWPC	Space Weather Prediction Center

TPE	Tree Parzen Estimators
UCAR	University Corporation for Atmospheric Research
UVN	Uni-Variate Normal
WDC	World Data Center
WMA	Weighted Moving Average
XGBoost	Extreme Gradient Boosting

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 The growth of time-series data in the contemporary era

In today's rapidly evolving world, data has become the lifeblood of decision-making and innovation across various domains. Among the myriad forms of data that have emerged as invaluable assets, time-series data stands out as a particularly crucial component in understanding phenomena and navigating the complex landscape of contemporary challenges. Time-series data, as the name suggests, refers to a sequence of data points collected or recorded at successive intervals over time. These data sets are omnipresent, ranging from financial market prices, weather measurements, and sensor data from industrial machinery to healthcare records, social media interactions, and beyond. What makes time-series data distinctive is its inherent temporal dimension, allowing us to analyze how variables change over time, detect patterns, and uncover hidden insights. As a result, time-series data has evolved from being merely a tool for retrospective analysis to a critical resource for predictive and prescriptive analytics, aiding in forecasting, anomaly detection, and decision support across numerous domains.

In parallel with the rise of time-series data, the advent of big data has revolutionized the way we collect, store, process, and utilize information. Big data is characterized not only by its sheer volume but also by its velocity, variety, and veracity. Massive datasets, generated from diverse sources, have become available at an unprecedented scale. Technologies like cloud computing, distributed storage systems, advanced analytics tools, machine learning, and artificial intelligence have empowered scientists to harness the potential of big data to uncover valuable insights, gain competitive advantages, and drive innovation.

The fusion of time-series data with big data has introduced new dimensions to our understanding of complex phenomena. By integrating historical trends with real-time information, scientists can make more informed decisions and adapt to changing circumstances with greater agility. For instance, financial institutions use time-series data to monitor market fluctuations and predict trading opportunities, while healthcare providers leverage it to enhance patient care through predictive diagnostics. In the realm of smart cities, time-series data helps optimize traffic management and energy consumption, improving the quality of urban life. Moreover, researchers and scientists rely on time-series data analysis to explore climate patterns, track disease outbreaks, and even study the behavior of subatomic particles.

The importance of time-series data and big data in today's world cannot be overstated. It is very important to develop new methods, utilizing state-of-the-art machine learning algorithms, capable of capturing relationships between data, modeling their behavior, and producing accurate forecasts of them. To demonstrate the application of these methodologies, this thesis includes a case study on the yearly mean total sunspot number time-series. Solar activity has various implications on satellite communications, navigation systems, power distribution systems, and climate change, while it has direct relation to space weather forecasting. Sunspots, the fundamental indicators of solar activity, are cooler dark areas that appear on the surface of the Sun, caused by a concentration of magnetic field lines. The solar, or sunspot, cycle is a quasi-periodic change in the Sun's activity, measured in terms of variations in the number of observed sunspots on the Sun's surface. The dominant 11-year period is presumably induced by the electromagnetic solar dynamo mechanism, and based on that, we can build physics-informed models.

How are the aforementioned machine learning algorithms built and how can they be used to model time-series data? How accurately can we predict the occurrence and behavior of sunspots? In our data-driven era, can advanced machine learning algorithms provide a novel and effective means to enhance our understanding and prediction of sunspot activity? In this thesis, we aim to address these questions and enhance our understanding of all the underlying procedures. In Section 2, we present fundamental principles of time-series analysis, including stochastic processes, stationarity, heteroscedasticity, and time-series decomposition methods. In Section 3, essential concepts under machine learning and time-series forecasting are introduced, while in Section 4 we highlight the significance of applying non-linear transformations to the data in order to achieve desired properties and enhance the performance of the underlying models. Next, in Sections 5, 6, and 7, the algorithms of "Gaussian Process Regression", "Gradient Boosting Decision Trees", and "Long Short-Term Memory" neural networks are presented. In Section 9, we get an idea of Heliophysics and solar activity, and in Section 10, we make an exploratory analysis of the time-series of both, yearly and monthly, sunspot numbers. Finally, in Section 11, we focus on the forecasting of the yearly mean total sunspot number using the presented machine learning methods.

1.2 Motivation for studying sunspots

Next, we will focus on the various aspects of solar activity in order to clarify the motivation behind studying them. The effects of sunspots become even more pronounced during periods of higher activity. Increased sunspot activity correlates with a higher frequency of solar flares and coronal mass ejections, which in turn intensify geomagnetic storm activity on Earth. These storms affect satellites in space, causing loss of data or operation. Furthermore, during sunspot maximums, we witness a surge in the Northern and Southern Lights, and increased risk of disruptions in radio transmissions and power grids. Below, we will refer to another aspect of interest in solar activity that has direct relation with Earth's climate and environment.

Beyond the aforementioned significant implications of sunspots, it is worth referring to a recently published paper by Liang *et al.* [8], which discusses the relationship between solar activity—specifically sunspot number—and the El Niño Modoki events. The El Niño Southern Oscillation (ENSO) is a climate phenomenon characterized by the warming of the sea surface temperatures in the central and eastern equatorial Pacific Ocean. It occurs irregularly every 2 to 7 years and can have significant impacts on weather patterns around the world. During El Niño, there is a weakening of the trade winds, which leads to a reduction in upwelling of cold water and a decrease in the strength of the eastern Pacific Ocean's cold tongue. This results in changes in atmospheric circulation patterns and can lead to droughts, floods, and other extreme weather events in different regions. El Niño Modoki is a specific type of El Niño that is characterized by a different pattern of sea surface temperature anomalies in the equatorial Pacific. Unlike the traditional El Niño, which is characterized by warm anomalies in the eastern Pacific, El Niño Modoki is characterized by warm anomalies in the central Pacific and cooler anomalies in the eastern and western Pacific. This different pattern of sea surface temperature anomalies leads to distinct climate impacts compared to the traditional El Niño. For example, during El Niño Modoki, the western coast of the United States may experience drought conditions, while during traditional El Niño events, it is typically wetter.

Liang *et al.* suggest that information flowing from solar activity 45 years ago can influence sea surface temperatures and result in a causal structure resembling the El Niño Modoki mode. This information flow, which represents the transfer of predictability, is computed using a multidimensional system constructed from sunspot number series with time delays of 22-50 years. The paper highlights that the predictability of El Niño Modoki events can be achieved by using the sunspot numbers as predictors. Specifically, the first 25 principal components of the sunspot number series are taken as predictors, and through causal AI based on the information flow, the events can be accurately reproduced up to 12 years in advance. This research provides valuable insights into the potential predictability of climate phenomena, such as El Niño Modoki, by studying sunspot time-series.

1.3 Peak solar activity is arriving sooner than expected

Several articles are written continuously regarding the solar activity and the prediction of the current cycle's peak. Sarah Scoles [9] states in a recent *Science* article that "scientists tackle a burning question; when will our quiet sun turn violent?". In her article, she makes clear that scientists are working to predict when the sun will reach its peak of violent bursts of magnetic activity, known as solar maximum, and the potential impact on technology. Emphasis is given on the potential consequences of a major solar storm, similar to the Carrington Event of 1859, which caused disruptions in telegraph lines and could have dire consequences for modern infrastructure. The publication further asserts the efforts of a panel of scientists sponsored by NASA and NOAA to analyze various models and come to a consensus about the next solar cycle. There is a debate among scientists regarding the best approach to predict solar activity. Some scientists question the use of sunspots as a proxy for predicting the Sun's behavior, considering them as symptoms rather than causes of solar activity. They argue that sunspots are just one aspect of a larger, still mysterious story playing out inside the sun. Given that, the article discusses different models and approaches being used to predict solar activity, including physics-based simulations and statistical correlations. It acknowledges that some models lack a strong connection to solar physics and rely on correlations found through statistical analysis. However, it also mentions the work of scientist Scott McIntosh, who proposes an alternative theory suggesting that "bright spots" in the Sun's outer atmosphere may be better markers for predicting solar activity.

Also reporting in *Science*, Zack Savitsky [10] points out that "the peak of solar activity is arriving sooner than expected and the Sun's flare-ups can threaten satellites and electric grids, highlighting the need for better forecasts". His article highlights the potential implications of the earlier and more intense peak solar activity. It mentions that the Sun's upcoming cycle is expected to be stronger than the previous one, which was relatively mild. The increased solar activity can lead to particle storms that pose risks to various technological systems, including satellites, radio transmissions, and power grids. The publication refers to the process of predicting solar activity. Scientists typically track solar cycles by counting sunspots, which are flares of activity caused by magnetic field loops. The prediction panel analyzed around 60 different forecast models in 2019, ranging from statistical models to advanced computer models that simulate the Sun's dynamo and magnetic fields. The panel's consensus was that the monthly sunspot count would peak at around 115 in July 2025, making it a relatively weak cycle. However, the Sun has already shown more activity than expected, with 159 sunspots in July and 115 in August. The discrepancy between the panel's prediction and the actual solar activity raises the need for better observations of the Sun and a deeper understanding of the factors influencing its magnetic field. The article also mentions the importance of the polar magnetic field in predictions and the limitations of current observations from the Wilcox Solar Observatory. Equally important, the source indicates ongoing research, such as the observations of "bright points" by Scott McIntosh and his colleagues, which suggest the interaction of magnetic field bands and their potential influence on solar activity. Savitsky concludes by emphasizing the need for continuous research and the readiness to revise predictions in the field of solar activity forecasting. He acknowledges the progress made in understanding the Sun's dynamo, but also highlights the work that still needs to be done to improve predictions and mitigate the potential impacts of increased solar activity on Earth.

In light of the above introduction, the study of the time series of sunspots by means of machine learning methods provides an exciting combination of new methodologies coupled with a very interesting physical application.

2 Fundamentals in Time-Series Analysis

2.1 Stochastic processes and time-series

A stochastic process, often referred to as a random process, constitutes a fundamental mathematical concept applied within the realms of probability theory and related fields. It can be conceptualized as a sequence of random variables, wherein the index of the sequence typically represents the concept of time [11]. The utility of stochastic processes extends across a wide spectrum of disciplines, including biology, chemistry, ecology, neuroscience, physics, image processing, signal processing, control theory, information theory, computer science, and telecommunications. In the domain of finance, stochastic processes have found extensive use, primarily driven by the need to capture and model the seemingly random movements observed in financial markets.

2.1.1 Probability space

A probability space is a triple $(\Omega, \mathcal{A}, P)$, where [12]:

- Ω is a non-empty set, which is called the sample space.
- $\mathcal{A}$ is a σ-algebra of subsets of Ω, i.e., a family of subsets closed with respect to countable union and complement with respect to Ω.
- P is a probability measure defined for all members of $\mathcal{A}$. That is a function $P : \mathcal{A} \to [0, 1]$ such that $P(A) \geq 0$ for all $A \in \mathcal{A}$, $P(\Omega) = 1$, $P(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$, for all sequences $A_i \in \mathcal{A}$ such that $A_k \cap A_j = \emptyset$ for $k \neq j$.

2.1.2 Random variables

A real-valued random variable X is a function that maps from a sample space Ω, which contains all possible outcomes of a random experiment, to the real numbers $\mathbb{R}$. Formally, it is

$$X : \Omega \to \mathbb{R}$$

Random variables can be classified into two main categories:

- **Discrete random variables**: These can take on a countable number of distinct values. An example is the number of heads in three coin tosses.
- **Continuous random variables**: These can take on an infinite number of possible values over a continuum. An example is the amount of time a bus takes to arrive.

2.1.3 PDF, PMF, and CDF

The probability density function (PDF) describes the likelihood of a continuous random variable X taking on a specific value x, and it is denoted as $f_X(x)$. Note that the value of the PDF at a specific point indicates the probability density, not the probability itself. The probability mass function (PMF) describes the probability of a discrete random variable X taking on a specific value x from the set R_X, and it is denoted as $P_X(X = x)$ or just $P_X(x)$. The cumulative distribution function (CDF) describes the probability that a random variable (continuous or discrete) will take on a value less than or equal to a certain value x, $P_X(X \leq x)$, and it is denoted as $F_X(x)$. It is

$$\begin{aligned} P(X \leq x) &= F_X(x) \\ &= \begin{cases} \int_{-\infty}^{x} f_X(u)du & \text{, if } X \text{ is continuous} \\ \sum_{x_i \leq x} P_X(x_i) & \text{, if } X \text{ is discrete with range } R_X = \{x_i, i \in I\} \end{cases} \end{aligned} \tag{2.1}$$

$$\begin{aligned} P(a \leq X \leq b) &= F_X(b) - F_X(a) \\ &= \begin{cases} \int_{a}^{b} f_X(u)du & \text{, if } X \text{ is continuous} \\ \sum_{a \leq x_i \leq b} P_X(x_i) & \text{, if } X \text{ is discrete with range } R_X = \{x_i, i \in I\} \end{cases} \end{aligned} \tag{2.2}$$

$$f_X(x) = \frac{d}{dx} F_X(x), \text{ if } X \text{ is continuous} \tag{2.3}$$

The integral of the PDF across the entire range of the random variable is equal to 1. Likewise, the summation of the PMF over all feasible values of the random variable also equals 1. As the random variable ranges from negative infinity to positive infinity, the CDF attains lower and upper limits of 0 and 1 respectively.

2.1.4 Expected value

In informal terms, the expectation of a discrete random variable, characterized by a countable set of potential outcomes, is defined in a manner analogous to a weighted average of all conceivable outcomes. In this context, the weights are determined by the probabilities associated with each specific outcome. This is to say that

$$E[X] = \sum_{i \in I} x_i p_i \tag{2.4}$$

where $R_X = \{x_i, i \in I\}$ are the possible outcomes of the random variable X and $P_X = \{p_i = P(X = x_i), i \in I\}$ are their corresponding probabilities.

Now consider a continuous random variable X which has a probability density function given by a function f_X on the real number line. This means that the probability of X taking on a value in any given open interval is given by the integral of f over that interval. The expectation of X is then given by the integral

$$E[X] = \int_{-\infty}^{\infty} x f_X(x)dx \tag{2.5}$$

2.1.5 Variance and standard deviation

Variance is the squared deviation from the mean of a random variable X. It is represented by $\operatorname{Var}(X)$ or σ_X^2 as it is often defined as the square of the standard deviation σ_X. Variance is a measure of dispersion, meaning it is a measure of how far a set of numbers is spread out from their average value. The variance of a random variable X is the expected value of the squared deviation from the mean of X, which is $\mu = E[X]$. Formally, it is

$$\operatorname{Var}(X) = E[(X-\mu)^2] = E[X^2] - (E[X])^2 \tag{2.6}$$

More specific, for each case of random variable X, we have

$$\operatorname{Var}(X) = \begin{cases} \int_{-\infty}^{\infty} (x-\mu)^2 f_X(x)dx & \text{, if } X \text{ is continuous} \\ \sum_{i\in I} (x_i-\mu)^2 p_i & \text{, if } X \text{ is discrete} \end{cases} \tag{2.7}$$

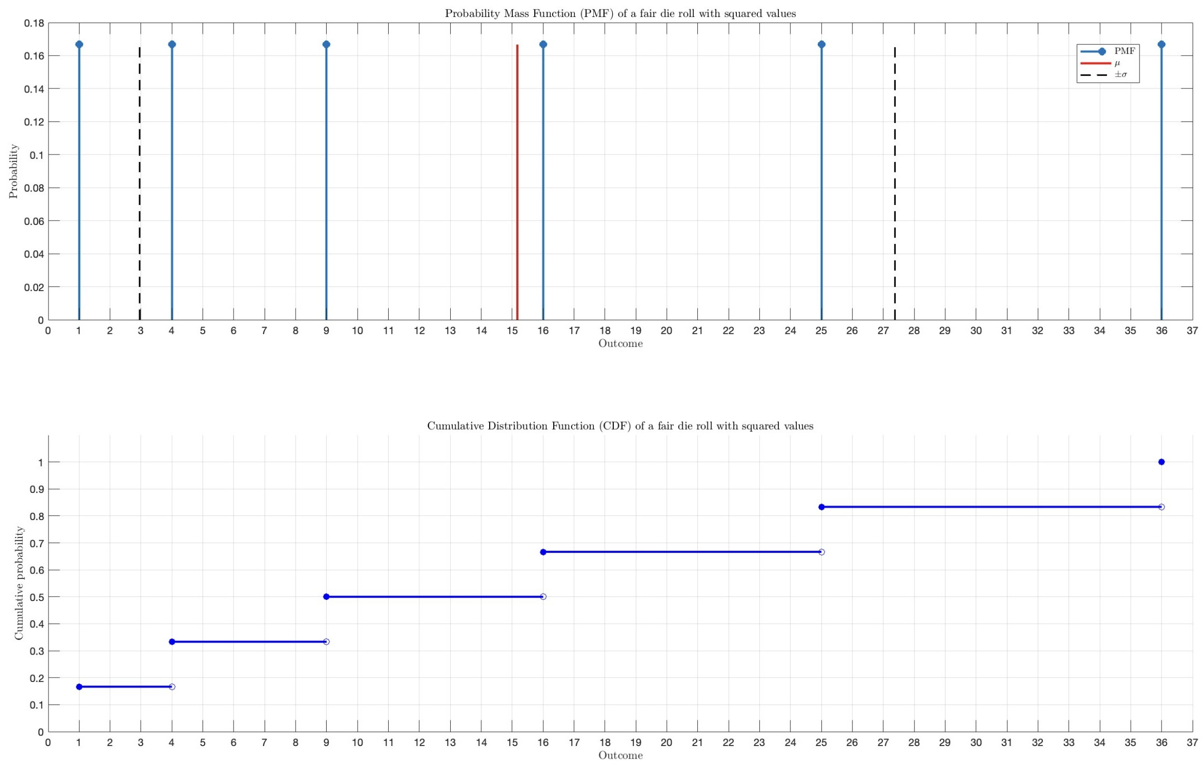


Figure 2.1: Probability Mass Function (PMF) and Cumulative Distribution Function (CDF) for a fair die roll with squared values on each side. In the PMF, the probabilities of each outcome (from 1^2 to 6^2) are plotted, each having an equal probability of 1/6. The CDF illustrates the cumulative probability of obtaining a value less than or equal to the given squared outcome. As expected for a fair die, the CDF increases linearly with each additional outcome, reaching a probability of 1 at 6^2.

2.1.6 Covariance

Covariance serves as a metric for quantifying the joint variability exhibited by two random variables. When the higher values of one variable predominantly align with the

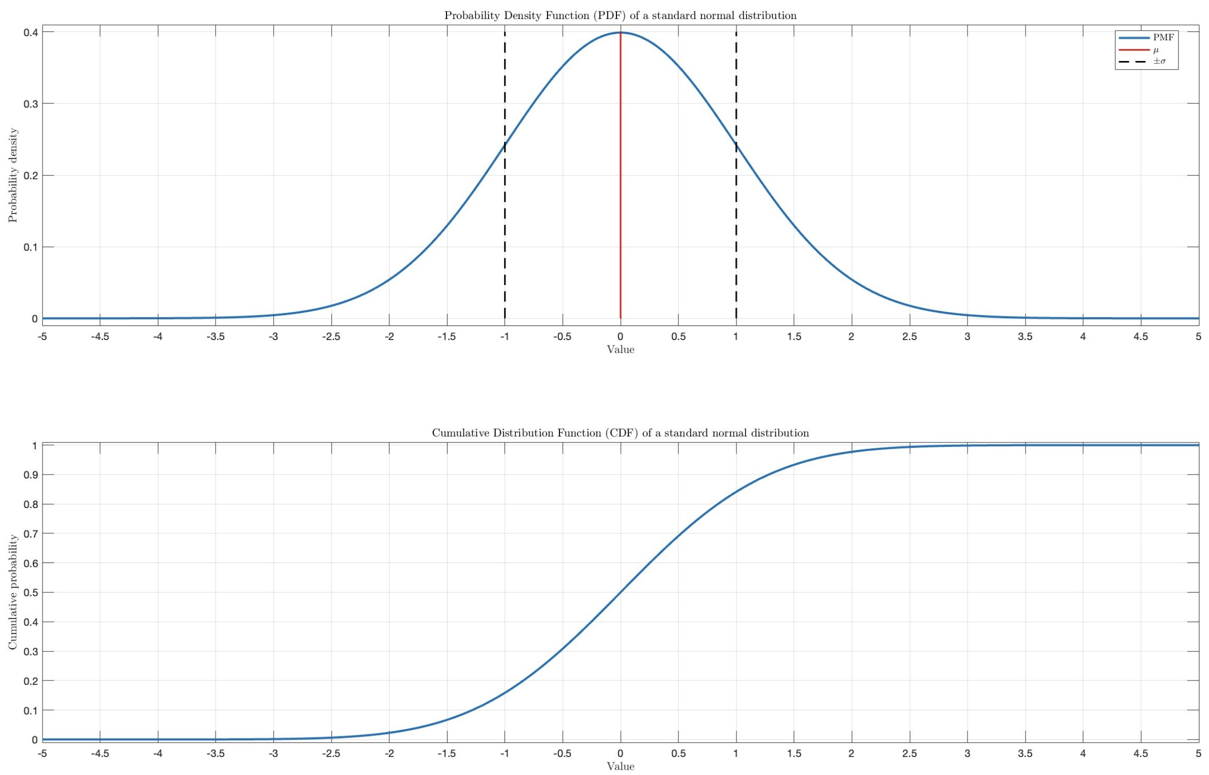


Figure 2.2: Probability Density Function (PDF) and Cumulative Distribution Function (CDF) for a standard normal distribution. The PDF illustrates the familiar bell-shaped curve, centered at 0, which represents the likelihood of each value in the continuous random variable. The CDF provides the cumulative probability of obtaining a value less than or equal to a given outcome. For the standard normal distribution, the CDF curve starts from 0, increases gradually in a sigmoidal shape, passing through 0.5 at the mean (which is 0), and asymptotically approaches 1 as the value goes towards positive infinity.

higher values of the other variable, and similarly, the lower values correspond, resulting in both variables displaying similar behavior, the covariance is positive. Conversely, when the greater values of one variable predominantly coincide with the lower values of the other, indicating opposite behavior between the variables, the covariance is negative. In essence, the sign of the covariance reflects the tendency in the linear relationship between these variables. The magnitude of the covariance is determined by the geometric mean of the variances that both random variables share in common.

For two jointly distributed real-valued random variables X and Y with finite second moments, the covariance is defined as the expected value (or mean) of the product of their deviations from their individual expected values. Formally, it is

$$\mathrm{Cov}(X, Y) = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X]E[Y] \tag{2.8}$$

where $E[X]$ is the expected value of X, also known as the mean of X. It is often to deal with sample populations and not with random variables. The covariance formula, when applied to a sample of N observations for the values of X and Y, takes the following form:

$$\mathrm{Cov}(X, Y) = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y}) \tag{2.9}$$

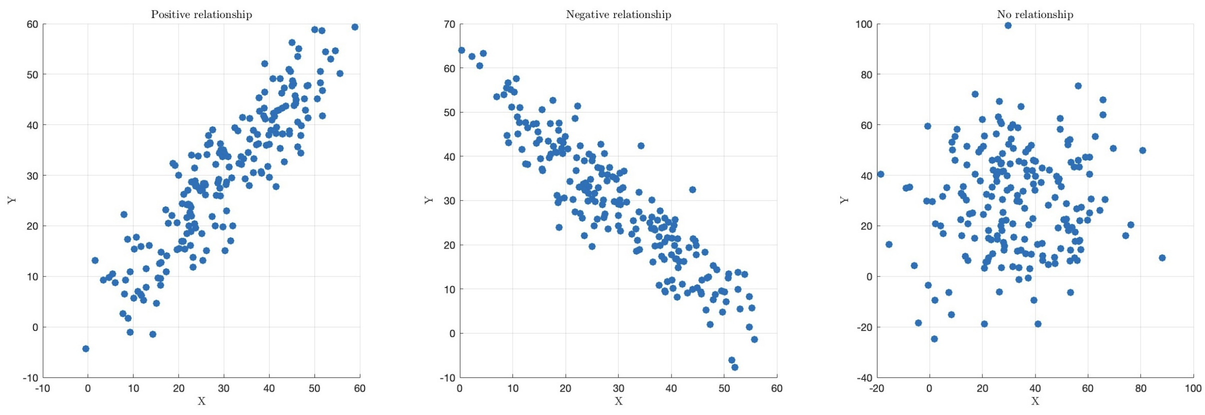


Figure 2.3: Scatter plots demonstrating covariances between two random variables X and Y under different relationships. Left plot: Represents a positive relationship between X and Y. As values of X increase, values of Y tend to also increase, suggesting a positive covariance. Middle plot: Illustrates a negative relationship between X and Y. An increase in values of X typically corresponds to a decrease in values of Y, implying a negative covariance. Right plot: Depicts no clear relationship between X and Y. The spread of data points is fairly uniform, suggesting that the covariance is close to zero or very minimal.

2.1.7 Pearson correlation coefficient

The Pearson correlation coefficient (PCC) serves as a correlation metric designed to assess the linear correlation between two sets of data. It quantifies the relationship by calculating the ratio between the covariance of these two variables and the product of their respective standard deviations. This computation normalizes the measurement of covariance, ensuring that the resulting value always falls within the range of -1 to 1. Similar to covariance, the Pearson correlation coefficient is exclusively capable of indicating linear correlations between variables. It does not account for or capture other forms of relationships or correlations that may exist between them.

Given a pair of random variables (X, Y), the PCC is given by

$$\rho_{X,Y} = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y} \tag{2.10}$$

Pearson correlation coefficient can, also, be applied to a sample of N observations for the values of X and Y. Given paired data $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ consisting of N pairs, then the formula of PCC is

$$\rho_{X,Y} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}} \tag{2.11}$$

2.1.8 Stochastic processes

A real-valued stochastic process is a sequence of random variables indexed by $t \in I$ on the probability space $(\Omega, \mathcal{A}, P)$:

$$\begin{aligned} I \times \Omega &\to \mathbb{R} \\ (t, \omega) &\to X(t, \omega) = X_t(\omega) \end{aligned}$$

where Ω is the sample space, $\omega \in \Omega$ is a state of the nature such that $x_t = X(t, \omega)$, $\mathcal{A}$ is a σ-algebra, and P is a probability measure [13].

We will always assume that the cardinality of I is infinite, either countable or uncountable. If the cardinality of I is finite, then X is not considered a stochastic process, but rather a random vector. It will be useful to consider separately the cases of discrete time and continuous time. If $I = \mathbb{Z}^+$, then we call $X = \{X_n, \quad n = 0, 1, 2, \ldots\}$ a **discrete time** stochastic process, and it is a countable collection of random variables indexed by the non-negative integers. If $I = \mathbb{R}^+$, then $X = \{X_t, \quad 0 \le t < \infty\}$ is said to be a **continuous time** stochastic processes, and it is an uncountable collection of random variables indexed by the non-negative real numbers.

2.1.9 Time-series

A realization of a stochastic process for a given $\omega \in \Omega$, $(X_t)_{t \in \mathbb{Z}^+}$, is the mapping defined by

$$\begin{aligned} \mathbb{Z}^+ &\to \mathbb{R} \\ t &\to x_t(\omega) \end{aligned}$$

The realization of a stochastic process is said to be a time-series or a chronological series [13].

In a more intuitive way, a time-series can be seen as a variable that is observed at different regular periods, $t = t_1, t_2, \ldots, t_k$. The time elapsed between two observations is constant, e.g., daily, monthly, quarterly or yearly. There are many datasets available in the form of such time-series. In that case, often, it is useful to calculate the sample mean and variance of the data. Assume that we have N real-valued observations, then

$$\hat{\mu} = \bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{2.12}$$

$$\hat{\sigma}^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})^2 \tag{2.13}$$

Also, we can identify the type of the distribution of our data by drawing the histogram and experiment with fitting various probability distributions until we find the one that fits our data the best.

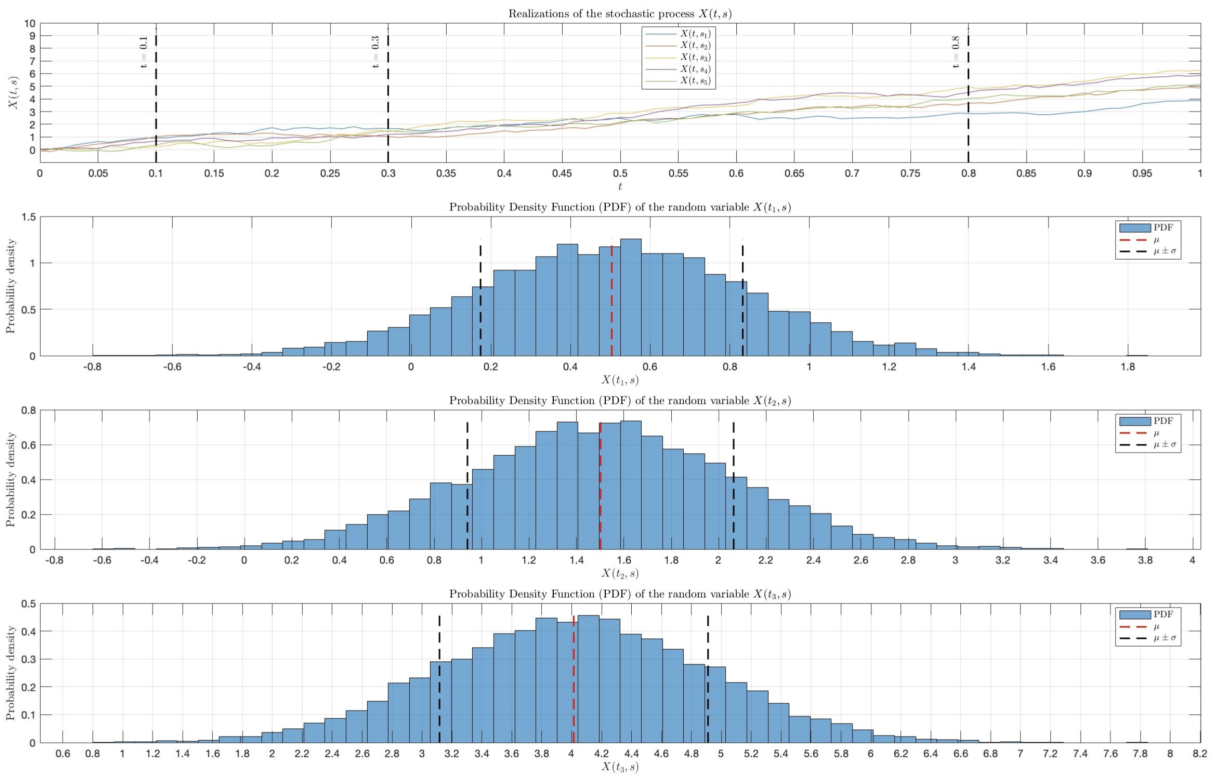


Figure 2.4: Let $X(t, s)$ be a stochastic process. At the first plot, we see five realizations of the stochastic process for different values of s. Each realization can be seen as a time-series. At the rest three plots we see the probability density function (PDF) of the random variables that result for three different values of t.

2.1.10 The auto-correlation and auto-covariance functions

We will introduce the auto-correlation function by first defining the auto-covariance function [14]. The **auto-covariance** function of a series $(X_t)_{t\in\mathbb{Z}^+}$ is defined as

$$\gamma_X(t, t+h) \equiv \text{Cov}(X_t, X_{t+h}) \tag{2.14}$$

where the definition of covariance is given by

$$\text{Cov}(X_t, X_{t+h}) \equiv E[X_t X_{t+h}] - E[X_t]E[X_{t+h}]$$

Similarly, the above expectations are defined as

$$E[X_t] \equiv \int_{-\infty}^{\infty} x f_t(x) dx$$

$$E[X_t X_{t+h}] \equiv \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x_1 x_2 f_{t,t+h}(x_1, x_2) dx_1 dx_2$$

where $f_t(x)$ and $f_{t,t+h}(x_1, x_2)$ denote, respectively, the density of X_t and the joint density of the pair (X_t, X_{t+h}). Considering the notation used above, it should be clear that X_t is assumed to be a continuous random variable. Since we generally consider stochastic processes with **constant zero mean**, we often have

$$\gamma_X(t, t+h) = E[X_t X_{t+h}]$$

In addition, in the context of this book we will normally drop the subscript referring to the time series (i.e. X in this case) if it is clear from the context which time series the auto-covariance refers to. For example, we generally use $\gamma(t, t+h)$ instead of $\gamma_X(t, t+h)$. Moreover, the notation is even further simplified when the covariance of X_t and X_{t+h} is the same as that of X_{t+j} and X_{t+h+j} (for all j), i.e. the covariance depends only on the time between observations and not on the specific time t. This is a consequence of an important property called stationarity that was mentioned earlier and will be discussed in the next section. In this case, we simply use the notation

$$\gamma(h) = \text{Cov}(X_t, X_{t+h}) \tag{2.15}$$

This is the definition of auto-covariance that will be used from this point on-wards and therefore this notation will generally be used throughout the text, thereby implying certain properties for the process $(X_t)_{t\in\mathbb{Z}^+}$ (i.e. stationarity). With this in mind, several remarks can be made on the auto-covariance function:

1. The auto-covariance function is symmetric. That is, $\gamma(h) = \gamma(-h)$ since $\text{Cov}(X_t, X_{t+h}) = \text{Cov}(X_{t+h}, X_t)$
2. The auto-covariance function "contains" the variance of the process as $\text{Var}(X_t) = \gamma(0)$
3. We have that $|\gamma(h)| \leq \gamma(0)$ for all h. The proof of this inequality is direct and follows from the Cauchy-Schwarz inequality, i.e.,
$$\begin{aligned}(|\gamma(h)|)^2 = (\gamma(h))^2 &= (E[(X_t - E[X_t])(X_{t+h} - E[X_{t+h}])])^2 \\ &\leq E[(X_t - E[X_t])^2]E[(X_{t+h} - E[X_{t+h}])^2] = (\gamma(0))^2\end{aligned}$$
4. Just as any covariance, $\gamma(h)$ is "scale dependent" since $\gamma(h) \in \mathbb{R}$. We therefore have:
    - if $|\gamma(h)|$ is "close" to zero, then X_t and X_{t+h} are "weakly" (linearly) dependent.
    - if $|\gamma(h)|$ is "far" from zero, then the two random variables present a "strong" (linear) dependence.

    However, it is generally difficult to assess what "close" and "far" from zero means in this case.
5. $\gamma(h) = 0$ does not imply that X_t and X_{t+h} are independent, but simply that they are uncorrelated. The independence is only implied by $\gamma(h) = 0$ in the jointly Gaussian case.

As hinted in the introduction, an important related statistic is the correlation of X_t with X_{t+h} or **auto-correlation** [14], which is defined as

$$\rho(h) = \text{Corr}(X_t, X_{t+h}) = \frac{\text{Cov}(X_t, X_{t+h})}{\sigma_{X_t}\sigma_{X_{t+h}}} = \frac{\gamma(h)}{\gamma(0)} \tag{2.16}$$

Similarly to $\gamma(h)$, it is important to note that the above notation implies that the auto-correlation function is only a function of the lag h between observations. Thus, auto-covariances and auto-correlations are one possible way to describe the joint distribution of a time series. Indeed, the correlation of X_t with X_{t+h} is an obvious measure of how persistent a time-series is. Remember that just as with any correlation:

1. $\rho(h)$ is "scale free" so it is much easier to interpret than $\gamma(h)$.
2. $|\rho(h)| \leq 1$ since $|\gamma(h)| \leq \gamma(0)$.
3. Causation and correlation are two very different things.

2.1.11 Univariate and multivariate time-series

A univariate time-series, as the name suggests, is a series with a single time-dependent variable. This type of time-series consists of scalar observations recorded sequentially over equal time increments. On the other hand, a multivariate time-series has more than one time-series variable. Each variable depends not only on its past values but also has some dependency on the other variables. This dependency is generally used for forecasting future values. In that type of time-series, particularly, we are interested to identify the degree of dependency between these variables. Granger's causality test can be used to identify the relationship between variables prior to model building. This is important because if there is no relationship between variables, they can be excluded and modeled separately. Conversely, if a relationship exists, the variables must be considered in the modeling phase.

2.1.12 Granger causality

Granger defined the causality relationship based on two principles [15]:

1. The cause happens prior to its effect.
2. The cause has unique information about the future values of its effect.

G-causality is normally tested in the context of linear regression models. For illustration, consider a bivariate linear autoregressive model of two variables, X_1 and X_2. It is

$$X_1(t) = \sum_{j=1}^{p} A_{11,j} X_1(t-j) + \sum_{j=1}^{p} A_{12,j} X_2(t-j) + E_1(t) \tag{2.17}$$

$$X_2(t) = \sum_{j=1}^{p} A_{21,j} X_1(t-j) + \sum_{j=1}^{p} A_{22,j} X_2(t-j) + E_2(t) \tag{2.18}$$

where p is the maximum number of lagged observations included in the model (the model order), the matrix $\mathbf{A}$ contains the coefficients of the model (i.e., the contributions of each lagged observation to the predicted values of $X_1(t)$ and $X_2(t)$, and E_1 and E_2 are residuals (prediction errors) for each time-series. If the variance of E_1 (or E_2) is reduced by the inclusion of the X_2 (or X_1) terms in the first (or second) equation, then it is said that X_2 (or X_1) Granger-(G)-causes X_1 (or X_2). In other words, X_2 G-causes X_1 if the coefficients in A_{12} are jointly significantly different from zero. This can be tested by performing an F-test of the null hypothesis that $A_{12} = 0$, given assumptions of covariance stationarity on X_1 and X_2. The magnitude of a G-causality interaction can be estimated by the logarithm of the corresponding F-statistic [16]. Note that model selection criteria, such as the Bayesian Information Criterion or the Akaike Information Criterion, can be used to determine the appropriate model order p [17].

2.2 Decomposition of time-series

Time-series data can display a multitude of distinctive patterns, and it proves advantageous to dissect a time series into multiple components, each representing a distinct underlying pattern category. In the context of this subsection, we explore several prevalent techniques used for extracting these components from time series data. Frequently, this decomposition process is undertaken to enhance comprehension of the time-series itself, but it also serves the purpose of refining forecast accuracy by isolating and modeling specific patterns or trends [18].

2.2.1 Components of a time-series

In describing a time-series, we use words such as "trend" and "seasonal", which need to be defined more carefully. A time-series is usually decomposed into[18]:

- **Trend component**: A trend exists when there is a long-term increase or decrease in the data. It does not have to be linear. Sometimes we will refer to a trend as "changing direction", when it might go from an increasing trend to a decreasing trend.

- **Seasonal component**: A seasonal pattern occurs when a time series is affected by seasonal factors, such as the time of the year or the day of the week. Seasonality is always of a fixed and known frequency.

- **Cyclic component**: A cycle occurs when the data exhibit rises and falls that are not of a fixed frequency. These fluctuations are repeated but non-periodic. The duration of these fluctuations depend on the nature of the time series.

- **Remainder component**: It describes random, irregular influences or "noise". It represents the residuals or remainder of the time series after all the other components have been removed.

Sometimes the trend and cyclical components are grouped into one, called the **trend-cycle component** [18]. The trend-cycle component can just be referred to as the "trend" component, even though it may contain cyclical behavior.

Seasonal behavior and cyclical behavior are frequently confused, but they are really quite different. If the fluctuations are not of a fixed frequency, then they are cyclic; if the frequency is unchanging and associated with some aspect of the calendar, then the pattern is seasonal. In general, the average length of cycles is longer than the length of a seasonal pattern, and the magnitudes of cycles tend to be more variable than the magnitudes of seasonal patterns.

Many time series include trend, cycles and seasonality. When choosing a forecasting method, we will first need to identify the time series patterns in the data, and then choose a method that is able to capture the patterns properly. If we assume an **additive decomposition**, then we can write [18]

$$y_t = S_t + T_t + R_t \tag{2.19}$$

where y_t is the data, S_t is the seasonal component, T_t is the trend-cycle component, and R_t is the remainder component, all at time t. Alternatively, a **multiplicative decomposition** would be written as [18]

$$y_t = S_t \cdot T_t \cdot R_t \tag{2.20}$$

An **alternative** to using a multiplicative decomposition is to first transform the data until the variation in the series appears to be stable over time, then use an additive decomposition. When a log transformation has been used, this is equivalent to using a multiplicative decomposition because [18]

$$y_t = S_t \cdot T_t \cdot R_t \quad \text{is equivalent to} \quad \log(y_t) = \log(S_t) + \log(T_t) + \log(R_t) \tag{2.21}$$

An additive model would be used when the variations around the trend do not vary with the level of the time series whereas a multiplicative model would be appropriate if the trend is proportional to the level of the time series.

2.2.2 Moving averages

The classical method of time series decomposition originated in the 1920s and was widely used until the 1950s. It still forms the basis of many time series decomposition methods, so it is important to understand how it works. The first step in a classical decomposition is to use a moving average method to estimate the trend-cycle [18].

Let y_t be a time series where $t = 0, 1, \ldots, n$, and y_t represents the observation at time t. A **simple moving average (SMA)** of order m at time t can be written as

$$\text{SMA}_t = \frac{1}{m} \sum_{i=0}^{m-1} y_{t-i} \tag{2.22}$$

where $m \in \mathbb{Z}$ is the window size or the number of periods over which we are averaging. The summation runs over the most recent m data points up to time t. Observations that are nearby in time are also likely to be close in value. Therefore, the average eliminates some of the randomness in the data, leaving a smooth trend-cycle component. We call this an m-MA, meaning a moving average of order m. The SMA provides a smoothed line which can help to identify the direction (upward or downward) of the underlying trend of the data. If the SMA line is rising, it suggests an upward trend, while a falling SMA line suggests a downward trend. By choosing an appropriate window size, you can filter out the higher-frequency fluctuations (like seasonal or irregular variations) to focus on the underlying cycle. For example, in monthly data with a yearly cycle, an SMA with a window size of 12 (i.e., 12 months) can help highlight the cyclical pattern.

In a **cumulative moving average (CMA)**, the data arrive in an ordered datum stream, and the user would like to get the average of all of the data up until the current datum. The CMA at time t is

$$\text{CMA}_t = \frac{1}{t} \sum_{i=0}^{t} y_i \tag{2.23}$$

CMA focuses on long-term behavior since it considers all prior data. While it is less commonly used than SMA for trend-cycle decomposition, CMA can provide insights into the overall trajectory or growth pattern of a series. Since CMA considers all past data, it's less suited for cycle identification, especially for longer time series where you might be interested in more recent cycles.

A **weighted moving average (WMA)** is an average that has multiplying factors to give different weights to data at different positions in the sample window. Mathematically, the weighted moving average is the convolution of the data with a fixed weighting function. The WMA at time t is

$$\mathrm{WMA}_t = \frac{\sum_{i=0}^{m-1} w_i y_{t-i}}{\sum_{i=0}^{m-1} w_i} \tag{2.24}$$

Here w_i is the weight assigned to the data point y_{t-i} within the moving window. Typically, weights decrease as data points get older. WMA can be more responsive to recent changes than SMA because of the weighting scheme. If recent data points are given higher weights, WMA will adjust more quickly to changes in the trend. Similar to the SMA, selecting an appropriate window size and weight distribution in WMA can help filter out high-frequency noise and highlight underlying cyclical patterns.

An **exponential moving average (EMA)**, also known as an exponentially weighted moving average (EWMA), is a first-order infinite impulse response filter that applies weighting factors which decrease exponentially. The weighting for each older datum decreases exponentially, never reaching zero. The EMA at time t is

$$\mathrm{EMA}_t = \alpha \cdot y_t + (1-\alpha) \cdot \mathrm{EMA}_{t-1} \tag{2.25}$$

where α is the smoothing factor, a number between 0 and 1. Larger values of α mean that the EMA will be more responsive to recent values. For the first calculation of the EMA (i.e., at $t = 1$), you can either use the first data point itself or an SMA as a starting value. EMA is particularly useful for trend identification in data with more volatility. Because it assigns exponentially decreasing weights to older data, it can react faster to recent changes in the trend than SMA. Adjusting the smoothing factor α can fine-tune its responsiveness. EMA can help identify cycles, especially if the cycles are somewhat irregular or if the amplitude of the cycle changes over time. Its sensitivity to recent data allows it to track more adaptive cycles.

In Figure 2.5, there is a synthetic time-series with all the aforementioned types of moving averages drawn on it.

2.2.3 Classical decomposition

The classical decomposition method originated in the 1920s. It is a relatively simple procedure, and forms the starting point for most other methods of time series decomposition. There are two forms of classical decomposition: an additive decomposition and a multiplicative decomposition [18]. These are described below for a time series with seasonal period m (e.g., $m = 4$ for quarterly data, $m = 12$ for monthly data, $m = 7$ for daily data with a weekly pattern). In classical decomposition, we assume that the seasonal component is constant from year to year. For multiplicative seasonality, the m values that form the seasonal component are sometimes called the "seasonal indices".

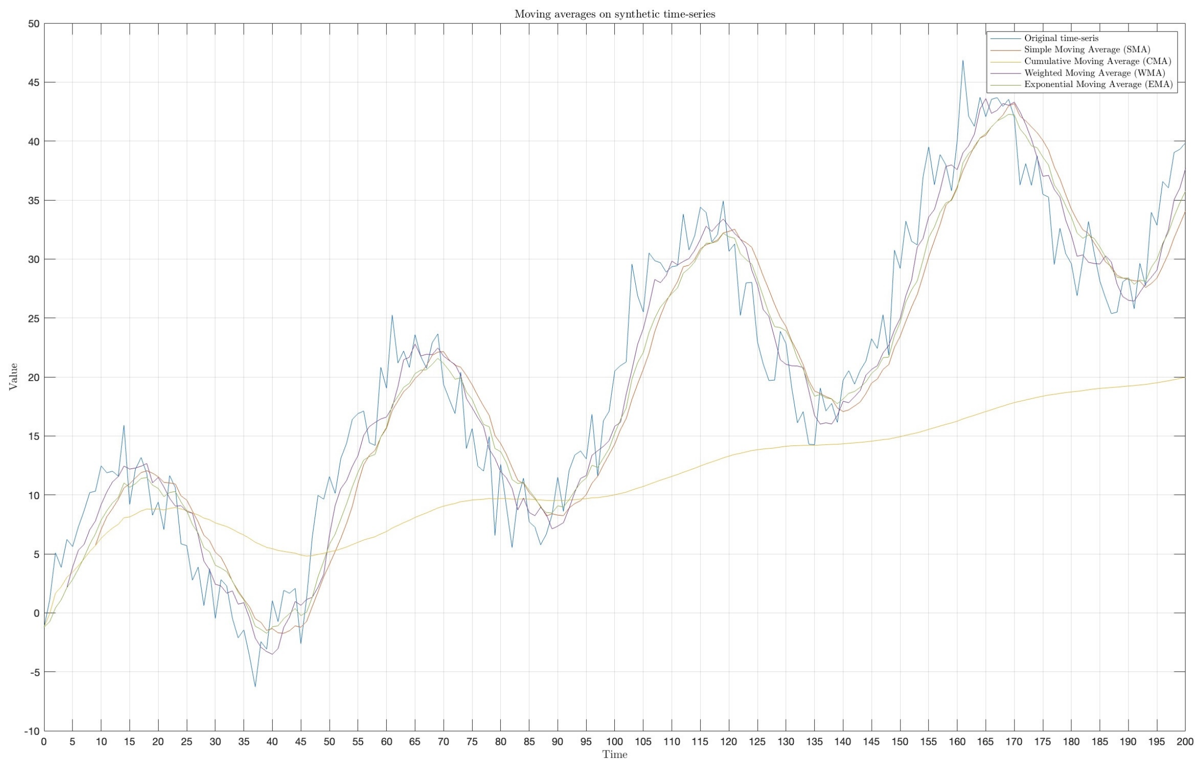


Figure 2.5: Synthetic time-series alongside its various moving averages: Simple (SMA), Cumulative (CMA), Weighted (WMA), and Exponential (EMA).

We will get through the basic steps of the classical decomposition method, for both, additive and multiplicative models. **Classical additive decomposition** includes the following steps [18]:

1. Compute the trend component $\hat{T}_t$. For example, we can use any of the moving average variants (e.g., m-SMA), or a linear regression based on ordinary least squares.
2. Calculate the detrended series by $y_t - \hat{T}_t$.
3. To estimate the seasonal component for each season, simply average the detrended values for that season. For example, with monthly data, the seasonal component for March is the average of all the detrended March values in the data. These seasonal component values are then adjusted to ensure that they add to zero. The seasonal component is obtained by stringing together these monthly values, and then replicating the sequence for each year of data. This gives $\hat{S}_t$.
4. The remainder component is calculated by subtracting the estimated seasonal and trend-cycle components: $\hat{R}_t = y_t - \hat{T}_t - \hat{S}_t$.

In Figure 2.6, there is a synthetic time-series, which is decomposed into trend, seasonality, and remainder, using the classical additive decomposition.

A **classical multiplicative decomposition** is similar, except that the subtractions are replaced by divisions. It includes the following steps [18]:

1. Compute the trend component $\hat{T}_t$. For example, we can use any of the moving average variants (e.g., m-SMA), or a linear regression based on ordinary least squares.

2. Calculate the detrended series by $\frac{y_t}{\hat{T}_t}$.
3. To estimate the seasonal component for each season, simply average the detrended values for that season. For example, with monthly data, the seasonal index for March is the average of all the detrended March values in the data. These seasonal indices are then adjusted to ensure that they add to m. The seasonal component is obtained by stringing together these monthly indices, and then replicating the sequence for each year of data. This gives $\hat{S}_t$.
4. The remainder component is calculated by subtracting the estimated seasonal and trend-cycle components: $\hat{R}_t = \frac{y_t}{\hat{T}_t \cdot \hat{S}_t}$.

In Figure 2.7, there is a synthetic time-series, which is decomposed into trend, seasonality, and remainder, using the classical multiplicative decomposition.

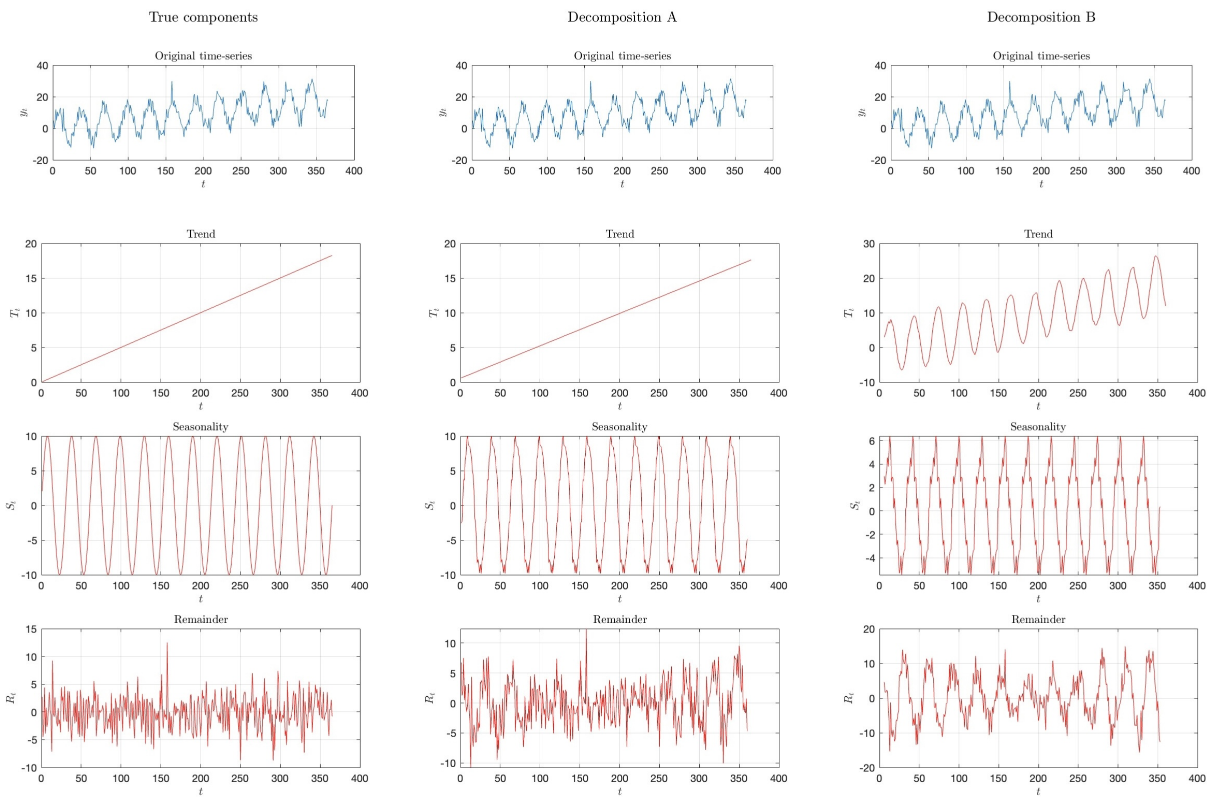


Figure 2.6: Decomposition of a synthetic time-series using an additive model comprising trend, seasonality, and remainder. Column 1: Original time-series along with its true components. Column 2: Classical decomposition using linear regression to estimate the trend. Column 3: Classical decomposition utilizing Simple Moving Average (SMA) for trend estimation. Both decomposition methods aim to isolate and represent the inherent trend, seasonality, and noise characteristics of the original series.

2.2.4 STL decomposition

Seasonal and Trend decomposition using Loess (STL) is a versatile, robust and widely-used time series decomposition technique that aids in the analysis of temporal data,

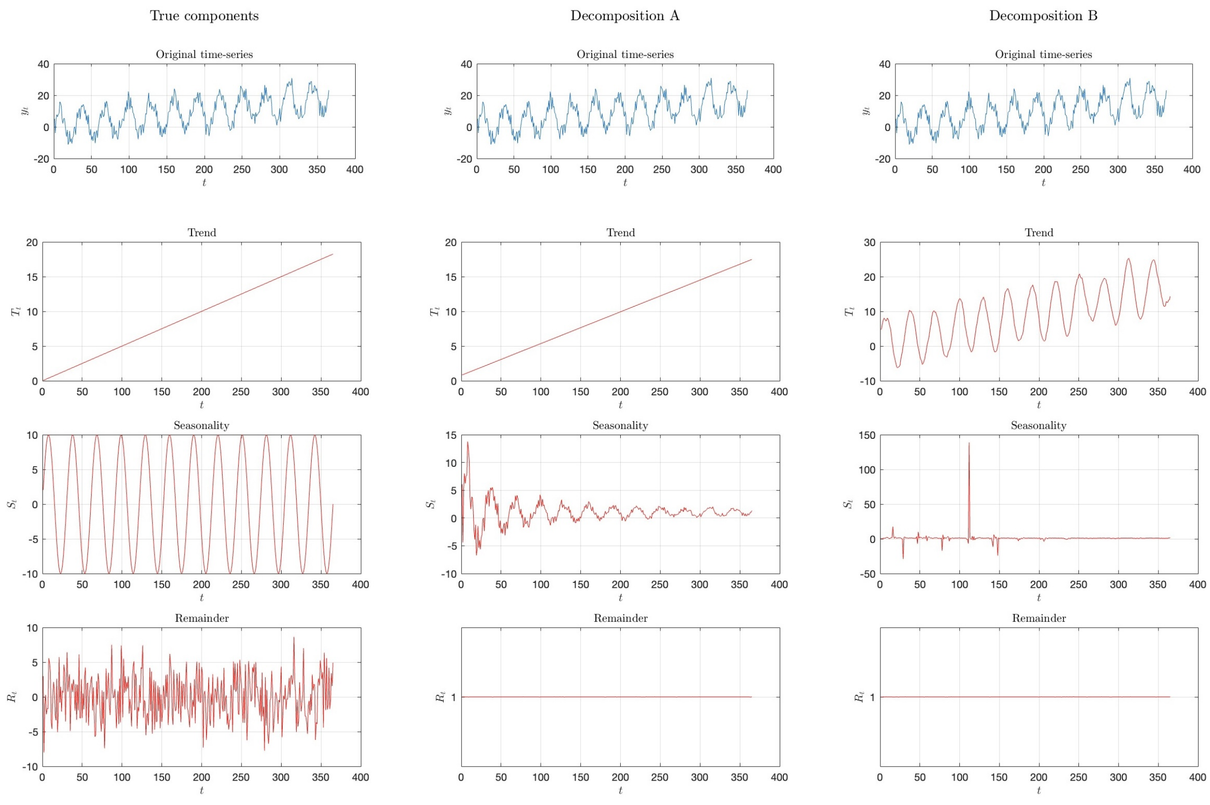


Figure 2.7: Decomposition of a synthetic time-series using a multiplicative model comprising trend, seasonality, and remainder. Column 1: Original time-series along with its true components. Column 2: Classical decomposition using linear regression to estimate the trend. Column 3: Classical decomposition utilizing Simple Moving Average (SMA) for trend estimation. Both decomposition methods aim to isolate and represent the inherent trend, seasonality, and noise characteristics of the original series.

while Loess is a method for estimating nonlinear relationships developed by Cleveland, McRae, and Terpenning (1990) [19], STL is particularly useful for decomposing time series data into its constituent components: the seasonal, trend, and remainder (or residual) components. This technique is valuable for various applications, including economics, environmental science, and epidemiology, where understanding underlying patterns is crucial.

STL offers several advantages for time-series analysis:

- **Robustness**: STL is robust to outliers and can handle data with irregular or non-uniform seasonal patterns.

- **Flexibility**: By adjusting the smoothing parameters, users can control the degree of smoothing applied to the seasonal and trend components, allowing for fine-tuning to the specific characteristics of the data. In addition, the seasonal component is allowed to change over time, and the rate of change can be controlled by the user.

- **Interpretability**: The decomposition into seasonal, trend, and remainder components makes it easier to interpret the underlying patterns and identify changes over time.

- **Prediction**: Once the components are separated, forecasting models can be applied to each component separately, which often leads to more accurate predictions.

The two main parameters to be chosen when using STL are the **trend-cycle window** (t.window) and the **seasonal window** (s.window). These control how rapidly the trend-cycle and seasonal components can change. Smaller values allow for more rapid changes. Both t.window and s.window should be odd numbers; t.window is the number of consecutive observations to be used when estimating the trend-cycle; s.window is the number of consecutive years to be used in estimating each value in the seasonal component. The user must specify s.window as there is no default. Setting it to be infinite is equivalent to forcing the seasonal component to be periodic (i.e., identical across years). Specifying t.window is optional, and a default value will be used if it is omitted. We will not get further into the STL decomposition here.

2.3 Stationarity

Stationarity is one of the most important concepts when working with time-series data. A stationary series is one in which the properties, i.e., mean, variance and covariance, do not vary with time. Let us understand this using an intuitive example. Consider the three plots shown in Figure 2.8 [20]:

- In the first plot, we can clearly see that the mean varies (increases) with time which results in an upward trend. Thus, this is a non-stationary series. For a series to be classified as stationary, it should not exhibit a trend.
- Moving on to the second plot, we certainly do not see a trend in the series, but the variance of the series is a function of time. As mentioned previously, a stationary series must have a constant variance.
- If we look at the third plot, the spread becomes closer as the time increases, which implies that the covariance is a function of time.

The three first examples shown in Figure 2.8 represent non-stationary time series. Now we look at the fourth plot in Figure 2.8. In this case, the mean, variance and covariance are constant with time. This is what a stationary time series looks like. Most statistical models require the series to be stationary to make effective and precise predictions.

2.3.1 Types of stationarity

A stochastic process (or a time-series $(X_t)_{t\in\mathbb{Z}}$ is said to be **strongly or strictly stationary** if the joint distribution of $(X_{t_1},\dots,X_{t_k})$ is identical to that of $(X_{t_1+t},\dots,X_{t_k+t})$ for all t, where k is an arbitrary positive integer and $(t_1,t_2,\dots,t_k)$ is a collection of k positive integers. An equivalent definition is given as follows. A stochastic process (or a time-series) $(X_t)_{t\in\mathbb{Z}}$ is said to be strongly or strictly stationary if the distribution of $(X_t)_{t\in\mathbb{Z}}$ is identical to that of $(X_t)_{t\in\mathbb{Z}}$ with $Y_t = X_t + h$. Strong stationarity is equivalent to say that the distribution is invariant over time. Strong stationarity is often too restrictive since it requires that the time-series is completely invariant over time, i.e., all moments are constant over time [13].

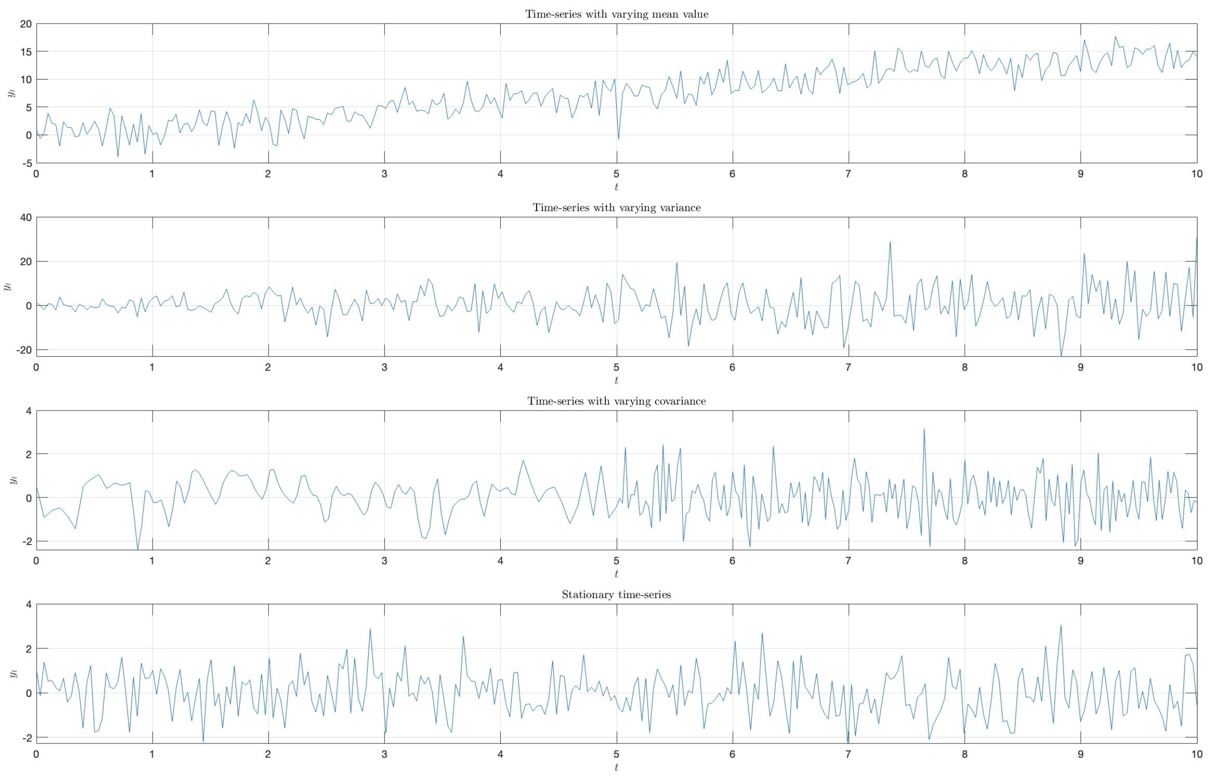


Figure 2.8: Various types of time-series. The first three are non-stationary. The fourth is stationary.

A stochastic process $(X_t)_{t\in\mathbb{Z}}$ is **weakly stationary**, if it satisfies the following properties:

1. $E[X_t] = m$ is time-invariant
2. $\text{Var}(X_t)$ is time-invariant
3. $\text{Cov}(X_t, X_{t+h}) = E[(X_t - m)(X_{t+h} - m)] = \gamma_X(h)$ is time-invariant

Weak stationarity exploits the "stability" of the first two moments, whereas strong stationarity implies the stability of all the moments (among others). Strong stationarity implies weak stationarity as long as the first two moments exist. The converse is not true in general [13].

The aim is to convert a non-stationary series into a strict stationary series for making predictions. Another type of stationarity is trend stationarity. A series that has no unit root but exhibits a trend is referred to as a **trend stationary** series [20]. Once the trend is removed, the resulting series will be strict stationary. The KPSS test classifies a series as stationary on the absence of unit root. This means that the series can be strict stationary or trend stationary. One more type of stationarity is difference stationarity. A time series that can be made strict stationary by differencing falls under **difference stationary** [20]. ADF test is also known as a difference stationarity test.

2.3.2 Dickey-Fuller test

The Dickey-Fuller (DF) test in statistics tests the null hypothesis that an autoregressive (AR) time series model contains a unit root. Depending on the test version being utilized, the alternative hypothesis may vary, but it is typically stationarity or trend-stationarity. The test is named after the statisticians David Dickey and Wayne Fuller, who developed it in 1979 [21].

A simple AR model is

$$y_t = \rho y_{t-1} + u_t \tag{2.26}$$

where y_t is the variable of interest, t is the time index, ρ is a coefficient, and u_t is the error term (assumed to be white noise). A unit root is present if $\rho = 1$. The model would be non-stationary in this case. The regression model can be written as

$$\Delta y_t = (\rho - 1)y_{t-1} + u_t = \delta y_{t-1} + u_t \tag{2.27}$$

where Δ is the first difference operator and $\delta \equiv \rho - 1$. This model is estimable, and checking for a unit root is the same as checking that $\delta = 0$. Standard t-distribution cannot be used to provide critical values since the test is conducted over the residual term rather than the raw data. As a result, the Dickey-Fuller table, which is a special distribution for this statistic t, was created. There are three main versions of the test [22]:

1. Test for a unit root:

$$\Delta y_t = \delta y_{t-1} + u_t \tag{2.28}$$

2. Test for a unit root with constant:

$$\Delta y_t = \alpha_0 + \delta y_{t-1} + u_t \tag{2.29}$$

3. Test for a unit root with constant and deterministic time trend:

$$\Delta y_t = \alpha_0 + \alpha_1 t + \delta y_{t-1} + u_t \tag{2.30}$$

Each version of the test has its own critical value which depends on the size of the sample. In each case, the null hypothesis is that there is a unit root, $\delta = 0$. The tests have low statistical power in that they often cannot distinguish between true unit-root processes ($\delta = 0$) and near unit-root processes (δ is close to zero). This is called the "near observation equivalence" problem [22].

The test is conceptualized as follows. It tends to go back to a fixed (or deterministically trending) mean if the series y is stationary (or trend-stationary). This means that small values will typically be followed by bigger values (positive changes), and large values by smaller values (negative changes). In light of this, the level of the series will have a negative coefficient and be a major predictor of the change in the following period. On the other hand, if the series is integrated, then positive changes and negative changes will happen with probabilities that are independent of the series' current level, much as how where you are at any one moment in a random walk has no bearing on which way you will move next [22].

2.3.3 Augmented Dickey-Fuller test

In statistics, an augmented Dickey–Fuller test (ADF) tests the null hypothesis that a unit root is present in a time series sample [23]. The alternative hypothesis is different depending on which version of the test is used, but is usually stationarity or trend-stationarity. It is an augmented version of the Dickey–Fuller test for a larger and more complicated set of time series models. The augmented Dickey–Fuller (ADF) statistic, used in the test, is a negative number. The more negative it is, the stronger the rejection of the hypothesis that there is a unit root at some level of confidence [24].

The testing procedure for the ADF test is the same as for the Dickey–Fuller test, but it is applied to the model

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \sum_{i=1}^{p-1} \delta_i \Delta y_{t-i} + u_t \tag{2.31}$$

where α is a constant, β the coefficient on a time trend and p the lag order of the autoregressive process. Imposing the constraints $\alpha = 0$ and $\beta = 0$ corresponds to modelling a random walk, and using the constraint $\beta = 0$ corresponds to modeling a random walk with a drift. Consequently, there are three main versions of the test, analogous to the ones discussed on Dickey–Fuller test [25].

By including lags of the order p the ADF formulation allows for higher-order autoregressive processes. This means that the lag length p has to be determined when applying the test. One possible approach is to test down from high orders and examine the t-values on coefficients. An alternative approach is to examine information criteria such as the Akaike information criterion, Bayesian information criterion or the Hannan–Quinn information criterion [25].

The unit root test is then carried out under the null hypothesis $\gamma = 0$ against the alternative hypothesis of $\gamma < 0$. Once a value for the test statistic

$$\mathrm{DF}_\tau = \frac{\hat{\gamma}}{\mathrm{SE}(\hat{\gamma})} \tag{2.32}$$

is computed, it can be compared to the relevant critical value for the Dickey–Fuller test. As this test is asymmetrical, we are only concerned with negative values of our test statistic DF_τ. If the calculated test statistic is less (more negative) than the critical value, then the null hypothesis of $\gamma = 0$ is rejected, and no unit root is present [25].

The intuition behind the test is that if the series is characterised by a unit root process, then the lagged level of the series (y_{t-1}) will provide no relevant information in predicting the change in y_t besides the one obtained in the lagged changes (Δy_{t-k}). In this case, $\gamma = 0$ and null hypothesis is not rejected. In contrast, when the process has no unit root, it is stationary and hence exhibits reversion to the mean - so the lagged level will provide relevant information in predicting the change of the series and the null hypothesis of a unit root will be rejected [25].

2.3.4 Augmented Dickey-Fuller Generalized Least Squares test

The ADF-GLS test (or DF-GLS test) is a test for a unit root in a time-series sample. It was developed by Elliott, Rothenberg and Stock (ERS) in 1992 [26] as a modification of the augmented Dickey–Fuller test (ADF).

	No constant, No trend				Constant, No trend				Constant, Trend			
N	0.01	0.025	0.05	0.10	0.01	0.025	0.05	0.10	0.01	0.025	0.05	0.10
25	-2.661	-2.273	-1.955	-1.609	-3.724	-3.318	-2.986	-2.633	-4.375	-3.943	-3.589	-3.238
50	-2.612	-2.246	-1.947	-1.612	-3.568	-3.213	-2.921	-2.599	-4.152	-3.791	-3.495	-3.181
100	-2.588	-2.234	-1.944	-1.614	-3.498	-3.164	-2.891	-2.582	-4.052	-3.722	-3.452	-3.153
250	-2.575	-2.227	-1.942	-1.616	-3.457	-3.136	-2.873	-2.573	-3.995	-3.683	-3.427	-3.137
500	-2.570	-2.224	-1.942	-1.616	-3.443	-3.127	-2.867	-2.570	-3.977	-3.670	-3.419	-3.132
> 500	-2.567	-2.223	-1.941	-1.616	-3.434	-3.120	-2.863	-2.568	-3.963	-3.660	-3.413	-3.128

Table 2.1: Critical values for Dickey–Fuller t-distribution [2].

A unit root test determines whether a time series variable is non-stationary using an autoregressive model. For series featuring deterministic components in the form of a constant or a linear trend, then ERS developed an asymptotically point optimal test to detect a unit root. This testing procedure dominates other existing unit root tests in terms of power. It locally de-trends data series to efficiently estimate the deterministic parameters of the series, and use the transformed data to perform a usual ADF unit root test. This procedure helps to remove the means and linear trends for series that are not far from the non-stationary region [26].

Consider a simple time-series model $y_t = d_t + u_t$ with $u_t = \rho u_{t-1} + e_t$ where d_t is the deterministic part and u_t is the stochastic part of y_t. When the true value of ρ is close to 1, estimation of the model, i.e. d_t will pose efficiency problems, because y_t will be close to non-stationarity. In this setting, testing for the stationarity features of the given times series will also be subject to general statistical problems. To overcome such problems, ERS suggested to locally difference the time series [27].

Consider the case where closeness to 1 for the autoregressive parameter is modelled as $\rho = 1 - \frac{c}{T}$ where T is the number of observations. Now consider filtering the series using $1 - \frac{\bar{c}}{T}L$ with L being a standard lag operator, i.e. $\bar{y}_t = y_t - \frac{\bar{c}}{T}y_{t-1}$. Working with $\bar{y}_t$ would result in power gain, as ERS show, when testing the stationarity features of y_t using the augmented Dickey-Fuller test. This is a point optimal test for which $\bar{c}$ is set in such a way that the test would have a 50 percent power when the alternative is characterized by $\rho = 1 - \frac{c}{T}$ for $c = \bar{c}$. Depending on the specification of d_t, $\bar{c}$ will take different values [27].

2.3.5 Kwiatkowski–Phillips–Schmidt–Shin test

In econometrics, Kwiatkowski–Phillips–Schmidt–Shin (KPSS) [28] tests are used for testing a null hypothesis that an observable time series is stationary around a deterministic trend (i.e. trend-stationary) against the alternative of a unit root.

Contrary to most unit root tests, the presence of a unit root is not the null hypothesis but the alternative. Additionally, in the KPSS test, the absence of a unit root is not a proof of stationarity but, by design, of trend-stationarity. This is an important distinction since it is possible for a time series to be non-stationary, have no unit root yet be trend-stationary. In both unit root and trend-stationary processes, the mean can be growing or decreasing over time; however, in the presence of a shock, trend-stationary processes are mean-reverting (i.e. transitory, the time series will converge again towards the growing mean, which was not affected by the shock) while unit-root processes have a permanent impact on the mean (i.e. no convergence over time) [29].

Later, Denis Kwiatkowski, Peter C. B. Phillips, Peter Schmidt and Yongcheol Shin (1992) [28] proposed a test of the null hypothesis that an observable series is trend-stationary (stationary around a deterministic trend). The series is expressed as the sum of deterministic trend, random walk, and stationary error, and the test is the Lagrange multiplier test of the hypothesis that the random walk has zero variance. KPSS-type tests are intended to complement unit root tests, such as the Dickey–Fuller tests. By testing both the unit root hypothesis and the stationarity hypothesis, one can distinguish series that appear to be stationary, series that appear to have a unit root, and series for which the data (or the tests) are not sufficiently informative to be sure whether they are stationary or integrated [30].

ADF Test	**KPSS Test**	**Result**
Non-Stationary	Non-Stationary	Series is not stationary
Stationary	Stationary	Series is stationary
Non-Stationary	Stationary	Trend stationary, remove trend to make series strict stationary
Stationary	Non-Stationary	Difference stationary, use differencing to make series stationary

Table 2.2: In general, it is always better to apply both tests, ADF and KPSS, so that we are sure that the series is truly stationary. This table shows the possible outcomes of applying these stationary tests.

2.3.6 Phillips–Perron test

In statistics, the Phillips–Perron test (PP), named after Peter C. B. Phillips and Pierre Perron, is a unit root test [31]. It is used in time-series analysis to test the null hypothesis that a time series is integrated of order 1. It builds on the Dickey–Fuller test of the null hypothesis $\rho = 1$ in $\Delta y_t = (\rho - 1)y_{t-1} + u_t$, where Δ is the first difference operator [32]. Like the augmented Dickey–Fuller test, the Phillips–Perron test addresses the issue that the process generating data for y_t might have a higher order of autocorrelation than is admitted in the test equation-making y_{t-1} endogenous and thus invalidating the Dickey–Fuller t-test. Whilst the augmented Dickey–Fuller test addresses this issue by introducing lags of Δy_t as regressors in the test equation, the Phillips–Perron test makes a non-parametric correction to the t-test statistic. The test is robust with respect to unspecified autocorrelation and heteroscedasticity in the disturbance process of the test equation.

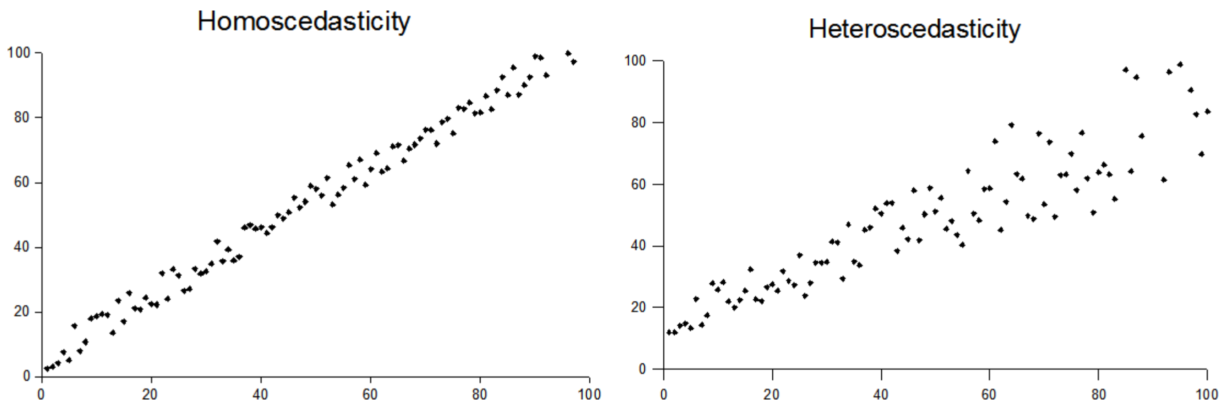
Davidson and MacKinnon (2004) [33] report that the Phillips–Perron test performs worse in finite samples than the augmented Dickey–Fuller test.

2.4 Homoscedasticity and heteroscedasticity

In statistics, a sequence or vector of random variables is considered **homoscedastic** if all its constituent random variables possess the same finite variance. This property is also referred to as homogeneity of variance. Conversely, the complementary concept is termed **heteroscedasticity**, which signifies a condition where the random variables exhibit varying variances. Assuming that a variable is homoscedastic when it is, in reality, heteroscedastic can lead to unbiased but inefficient point estimates and biased estimates of standard errors. Such a misjudgment may also result in an overestimation of the goodness of fit, as quantified by the Pearson correlation coefficient.

In regression analysis, we talk about heteroscedasticity in the context of the residuals or error term. We are assuming a linear regression model applied on time-series data. The time-series is considered to be heteroscedastic if there is a systematic change in the spread of the residuals over the range of measured values. In other words, heteroscedasticity is a condition where the error variance is not constant on the independent variable. on the other hand, homoscedasticity is a condition where a variance error is constant in any condition of the independent variable. Time series with non-constant variance often have a long-tailed distribution. The data is left- or right-skewed.

The presence of heteroscedasticity poses a significant challenge in regression analysis and the analysis of variance. It undermines the validity of statistical significance tests that presuppose all modeling errors to have identical variances. In addition, many statistical methods for time-series analysis assume homoscedasticity. Consequently, it is crucial to mitigate this phenomenon. One way to do that, is by applying an appropriate transformation on the data. Such transformations are the logarithmic, the k-logarithmic, or the Box-Cox (see Section 4).



(a) Plot with random data showing homoscedasticity. At each value of x, the y-value of the dots has about the same variance.

(b) Plot with random data showing heteroscedasticity. The variance of the y-values of the dots increase with increasing values of x.

Figure 2.9: Illustration of homoscedasticity and heteroscedasticity.

3 Time-Series Forecasting

Time-series forecasting is an important component of data analysis that aims to predict future values based on historical observations. As data collection becomes larger and more sophisticated, the demand for accurate forecasts to support decision-making processes has led to the integration of machine learning techniques into time-series analysis.

3.1 Machine learning in time-series analysis

Machine learning techniques have found widespread application in time-series analysis due to their ability to capture complex patterns and relationships in time data. These applications can perform extensive classification and regression tasks.

In **classification**, a label or category is assigned to a given data point. Chronologically, the classification task may include the identification of events, anomalies, or trends in a sequence of observations. For example, determining whom a voice recording is, or categorizing an electrocardiography signal as normal or give the type of abnormality, fall under the umbrella of classification. On the other hand, the **regression** predicts a continuous numerical value depending on the input variables. In time-series analysis, regression models are often used to predict future values of a variable based on its historical behavior. This fits well with the primary goal of time-series forecasting, which is to forecast the future based on past observations.

In this **thesis**, our primary focus lies in the realm of **time-series regression** and **forecasting**. Regression techniques aim to model the relationship between the input features and the target variable, allowing us to predict future values based on historical patterns. Forecasting, a specific form of regression, is centered around predicting future observations within a time-series dataset.

3.2 Regression in machine learning

Regression is a statistical modeling technique used to understand the relationship between a dependent variable and one or more independent variables. It aims to build a mathematical model capable of making predictions or estimating the values of the dependent variable for new or unseen data points. The dependent variable is also known as the target variable or the outcome variable, while the independent variables are often referred to as predictors, features, or input vector.

In regression, the **dependent variable** is usually continuous, meaning it can take on any numerical value within a given range. The **independent variables** can be either **continuous** or **categorical**. If there is only one independent variable, it is called **simple regression**, while multiple independent variables are referred to as **multiple regression**.

The most common type of regression is **linear regression**, where the relationship between the variables is assumed to be linear. Other types of regression include **polynomial regression**, which allows for curved relationships, and **logistic regression**, which is used when the dependent variable is categorical. However, more advanced techniques have been developed to capture complex relationships and improve prediction accuracy.

Gaussian process regression is a flexible and non-parametric approach that models the relationship between variables as a distribution of functions. It allows for uncertainty estimation and can capture non-linear relationships effectively. By defining a prior distribution over functions, Gaussian process regression provides a posterior distribution that represents the possible functions consistent with the data.

Neural networks are a type of computational model inspired by the structure and function of biological neurons. In the context of regression, neural networks can learn complex patterns and relationships by organizing interconnected layers of artificial neurons. They can approximate non-linear functions and handle large amounts of data. Training a neural network involves adjusting the weights and biases of its neurons to minimize the prediction error.

Regression trees, also known as decision trees, are a non-parametric approach that partitions the data space into regions and assigns a prediction value to each region. Each internal node of the tree represents a splitting criterion based on one of the independent variables, while the leaf nodes contain the predicted values. Regression trees are capable of capturing non-linear relationships and interactions between variables. Ensembles of regression trees, such as random forests or gradient boosting, further improve prediction accuracy.

Regression analysis involves various steps, including data collection, data pre-processing, model selection, and model evaluation. The quality of the regression model is assessed using metrics such as the coefficient of determination (**R-squared**), mean squared error (**MSE**), root mean squared error (**RMSE**), relative root mean squared error (**RRMSE**), or mean absolute error (**MAE**).

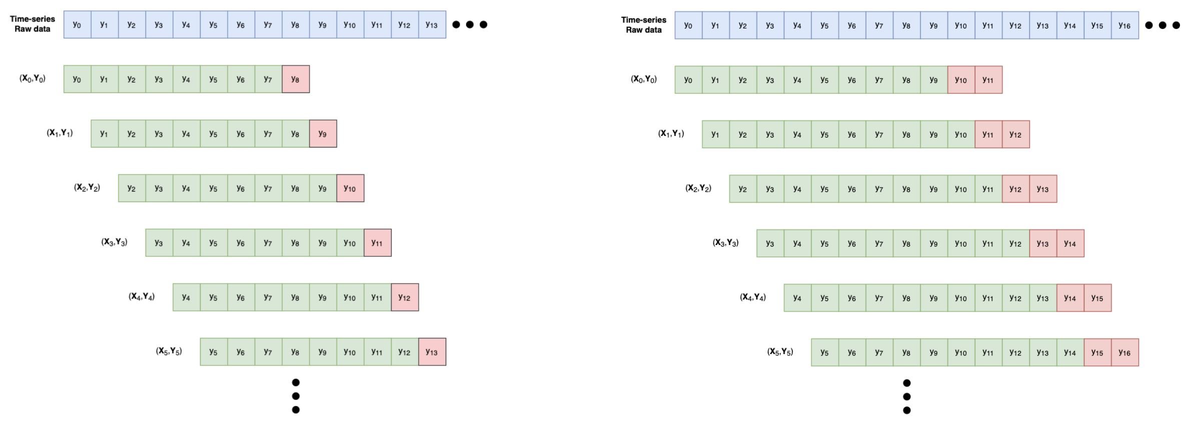
Overall, regression analysis is a powerful tool and has numerous applications across different fields, including economics, finance, social sciences, healthcare, and engineering. It is commonly used for forecasting, trend analysis, impact assessment, and understanding the relationship between variables.

3.3 Creating a data set from time-series raw data

When preparing a time-series dataset for a machine learning algorithm that creates a forecasting model, the first thing to do is to create the data set from which the algorithm will learn. Often, at the beginning, we have raw data, in the sense of time-indexed observations $\{y_0, y_1, \ldots, y_N\}$. It is important to understand the form of the data set that the algorithm accepts as input and see how we can get from raw data to that form.

Some machine learning algorithms, that are more based on statistical models, require the data to be in pairs of time step t and the corresponding observation y_t. In that case, the time-series raw data do not need any processing. We have $\{t_0, t_1, \ldots, t_N\}$, and $\{y_0, y_1, \ldots, y_N\}$, respectively. Such algorithm is the Gaussian Process Regression framework.

In deep learning models, the objective is to structure raw data into pairs of features and targets. In this context, features consist of sequential observations from the time-series data, while the target corresponds to the subsequent observation in the sequence. For instance, we have feature vector $\{y_i, y_{i+1}, \ldots, y_{i+n-1}\} \in \mathbb{R}^n$, which means that the model can look back at n past observations, and target vector $\{y_{i+n}, y_{i+n+1}, \ldots, y_{i+n+m-1}\} \in \mathbb{R}^m$, which means that the model can look front at m time steps in the future. This approach enables the deep learning model to learn patterns and relationships within the time-series data. By organizing the data in this way, the model can effectively capture temporal dependencies and make predictions about future values based on the provided historical information. This process involves creating sliding windows over the time-series, where each window represents a set of consecutive observations used as features, and the following observation is designated as the target. This data set structuring facilitates the training of deep learning models for tasks like forecasting, as it enables the model to learn and generalize patterns in the temporal data. In Figure 3.1, we visualize the process of creating a data set of pairs of feature vector and the corresponding target by applying a sliding window on the time-series raw data.



(a) Sliding window of size 9. The feature vector is of size 8, and the target is the subsequent observation (single value).

(b) Sliding window of size 12. The feature vector is of size 10, and the target is the subsequent 2 observations.

Figure 3.1: Illustration when creating a data set of pairs of feature vector and the corresponding target by applying a sliding window on the time-series raw data.

3.4 Data set splitting

Building algorithms that can learn from data and generate predictions is a common challenge in machine learning. Such algorithms work by creating a mathematical model from incoming data and then making data-driven predictions or decisions [34]. Typically, the input data required to develop the model are split into different data sets. Training, validation, and test sets are the three data sets that are most frequently utilized at various phases of model construction.

A **training data set** is a data set of examples used during the learning process and is used to fit the parameters of the machine learning model [35]. For example, a supervised learning algorithm looks at the training data set to determine, or learn, the optimal combinations of parameters that will generate a good predictive model. The goal is to produce a trained (fitted) model that generalizes well to new, unknown data.

A data set of examples used to adjust the model's hyperparameters (or architecture) is known as a **validation data set** [35]. It may also go by the names "dev set" or "development set". The number of hidden units within each layer is an illustration of a hyperparameter for artificial neural networks. It should have the same probability distribution as the training data set, as should the testing set. In addition to the training and test datasets, a validation data set is required in order to prevent overfitting when any classification parameter needs to be adjusted. For instance, if the best classifier for the problem is sought after, the training data set is utilized to train the various candidate classifiers, the validation data set is utilized to compare their performances and select the best classifier, and finally, the test data set is utilized to obtain performance characteristics such as accuracy, sensitivity, specificity, F-measure, and other metrics. The validation data set performs as a hybrid: it is training data that is utilized for testing, but not as part of the initial low-level training or the last round of testing.

The following is the fundamental methodology for **selecting a model** using validation data set [35]:

> Since our objective is to identify the network that performs the best on unseen data, the most straightforward method for comparing various networks is to assess the error function using data that is separate from the training data. Different networks are trained by minimizing of an appropriate error function established with respect to a training data set. The network with the minimum error relative to the validation set is chosen after comparing the performance of the networks by assessing the error function using an independent validation set. The **hold-out method** is the name given to this strategy. The performance of the chosen network should be verified by assessing its performance on a third independent set of data called a test set because this approach can result in some overfitting to the validation set.

This approach is used in **early stopping**, where the candidate models are successive iterations of the same network, and training is stopped when the error on the validation set increases, selecting the prior model (the one with the least error).

An independent data set that shares the same probability distribution as the training data set is referred to as a **test data set** [35]. There has not been any overfitting if a model that fits the training data set also fits the test data set well. Overfitting is typically shown by the training data set fitting the model better than the test data set. Therefore, a test set is a collection of instances used solely to evaluate the effectiveness (i.e. generalization) of a fully described classifier. To do this, classifications of cases in the test set are predicted using the final model. To evaluate the model's precision, those predictions are contrasted with the actual classifications of the cases.

In Figure 3.2, we visualize how the time-series raw data are first split into training, validation, and test sets, and then each set is formed into pairs of features and targets (when the purpose is to work with a deep learning model).

3.5 Overfitting and the bias-variance trade-off

3.5.1 Overfitting and underfitting

Overfitting is defined as "the production of an analysis that corresponds too closely or exactly to a particular set of data, and may, therefore, fail to fit to additional data or predict future observations reliably" [36]. A mathematical model is said to be overfitted if it has more parameters than the data can support [37]. These parameters, in a mathematical sense, represent for example the degree of a polynomial. Overfitting is the process

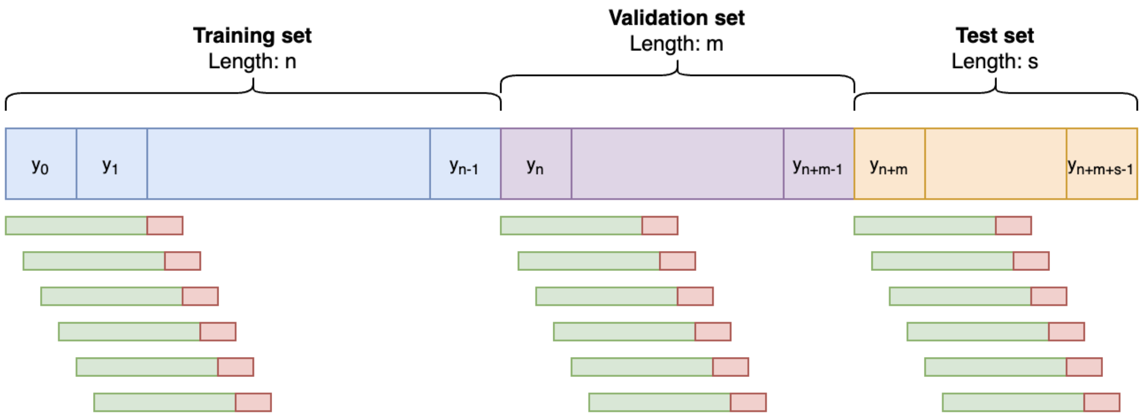


Figure 3.2: Visualization of the time-series raw data, and the split into training set, validation set, and test set. Next, we apply a sliding window method at each set in order to create the corresponding data sets (when the data is intended for a deep learning model). The data sets are in the form of pairs of feature vectors and the corresponding target vectors.

of unintentionally extracting some residual variance, also known as noise, and mistaking it for underlying model structure [38]. Because the criteria used to choose the model and the criteria to assess a model's fitness are different, there is a chance of overfitting. Overfitting happens when a model starts to "memorize" training data rather than "learning" to generalize from a trend. For instance, a model may be chosen by maximizing its performance on some set of training data, but its suitability may be determined by its ability to perform well on unseen data [39].

Underfitting is the inverse of overfitting, meaning that the statistical model or machine learning algorithm is too simplistic to accurately capture the patterns in the data. An underfitted model is one that lacks certain parameters or terms that would be present in a properly defined model. For instance, underfitting could happen when fitting a linear model to non-linear data. Such a model will typically do poorly in terms of prediction [39].

3.5.2 Bias–variance trade-off

In statistics and machine learning, the **bias–variance trade-off** is the property of a model that the variance of the parameter estimated across samples can be reduced by increasing the bias in the estimated parameters [40]. The bias–variance dilemma or bias–variance problem is the conflict in trying to simultaneously minimize these two sources of error that prevent supervised learning algorithms from generalizing beyond their training set:

- **Bias** in machine learning refers to the difference between a model's predictions and the actual distribution of the value it tries to predict. Models with high bias oversimplify the data distribution function, resulting in high errors in both the training outcomes and test data analysis results [41].

- **Variance** stands in contrast to bias; it measures how much a distribution on several

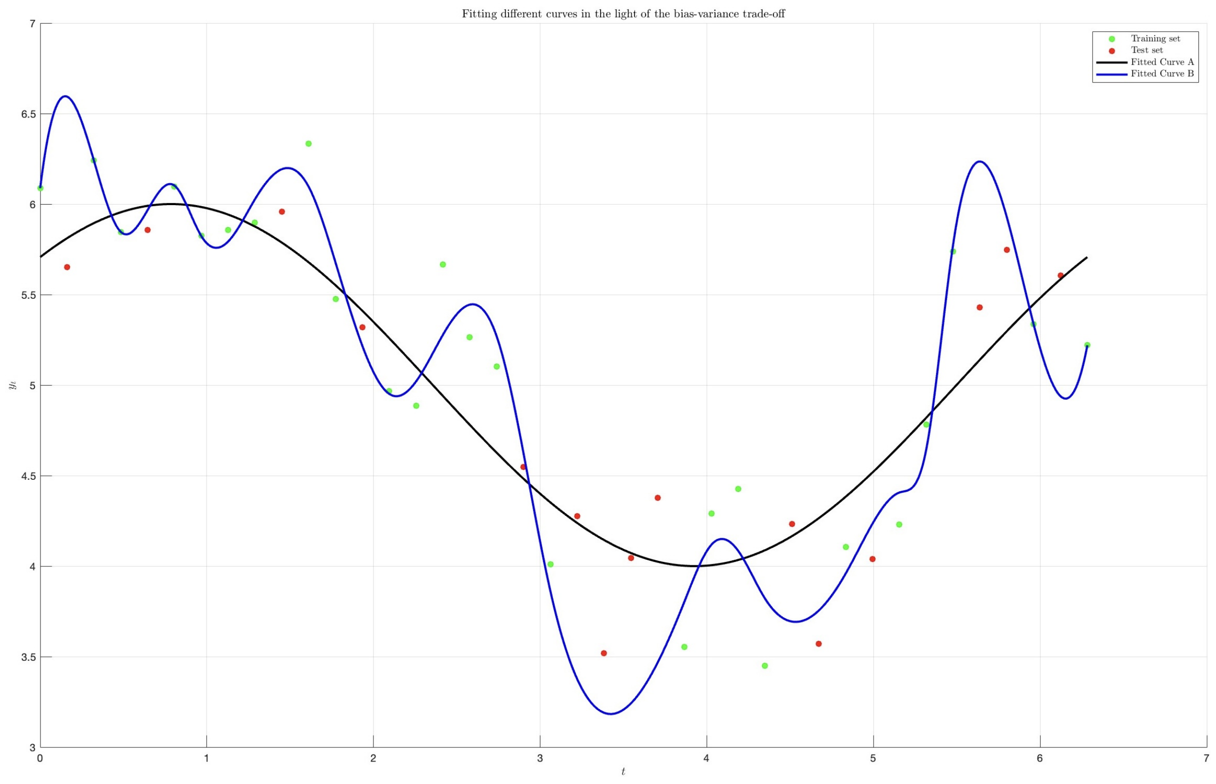


Figure 3.3: Different curves fitted to the same training data set. Curve A has high bias and low variance, whereas curve B has low bias and high variance. Curve B is overfitted to the training data set, which results in bad performance to the test set. In contrast, curve A has not so good performance in the training set, but succeeds to generalize and performs good on the test set.

sets of data values differs from each other. The most common approach to measuring variance is by performing cross-validation experiments and looking at how the model performs on different random splits of your training data [41].

The bias–variance trade-off is a central problem in supervised learning. Ideally, one wants to choose a model that both accurately captures the regularities in its training data, but also generalizes well to unseen data. Unfortunately, it is typically impossible to do both simultaneously. High-variance learning methods may be able to represent their training set well but are at risk of overfitting to noisy or unrepresentative training data. In contrast, algorithms with high bias typically produce simpler models that may fail to capture important regularities (i.e., underfit) in the data [40].

Now let's take a look at the different combinations of bias and variance in machine learning models and the results they provide [41]:

- **Low bias, low variance: ideal model**
  A machine learning model with low bias and low variance is considered ideal but is not often the case in the machine learning practice, so we can speak of "reasonable bias" and "reasonable variance".

- **Low bias, high variance: results in overfitting**
  This combination results in inconsistent predictions that are accurate on average. It

occurs when a model has too many parameters and fits too closely to the training data.

- **High bias, low variance: results in underfitting**
  Predictions are consistent but inaccurate on average in this scenario. This happens when the model does not learn well from the training data or has too few parameters, leading to underfitting issues.

- **High bias, high variance: results in inaccurate predictions**
  With both high bias and high variance, the predictions are both inconsistent and inaccurate on average.

In Figure 3.4, there is an illustration of the four cases for low and high bias and variance. In Figure 3.3, there is some synthetic data, where the fitting of two curves with different bias-variance is illustrated.

Bias-variance decomposition is a mathematical technique that divides the generalization error in a predictive model into two components: bias and variance. In machine learning, as you try to minimize one component of the error (e.g., bias), the other component (e.g., variance) tends to increase, and vice versa. Finding the right balance of bias and variance is key to creating an effective and accurate model.

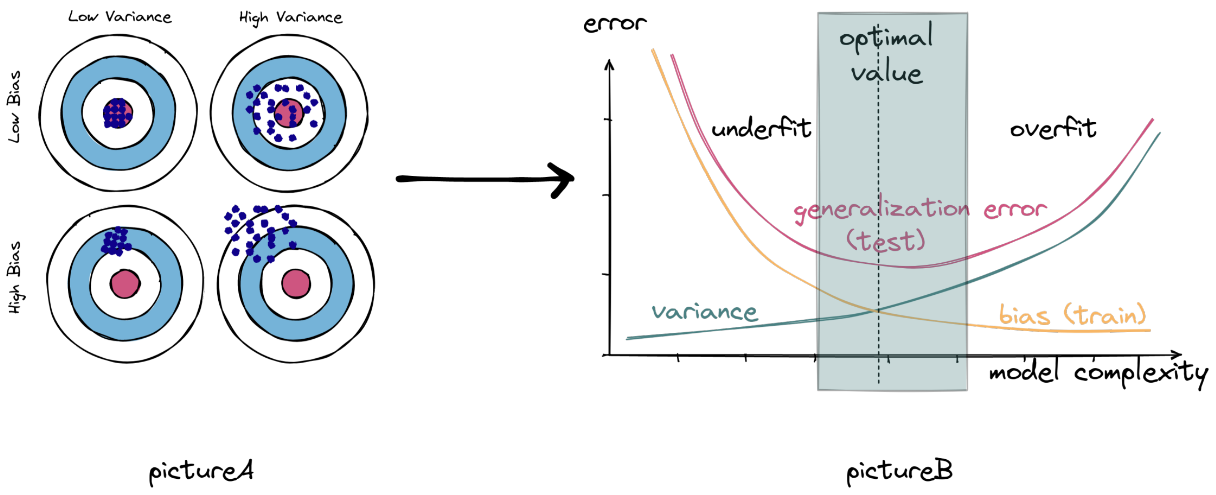


Figure 3.4: The bias-variance trade-off. With increased model complexity, the model can more accurately match the underlying relation at the risk of increasing the variance (amount of overfitting). The bias-variance trade-off corresponds to minimizing the total prediction error, which is the sum of bias and variance in the light of the bias-variance decomposition.

3.6 Forecasting strategies

Generally, time-series forecasting describes predicting the observation at the next time step. This is called a one-step forecast, as only one time step is to be predicted. There are some time-series problems where multiple time steps must be predicted. Contrasted to the one-step forecast, these are called multi-step time-series forecasting problems. Below, we will see some of the most common forecasting strategies for both cases, one-step and multi-step ahead predictions.

3.6.1 One-step ahead prediction

One-step forecast refers to the prediction of the series value at the next time step. This type of strategy is defined below:

- **Direct one-step ahead forecast**:
  Assume that we have a time-series of observations y_t. Let `model` be the trained model that takes into account n past observations $\{y_t, y_{t-1}, \ldots, y_{t-n+1}\}$ and makes a prediction for the next time step $\hat{y}_{t+1}$. We have

$$\hat{y}_{t+1} = \texttt{model}(y_t, y_{t-1}, \ldots, y_{t-n+1}) \tag{3.1}$$

3.6.2 Multi-step ahead prediction

Multi-step forecast refers to the prediction of the series values at the next m time steps. We define the following variations of this type of strategy:

- **Recursive one-step ahead forecast**:
  This strategy utilizes the one-step method in a recursive way in order to perform a multi-step forecast. Assume that we have a time-series of observations y_t. Let `model` be the trained model that takes into account n past observations $\{y_t, y_{t-1}, \ldots, y_{t-n+1}\}$ and makes a prediction for the next time step $\hat{y}_{t+1}$. We have

$$\begin{aligned}
\hat{y}_{t+1} &= \texttt{model}(y_t, y_{t-1}, \ldots, y_{t-n+1}) \\
\hat{y}_{t+2} &= \texttt{model}(\hat{y}_{t+1}, y_t, y_{t-1}, \ldots, y_{t-n+2}) \\
\hat{y}_{t+3} &= \texttt{model}(\hat{y}_{t+2}, \hat{y}_{t+1}, y_t, y_{t-1}, \ldots, y_{t-n+3}) \\
&\vdots \\
\hat{y}_{t+m} &= \texttt{model}(\hat{y}_{t+m-1}, \ldots, \hat{y}_{t+1}, y_t, \ldots, y_{t-n+m})
\end{aligned} \tag{3.2}$$

- **Multiple output multi-step ahead forecast**:
  Assume that we have a time-series of observations y_t. Let `model` be the trained model that takes into account n past observations $\{y_t, y_{t-1}, \ldots, y_{t-n+1}\}$ and makes a prediction of the series values at the next m time steps $\{\hat{y}_{t+1}, \hat{y}_{t+2}, \ldots, \hat{y}_{t+m}\}$. We have

$$\hat{y}_{t+1}, \hat{y}_{t+2}, \ldots, \hat{y}_{t+m} = \texttt{model}(y_t, y_{t-1}, \ldots, y_{t-n+1}) \tag{3.3}$$

- **Direct multi-step ahead forecast**:
  The direct multi-step prediction method involves developing and training a separate model for each forecast time step. Assume that we have a time-series of observations y_t. Let $\{\texttt{model\_1}, \texttt{model\_2}, \ldots, \texttt{model\_m}\}$ be the m trained models that take into

account n past observations $\{y_t, y_{t-1}, \dots, y_{t-n+1}\}$ and make a prediction of the series values at the next m time steps $\{\hat{y}_{t+1}, \hat{y}_{t+2}, \dots, \hat{y}_{t+m}\}$, respectively. We have

$$\begin{aligned} \hat{y}_{t+1} &= \texttt{model\_1}(y_t, y_{t-1}, \dots, y_{t-n+1}) \\ \hat{y}_{t+2} &= \texttt{model\_2}(y_t, y_{t-1}, \dots, y_{t-n+1}) \\ &\vdots \\ \hat{y}_{t+m} &= \texttt{model\_m}(y_t, y_{t-1}, \dots, y_{t-n+1}) \end{aligned} \tag{3.4}$$

Note that **multiple output multi-step ahead forecast** and **direct multi-step ahead forecast** can, also, be utilized in a recursive way, when someone wants to look very deep in the future.

3.6.3 Recursive strategies

Recursive forecasting strategies involve making predictions for future time periods by utilizing previously predicted values as inputs for subsequent predictions. This approach updates forecasts step by step, using the most recent forecasted value as a basis for predicting the next one. This recursive technique is commonly used when someone wants to extend the forecasting horizon, without changing the developed models.

Recursive forecasting strategies offer the advantage of adaptability, as they can swiftly capture short-term fluctuations by using recent predictions as the basis for subsequent forecasts. They also allow real-time updates, enabling adjustments as new data emerges without the need to reanalyze the entire historical dataset. These methods are relatively simple to implement and are computationally efficient. However, they come with the risk of cumulative errors that can magnify over time, potentially leading to less accurate long-term predictions. Additionally, the reliance on recent data might result in a limited memory of past patterns, undermining the utilization of valuable historical information. Moreover, these strategies can be vulnerable to instability when faced with sudden data changes or noise, as errors could propagate and disrupt the forecasting process. Therefore, the decision to employ a recursive forecasting approach should carefully consider the nature of the data and the trade-off between short-term precision and long-term reliability.

In the context of model assessment on a test set, a recursive technique can be employed with a slight alteration. Instead of employing previously predicted values as inputs for subsequent forecasts, the true observations can be utilized, taking advantage of the complete knowledge of actual outcomes in the test set. By using the true observations as inputs, this approach assesses the model's performance in a more realistic scenario, allowing for a direct comparison between the model's predictions and the actual outcomes. This methodology offers insights into how well the model can adapt to new data points and provides a clear measure of its predictive accuracy on unseen data. While retaining the adaptability and real-time updating benefits of recursive strategies, this modified approach ensures a more accurate evaluation of the model's predictive capabilities by eliminating the potential influence of cumulative errors that might arise from using predicted values as inputs.

3.7 Cross-validation

3.7.1 Cross-validation in general

Assessing the performance of a model is a crucial step in its development. Cross-validation emerges as a statistical technique designed to facilitate this process effectively. In the context of k-fold cross-validation, the dataset is partitioned into k folds. The model is then trained on all but one fold, and its performance is evaluated on the excluded fold. This process iterates until the model has been tested on each fold, and the final performance metrics are calculated as the average of the scores obtained across all folds. The utility of k-fold cross-validation extends beyond mere model evaluation; it serves as a method for hyperparameters fine-tuning and a potent tool to mitigate overfitting, ensuring that the model's performance assessment is more robust and reliable compared to a simple train-test split. Figure 3.5 depicts the split of the data in a typical 5-fold cross-validation.

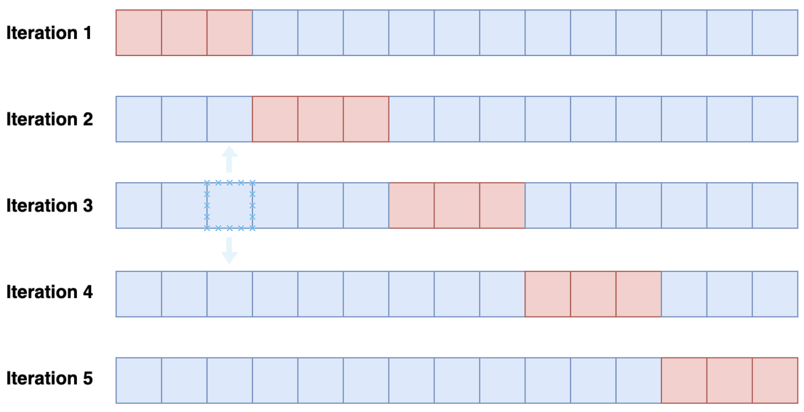


Figure 3.5: Illustration of the data set splits when performing an ordinary 5-fold cross-validation. This method cannot be used in a time-series data set.

This concept forms the foundation of cross-validation, a widely embraced practice in the machine learning realm. The most commonly adopted technique involves the initial train-test split and then the random selection of samples from the available training data set and their division into training and validation sets. To be precise, the standard steps typically encompass the following:

1. Split randomly data in train and test set.
2. Focus on train set and split it again randomly in chunks (called folds).
3. Let's say we got 10 folds; train on 9 of them and test on the 10th.
4. Repeat step three 10 times to get 10 accuracy measures (scores) on 10 different and separate folds.
5. Compute the average of the 10 scores, which is the final reliable number telling us how the model is performing.

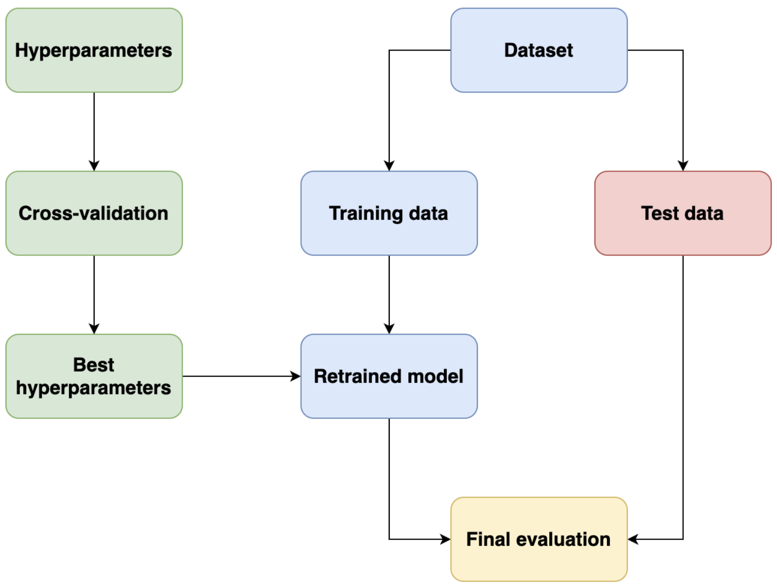


Figure 3.6: Illustration of the flowchart when performing cross-validation in order to fine-tune the model (i.e., select the best hyperparameters).

3.7.2 Cross-validation in time-series

Traditional techniques like cross-validation, which entail random data partitioning into training and validation subsets, operate under the assumption of independent and identically distributed (i.i.d.) observations. This assumption does not hold in the case of time series data, where observations are intrinsically reliant on each other. In simple word, we want to avoid future-looking when we train our model. There is a temporal dependency between observations, and we must preserve that relation during testing. Nonetheless, certain situations in time-series data permit the application of modified cross-validation techniques. Such techniques are explained below.

3.7.3 Prequential growing blocks

One method that can be used for cross-validating the time-series model is cross-validation on a rolling basis. Start with a small subset of data for training purpose, forecast for the later data points and then checking the accuracy for the forecasted data points. The same forecasted data points are then included as part of the next training dataset and subsequent data points are forecasted. A visualization of this method is shown in Figure 3.7.

3.7.4 Prequential sliding blocks

In this approach, the process of training and testing the model is performed by sliding over blocks of data, rather than adding testing data to the training data after each iteration. This means that the testing data is not combined with the training data during the iterative process. A visualization of this method is shown in Figure 3.8.

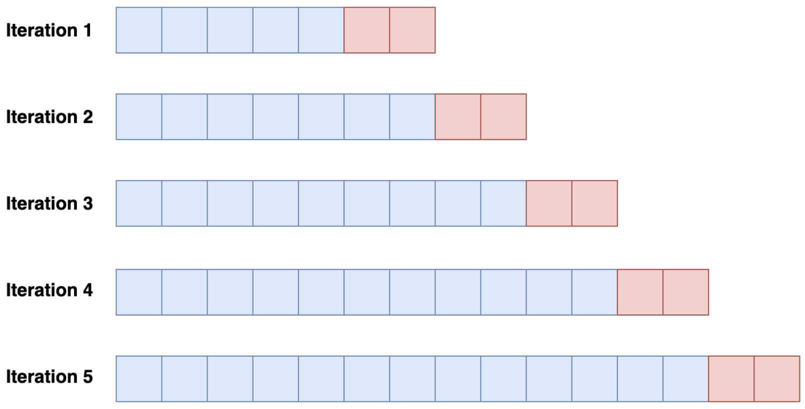


Figure 3.7: Illustration of the data set splits when performing 5-fold cross-validation using prequential growing blocks in order to fine-tune the model (i.e., select the best hyperparameters). Blue color represents training set, while red color represents validation set.

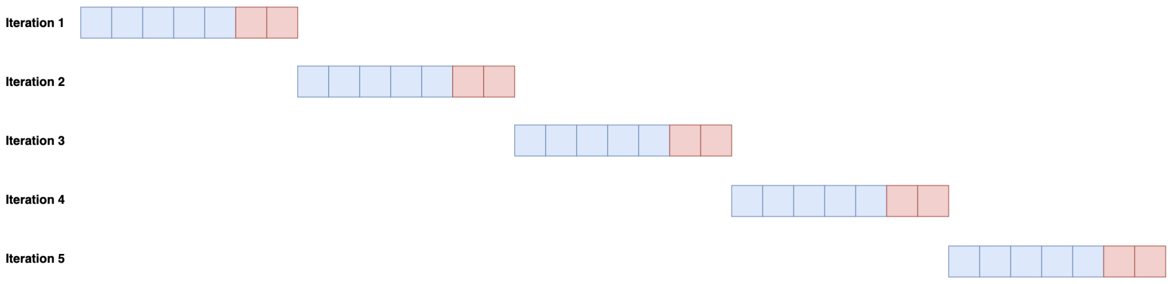


Figure 3.8: Illustration of the data set splits when performing 5-fold cross-validation using prequential sliding blocks in order to fine-tune the model (i.e., select the best hyperparameters). Blue color represents training set, while red color represents validation set.

3.8 Hyperparameters tuning

Hyperparameters are a model's in-built configuration variables. These variables require fine-tuning to produce a better performing model. These parameters are model dependent and vary from model to model.

Hyperparameters generally have a significant impact on the success of machine learning algorithms. A poorly configured machine learning model may perform no better than chance, while a well configured one could achieve state-of-the-art result. The process to find the best hyperparameters could be really tedious and is more of an art than science. This process of fine-tuning model parameters is called hyperparameter optimization.

3.8.1 Grid search

Grid search is the most basic algorithmic method for hyperparameter optimisation. It's like running nested loops on all possible values of your inbuilt features. Grid search is an example of uninformed search, meaning the next of feature set is independent of the output of the last runs. This method, also, requires retraining in every iteration, which incurs a huge cost.

3.8.2 Random search

Random search is grid search where the next feature set is selected randomly and the total number of runs have a upper limit. Random search is another example of uninformed search meaning the next of feature set is independent of the output of the last runs and it require retraining in every iteration which incurs a huge cost.

3.8.3 Sequential Model Based Optimization (SMBO)

Sequential Model Based Optimization (SMBO) minimizes validation loss by sequentially selecting different hyperparameter sets, where the next set is selected by **Bayesian reasoning** (dependent on the previous runs). Intuitively speaking, SMBO looks back at the result of last runs to focus future searches on areas which look more promising.

SMBO is utilized whenever the fitness function ($f : \mathbf{X} \rightarrow \mathbb{R}$) is costly to evaluate. In such cases, an approximate f (surrogate M) is calculated. This model M is cheaper to compute. Typically, the inner loop in SMBO is the numerical optimization of this surrogate or some transformation of this surrogate (line 3 in below code). The point $\mathbf{x}_*$ that maximizes the surrogate becomes the proposal for where the true loss function f should be evaluated (line 4).

In essence, after each run of hyperparameters on the objective function, the algorithm makes an educated guess which set of hyperparameters is most likely to improve the score and should be tried in the next run. It is done by getting regressor predictions on many points (hyperparameter sets) and choosing the point that is the best guess based on the so-called acquisition function. **Acquisition function** (S) defines a balance between exploring new areas in the objective space and exploiting areas that are already known to have favorable values. There are quite a few acquisition function options to choose from:

- **EI and PI**: **Negative expected improvement and Negative probability improvement**. Basically, when our algorithm is looking for the next set of hyperparameters, we can decide how small of the expected improvement we are willing to try on the actual objective function. The higher the value, the bigger the improvement (or probability of improvement) our regressor expects.

- **LCB**: **Lower confidence bound**. In this case, we want to choose our next point carefully, limiting the downside risk. We can decide how much risk we want to take at each run. By making the kappa parameter small we lean toward exploitation of what we know, by making it larger we lean toward exploration of the search space.

Algorithm 3.1 shows a generic SMBO pseudocode. The explanation of that pseudocode is the following:

- Line 1 : Initiate an empty $\mathcal{H}$.

- Line 2 : Initiate the loop with a fixed number of trials.

- Line 3 : (i) Learn a surrogate function for f called M. (ii) Define a evaluation criterion which needs to be minimized. (iii) $S(\mathbf{x}, M)$ is run for multiple instances of $\mathbf{x}$ to find $\mathbf{x}_*$ which minimizes S.
- Line 4 : f is evaluated for $\mathbf{x}_*$.
- Line 5 : $\mathcal{H}$ is updated with current value of $\mathbf{x}_*$, $f(\mathbf{x}_*)$.
- Line 6 : M is updated after every iteration to become a better approximation of f.
- Line 7 : The process ends after a fixed number of iterations.

Algorithm 3.1: A generic sequential model based optimization algorithm

```
Input: True function: $f$, Initial surrogate function to approximate $f$: $M_0$,
       Number of trials: $T$, and Acquisition function that computes the next
       hyperparameter assignment: $S$
Output: $\mathcal{H}$
1 $\mathcal{H} \leftarrow \emptyset$;
2 for $t \leftarrow 1$ to $T$ do
3     $\mathbf{x}_* \leftarrow \arg\min_{\mathbf{x}} S(\mathbf{x}, M_{t-1})$;
4     Evaluate $f(\mathbf{x}_*)$;
5     $\mathcal{H} \leftarrow \mathcal{H} \cup (\mathbf{x}_*, f(\mathbf{x}_*))$;
6     Fit a new model $M_t$ to $\mathcal{H}$;
```

It is worth noting, that different flavors of SMBO use different algorithms to optimize EI, for example. Two commonly used methods to optimize EI are **Tree Parzen Estimators (TPE)**, and **Gaussian processes (GPs)**.

The TPE replaces the generative process of choosing parameters from the search space in a tree like fashion with a set of non-parametric distributions. It replaces choices for parameter distributions with either a truncated Gaussian mixture, an exponentiated truncated Gaussian mixture or a re-weighted categorical to form two densities—one density for the loss function, where the loss is below a certain threshold, and another for a density where the loss function is above a certain threshold for values in the hyperparameter space. With each sampled configuration from the densities that is evaluated, the densities are updated to make them represent the true loss surface more precisely.

Instead of using the tree regressors, in the GP method the objective function is approximated by the Gaussian process. Nonetheless, we will not delve into more details here.

3.9 Evaluation metrics

In the domain of machine learning, regression tasks involve predicting continuous numerical values. When developing regression models, it is essential to assess the model's performance and reliability. Various evaluation metrics are employed to quantify the quality of predictions and to assist in model selection and refinement. This section discusses some of the most commonly used regression evaluation metrics, including Mean Absolute

Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Relative Root Mean Squared Error (RRMSE), and R-squared (R^2).

3.9.1 Mean Absolute Error (MAE)

Mean Absolute Error (MAE) is a straightforward metric that measures the average absolute difference between predicted and actual values. It is computed as the average of the absolute differences between each predicted value $\hat{y}$ and the corresponding true value y. Considering N points, we have

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{3.5}$$

MAE is useful because it provides a clear understanding of the average prediction error in the same units as the target variable. Smaller MAE values indicate better model performance.

3.9.2 Mean Squared Error (MSE)

Mean Squared Error (MSE) is another widely used metric that measures the average squared difference between predicted values $\hat{y}$ and actual values y. Considering N points, we have

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.6}$$

MSE places more weight on larger errors than MAE and is sensitive to outliers. It is commonly used in machine learning because it is mathematically convenient for optimization.

3.9.3 Root Mean Squared Error (RMSE)

Root Mean Squared Error (RMSE) is derived from MSE by taking the square root of the average squared differences. Considering the predicted values $\hat{y}$, the actual values y, and N points, we have

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{3.7}$$

RMSE has the advantage of providing an error metric in the same units as the target variable, making it interpretable. It also penalizes larger errors more significantly than MAE, which can be desirable in certain applications.

3.9.4 Relative Root Mean Squared Error (RRMSE)

Relative Root Mean Squared Error (RRMSE) is a normalized version of RMSE that expresses the error as a percentage of the range of the target variable. It helps to assess the error relative to the magnitude of the data. RRMSE is calculated as follows:

$$RRMSE = \frac{RMSE}{\sqrt{\sum_{i=1}^{N} \hat{y}_i^2}} \cdot 100\% \tag{3.8}$$

RRMSE values below 10% are often considered indicative of a good model fit.

3.9.5 R-squared (R^2)

R-squared (R^2), also known as the coefficient of determination, quantifies the proportion of the variance in the target variable that is explained by the regression model. R^2 values range from 0 to 1, where 0 indicates that the model explains none of the variance, and 1 indicates a perfect fit. It is computed as:

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (\bar{y} - \hat{y}_i)^2} \tag{3.9}$$

where $\bar{y} = \sum_{i=1}^{N} y_i$, RSS is the residual sum of squares, and TSS is the total sum of squares. Higher R^2 values suggest better model fit, but it should be interpreted alongside other metrics.

4 Transformations and Data Warping

Time-series data often needs pre-processing to ensure stationarity or to fulfill certain assumptions of the modeling approach being used, such as normality. In this section, we will delve into some useful and well-known transformations used in time series analysis: Differencing, Standardization, Box-Cox, and κ-Logarithmic transformation.

4.1 Differencing

A stationary time series is one that has constant mean, variance, and autocorrelation over time [18]. This means that the statistical properties of the series do not change with time or depend on the time period. Thus, time series with trends, or with seasonality, are not stationary — the trend and seasonality will affect the value of the time series at different times. On the other hand, a white noise series is stationary — it does not matter when you observe it, it should look much the same at any point in time. Stationarity is desirable for many forecasting methods, such as ARIMA, because it makes the models simpler and more reliable.

Transformations such as logarithms can help to stabilize the variance of a time series. Differencing can help stabilise the mean of a time series by removing changes in the level of a time series, and therefore eliminating (or reducing) trend and seasonality. Differencing is a technique to transform a non-stationary time series into a stationary one by computing the differences between consecutive observations. More specific, it involves subtracting the current value of the series from the previous one, or from a lagged value. Let y_t be the value of the series at time t, and Δ be the differencing operator. In particular, for first-, second-, and third-order differencing we have:

- **First-order difference**:

$$\Delta y_t = y_t - y_{t-1} \tag{4.1}$$

- **Second-order difference**:

$$\begin{aligned} \Delta^2 y_t &= \Delta\left(\Delta y_t\right) \\ &= \Delta y_t - \Delta y_{t-1} \\ &= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) \\ &= y_t - 2y_{t-1} + y_{t-2} \end{aligned} \tag{4.2}$$

- **Third-order difference**:

$$\begin{aligned} \Delta^3 y_t &= \Delta\left(\Delta^2 y_t\right) \\ &= \Delta^2 y_t - \Delta^2 y_{t-1} \\ &= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) - [(y_{t-1} - y_{t-2}) - (y_{t-2} - y_{t-3})] \\ &= y_t - 3y_{t-1} + 3y_{t-2} - y_{t-3} \end{aligned} \tag{4.3}$$

By first-order differencing, we model the “changes” in the data, while second-order differencing would model the “change in the changes” of the original data. The order of differencing is the number of times the series is differenced to achieve stationarity. It is usually denoted by d in the ARIMA notation. Choosing the right order of differencing is important, because too much or too little differencing can affect the accuracy and validity of the forecasts when modeling a time-series. One way to choose the order of differencing is to start with $d = 0$ and increase it until the series becomes stationary, as indicated by the plots and tests. Another way is to use the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC) to compare different models with different orders of differencing and select the one with the lowest value. In practice, it is almost never necessary to go beyond second-order differences. Figure 4.1 shows the result of applying differencing on a synthetic time-series.

Stationarizing our data can also have some disadvantages for forecasting. First, it can introduce noise and randomness into the data, as it eliminates some of the information and structure of the original series. Second, it can reduce the sample size and the degrees of freedom of the data, as it discards some of the observations and parameters of the series. Third, it can distort the long-term trends and relationships of the data, as it focuses on the short-term fluctuations and differences of the series.

There are several ways to check if a time-series is stationary or not. One is to plot the series and look for visual clues, such as trends, cycles, or changes in variance. Another is to use statistical tests, such as the Augmented Dickey-Fuller (ADF) test, that compare the null hypothesis of non-stationarity with the alternative hypothesis of stationarity. A low p-value from the test indicates that the series is likely to be stationary.

After applying differencing, there might be scenarios where one would need to revert to the original series — be it for interpretation, validation, or any other purpose. This process of reverting to the original series from its differenced form is termed as the **inverse transformation**. More specific, for first-, second-, and third-order differencing we have:

- **First-order difference**:
  To find y_t, we simply add the difference at time t to the original value at time $t - 1$. This is akin to reversing the subtraction that was originally performed to get the difference.

$$y_t = y_{t-1} + \Delta y_t \tag{4.4}$$

- **Second-order difference**:
  First, we integrate (cumulatively sum) the second-order differences to retrieve the first-order differences. Then, we integrate the resulting first-order differences to get back to the original series.

$$\begin{aligned} \Delta y_t &= \Delta y_{t-1} + \Delta^2 y_t \\ y_t &= y_{t-1} + \Delta y_t \end{aligned} \tag{4.5}$$

- **Third-order difference**:
  First, we integrate the third-order differences to get the second-order differences. Then, we integrate the resulting second-order differences to retrieve the first-order differences. Finally, we integrate the first-order differences to revert to the original series.

$$\begin{aligned} \Delta^2 y_t &= \Delta^2 y_{t-1} + \Delta^3 y \\ \Delta y_t &= \Delta y_{t-1} + \Delta^2 y_t \\ y_t &= \Delta y_t + y_{t-1} \end{aligned} \tag{4.6}$$

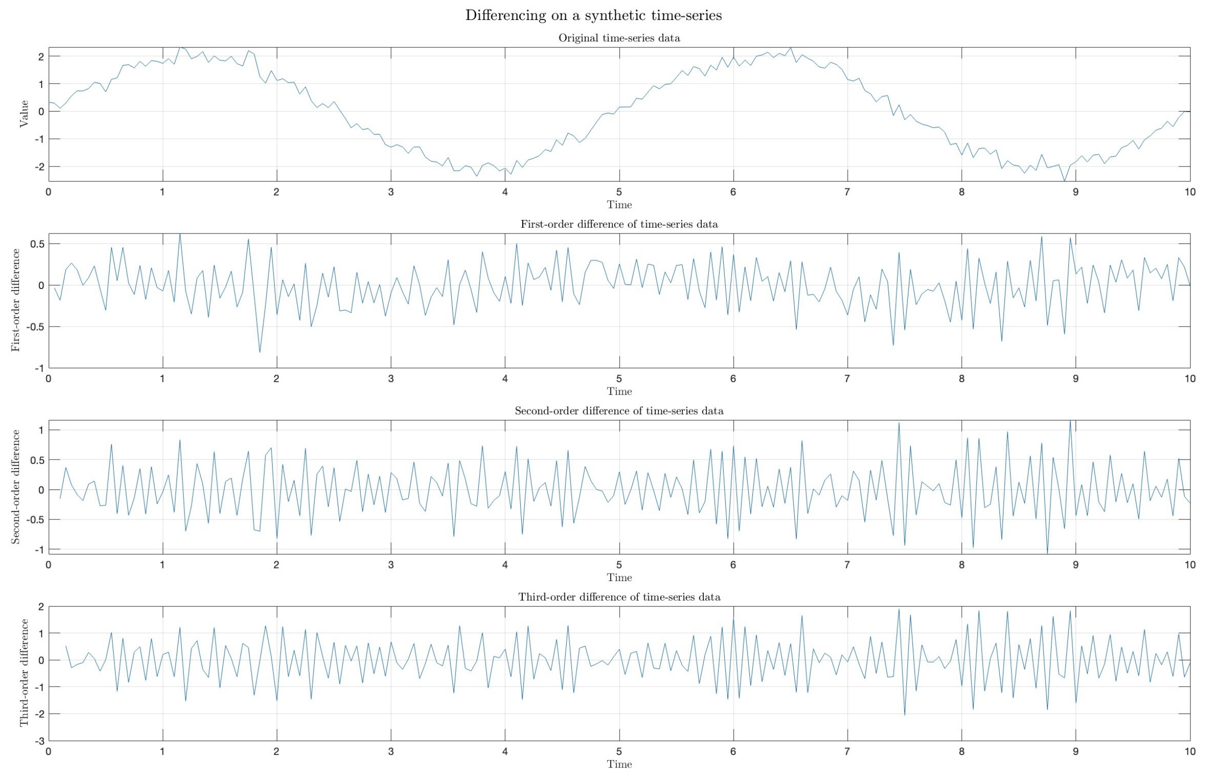


Figure 4.1: Differencing as a transformation on a synthetic time-series. The figure show the result after applying differencing up to three times.

4.2 Standardization

Standardization refers to the process of removing the mean and scaling to unit variance. This transformation is especially useful when the time series is expected to have a shifting mean, varying variance, or generally large scale. In addition, data standardization comes into the picture when features of the input data set have large differences between their ranges, or simply when they are measured in different units. These differences in the ranges of initial features cause trouble for many machine learning models. As far as the sunspots are concerned, the time-series has only one feature. Although, it is important to standardize the data when the modeling is going to be made by machine learning and deep learning methods.

Standardization typically means rescales data to have a mean of 0 and a standard deviation of 1 (unit variance). Let μ be the mean, and σ be the standard deviation of our original data. In order to standardize our data, we simply subtract μ from the data values and then divide by σ. More specific, we have:

- **Forward transformation**:

$$z = \frac{y - \mu}{\sigma} \tag{4.7}$$

- **Inverse transformation**:

$$y = z\sigma + \mu \tag{4.8}$$

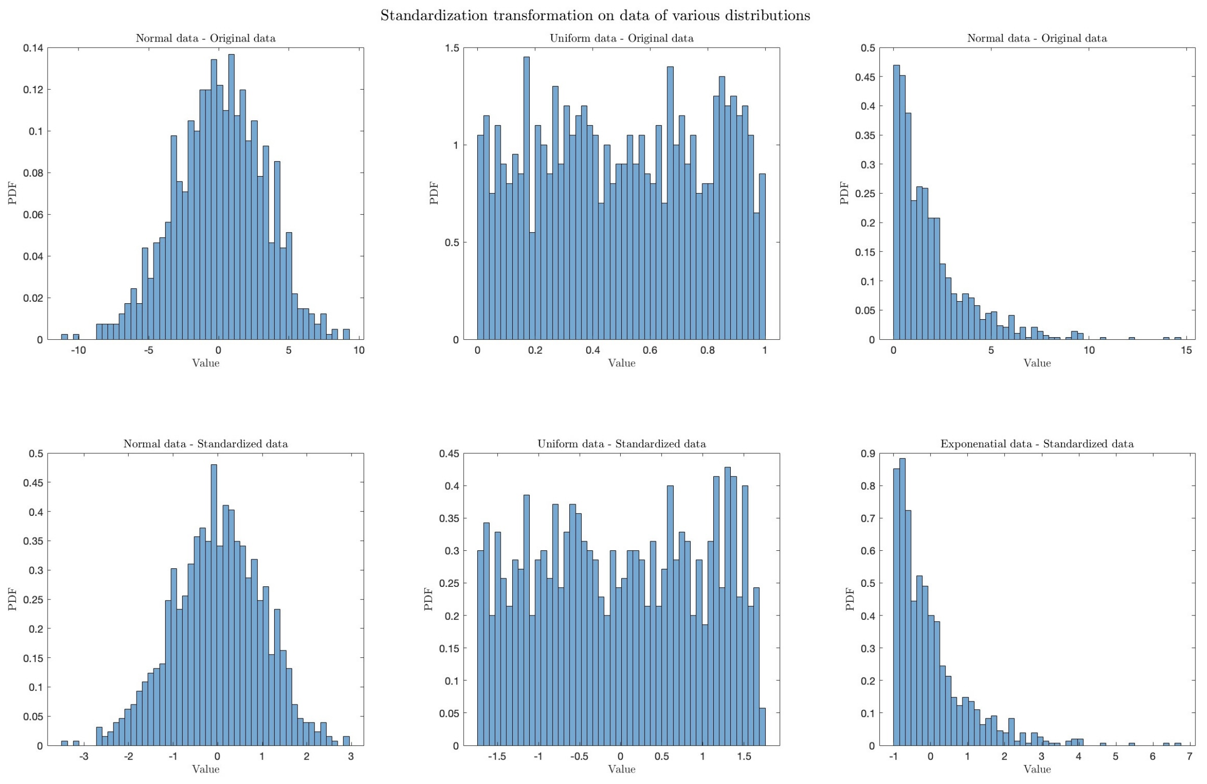


Figure 4.2: Standardization as a transformation on data of various distributions. Standardization transforms data to have a mean of zero and a variance of one.

Figure 4.2 shows the result of applying standardization on data of various distributions.

Overall, standardization is a pre-processing step commonly used in machine learning algorithms for several important reasons:

- **Scale invariance**: Machine learning algorithms often use distance-based metrics to make decisions or predictions. If the features in your dataset have different scales (i.e., some features are measured in large units, while others are in small units), the algorithm may give undue importance to features with larger scales. Standardizing the data by centering it around zero and scaling it to unit variance ensures that all features contribute more evenly to the model's performance, making the algorithm scale-invariant.

- **Faster convergence**: Many machine learning algorithms, particularly gradient-based optimization techniques (e.g., gradient descent), converge faster when the data is standardized. This is because the gradient updates are more balanced across features, preventing certain features from dominating the learning process and potentially leading to slower convergence or convergence to suboptimal solutions.

- **Improved model interpretability**: Standardizing the data allows for better interpretation of model coefficients or feature importances. When features are on different scales, it can be challenging to compare their relative importance in the model. Standardization ensures that the coefficients or feature importance scores represent the impact of a one-unit change in the respective feature, which makes them more interpretable.

- **Regularization effectiveness**: Regularization techniques, such as L1 (Lasso) and L2 (Ridge) regularization, penalize the magnitude of feature coefficients. Standardization ensures that the regularization penalty is applied fairly to all features, preventing some features from being overly penalized due to their scale.
- **Avoiding numerical instabilities**: In some algorithms, particularly those involving matrix operations, data with widely varying scales can lead to numerical instability issues. Standardizing the data can mitigate these problems and make the computations more stable.
- **Comparison across datasets**: Standardization makes it easier to compare and combine datasets that may have different units or scales. This is especially important in cases where data from multiple sources need to be merged for analysis, or when comparing models trained on different datasets.
- **Model performance**: Many machine learning models, such as support vector machines and k-means clustering, are sensitive to the scale of the input features. Standardizing the data often leads to better model performance, as it reduces the impact of scale-related biases.

4.3 Box-Cox Transformation

The Box-Cox transformation is a family of power transformations that are used to stabilize variance and make a dataset more closely follow a normal distribution. It is particularly useful for time-series data where the variance is not constant across levels (heteroscedasticity). The goal is to find an appropriate transformation that leads to data that meets the assumptions of homoscedasticity and normality.

4.3.1 Mathematical formulation

Given a time-series $y(t)$, we equivalently have a dataset of **strictly positive** scalars $Y = \{y_1, y_2, \ldots, y_n\}$. Henceforth, for simplicity, we will omit the variable t. The Box-Cox transformed series is defined as $z(t, \lambda)$, or equivalently $Z(\lambda) = \{z_1(\lambda), z_2(\lambda), \ldots, z_n(\lambda)\}$, and each element is given by

$$z(\lambda) = \begin{cases} \frac{y^\lambda - 1}{\lambda} & , \quad \text{if} \quad \lambda \neq 0 \\ \log(y) & , \quad \text{if} \quad \lambda = 0 \end{cases} \tag{4.9}$$

where λ is the transformation parameter. The transformation ensures that for any value of λ, the transformed data will be continuous and will not contain any discontinuities or abrupt shifts.

Now, we will refer to an extended form of the Box-Cox transformation, which could accommodate **negative or zero** values of y. That transformation is given by

$$z(\boldsymbol{\lambda}) = \begin{cases} \frac{(y+\lambda_2)^{\lambda_1} - 1}{\lambda_1} & , \quad \text{if} \quad \lambda_1 \neq 0 \\ \log(y + \lambda_2) & , \quad \text{if} \quad \lambda_1 = 0 \end{cases} \tag{4.10}$$

Here, $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)^T$. In practice, we could choose λ_2 such that $y + \lambda_2 > 0$ for any y. So, we could only view λ_1 as the model parameter. Consequently, the analysis below remains the same for both forms of Box-Cox transformation.

The inverse Box-Cox transformation is given by

$$y(\lambda) = \begin{cases} (\lambda z + 1)^{\frac{1}{\lambda}} & , \quad \text{if} \quad \lambda \neq 0 \\ \exp(z) & , \quad \text{if} \quad \lambda = 0 \end{cases} \tag{4.11}$$

As we will see below, for the estimation of parameter λ, we assume that z follows a normal distribution. This raises a theoretical problem in that y must be positive, which means that z must follow a truncated normal distribution conditioned on $\lambda z > -1$.

In the Box-Cox transformation, the choice of λ plays a critical role in determining the nature of the transformation applied to the data. When $\lambda = 1$, the transformation is essentially linear, resulting in $z = y - 1$. This means that the original values undergo a minor shift, but otherwise remain largely intact. For values of $0 < \lambda < 1$, the transformation tends to compress the larger values in the dataset while slightly expanding the smaller ones. As λ approaches 1 from values below it, the transformation's impact reduces and the data moves closer to its original shape. Conversely, as λ closes in on 0 from values greater than it, the transformation becomes progressively skewed. This kind of transformation can be especially useful when the data has a right-skewed distribution. A special case arises when $\lambda = 0$. Here, the Box-Cox transformation becomes a logarithmic one, represented as $z = \log(y)$. This transformation is particularly effective for data that exhibits heavy right skewness. It works by significantly compressing the higher values while expanding the smaller ones. When $-1 < \lambda < 0$, the transformation exerts a stronger compression effect on the larger values. This becomes even more pronounced as λ approaches 0 from negative values. On the flip side, as λ edges closer to -1, the transformation becomes more extreme, heavily penalizing larger values. This can be beneficial for stabilizing the variances in datasets that exhibit a marked decreasing exponential trend. As λ decreases further beyond -1, the transformation becomes increasingly severe. This range can yield highly transformed values that might significantly alter the intrinsic relationships in the original data, hence requiring cautious interpretation. Interestingly, as λ grows beyond 1, the transformation tends to spread out the larger values even more, potentially resulting in a more right-skewed distribution. For instance, with $\lambda = 2$, the transformation effectively squares each value. Consequently, higher values get dispersed further, whereas values between 0 and 1 get more compressed. As λ continues to rise, this effect intensifies, causing the larger values to stretch even more.

In practical data analysis, extreme λ values, especially much larger than 1, are employed sparingly because of the risk of disproportionately magnifying outliers. Nevertheless, in certain scenarios or datasets, enhancing the contrast or spread of the higher values might be advantageous. As always, the choice of λ should be informed by visualizations, statistical tests, and the overarching goals of the analysis.

Figure 4.3 shows the result of applying Box-Cox transformation on data of various distributions.

4.3.2 Tuning the parameter λ

Choosing the correct value for λ is essential for the transformation's effectiveness. The goal is to find an appropriate transformation (through a suitable λ) that makes the data as "normal" as possible. Two common approaches to find the optimal λ are the following:

- **Likelihood maximization**: Using a profile likelihood method, we can evaluate the likelihood of the transformed data for a range of λ values and choose the one that maximizes it.
- **Graphical methods**: Plot the transformed data for a variety of λ values and visually determine which one provides better results.

Maximum Likelihood Estimation (MLE) is a method used to estimate the parameters of a statistical model. In the context of the Box-Cox transformation, MLE can be employed to find the optimal value of the λ parameter that maximizes the likelihood of observing the given data. When using MLE with Box-Cox, the goal is typically to make the transformed data as "normal" as possible, since many statistical methods assume data normality. Next, we will see how MLE can be used for tuning the λ parameter in the Box-Cox transformation.

The likelihood of a statistical model is a measure of how well the model explains the observed data. Specifically, for a given set of parameters, the likelihood gives the probability (or probability density, for continuous data) of observing the given set of data. Suppose we have a statistical model defined by a probability distribution $f(\mathbf{y}|\theta)$, where $\mathbf{y}$ is the observed data and θ represents the parameters of the model. The likelihood of the parameters θ given the data $\mathbf{y}$ is denoted as $L(\theta|\mathbf{y})$ and is equal to the joint probability (or probability density) of the observed data given those parameters. For independent and identically distributed (i.i.d.) data points the likelihood is given by

$$L(\theta|\mathbf{y}) = f(\mathbf{y}|\theta) = f(y_1|\theta) \cdot f(y_2|\theta) \cdot \ldots \cdot f(y_n|\theta) = \prod_{i=1}^{n} f(y_i|\theta)$$

The main goal in many statistical problems is to find the parameter values that maximize this likelihood function, as these values are the ones that make the observed data most probable. This approach is called MLE. Most of the time we prefer maximizing the log-likelihood function, as it is mathematically more convenient. It is given by

$$\ell(\theta|\mathbf{y}) = \log\left(L(\theta|\mathbf{y})\right) = \log\left(\prod_{i=1}^{n} f(y_i|\theta)\right) = \sum_{i=1}^{n} \log\left(f(y_i|\theta)\right)$$

Once the data are transformed, we often assume that they follow a normal distribution. The normal probability density function is

$$f(z_i|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z_i - \mu)^2}{2\sigma^2}\right)$$

where μ and σ^2 are the mean and variance of the transformed data, respectively. The Box-Cox transformation changes the scale of our data, so when we compute the likelihood on the transformed scale, it is not directly comparable to the likelihood on the original scale. The Jacobian of the transformation corrects for this by giving us the factor by which we should adjust the likelihood on the transformed scale to make it comparable to the original scale. The Jacobian matrix is a matrix of all first-order partial derivatives of a vector-valued function. In essence, it provides a linear approximation for how small changes in input variables can influence the multiple outputs of a function. Using the Jacobian in the context of transformations involves calculating the determinant of the Jacobian matrix J and applying it to adjust probabilities, densities, or likelihoods. Since $z(\lambda)$ is a scalar function of a scalar variable, J reduces to a single element. Here, it is given by

$$J(y,\lambda) = \frac{dz(\lambda)}{dy} = y^{\lambda-1} \quad , \quad \lambda \in \mathbb{R}$$

So, for the adjusted probability density function we have

$$f_{adj}(y(\lambda)) = f(z(\lambda))|J(y,\lambda)| = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) y^{\lambda-1}$$

The probability density of $\mathbf{y}$, and also the likelihood of observing our i.i.d. data points $\mathbf{y}$ is the product of the individual densities

$$L_{adj}(\lambda,\mu,\sigma^2|\mathbf{y}) = f_{adj}(y_1|\lambda,\mu,\sigma^2)\cdot f_{adj}(y_2|\lambda,\mu,\sigma^2)\cdot\ldots\cdot f_{adj}(y_n|\lambda,\mu,\sigma^2) = \prod_{i=1}^{n} f_{adj}(y_i|\lambda,\mu,\sigma^2)$$

Substituting in our expression for $f_{adj}(y_i|\mu,\sigma^2)$ we get

$$L_{adj}(\lambda,\mu,\sigma^2|\mathbf{y}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i-\mu)^2}{2\sigma^2}\right) y_i^{\lambda-1}$$

The log-likelihood is simply the natural logarithm of the likelihood function. Taking the logarithm transforms the product of probabilities (in the likelihood) into a sum of logarithms, which is mathematically more tractable. Consequently, taking the logarithm we have

$$\begin{aligned}
\ell_{adj}(\lambda,\mu,\sigma^2|\mathbf{y}) &= \log\left(L_{adj}(\lambda,\mu,\sigma^2|\mathbf{y})\right) \\
&= \log\left(\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i-\mu)^2}{2\sigma^2}\right) y_i^{\lambda-1}\right) \\
&= \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i-\mu)^2}{2\sigma^2}\right) y_i^{\lambda-1}\right) \\
&= \sum_{i=1}^{n} \left[\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \log\left(\exp\left(-\frac{(y_i-\mu)^2}{2\sigma^2}\right)\right) + \log\left(y_i^{\lambda-1}\right)\right] \\
&= \sum_{i=1}^{n} \left[\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)\right] + \sum_{i=1}^{n} \left[\log\left(\exp\left(-\frac{(y_i-\mu)^2}{2\sigma^2}\right)\right)\right] + \sum_{i=1}^{n} \left[\log\left(y_i^{\lambda-1}\right)\right] \\
&= -\frac{n}{2}\log\left(2\pi\sigma^2\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu)^2 - (1-\lambda)\sum_{i=1}^{n}\log\left(y_i\right)
\end{aligned}$$

The negative log-likelihood function is given by

$$-\ell_{adj}(\lambda,\mu,\sigma^2|\mathbf{y}) = \frac{n}{2}\log\left(2\pi\sigma^2\right) + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu)^2 + (1-\lambda)\sum_{i=1}^{n}\log\left(y_i\right) \tag{4.12}$$

For the values of μ and σ^2, we use the sample mean and variance estimations of $\mathbf{z}$. To find the optimal λ, we would differentiate $-\ell_{adj}(\lambda, \mu, \sigma^2|\mathbf{y})$ with respect to λ and set it to zero, and then solve for λ. This will give us the maximum likelihood estimate of λ. The actual differentiation can be quite complex. In practice, numerical optimization methods are often used to find the λ value that maximizes the log-likelihood function.

The main objective of the **graphical methods** is to visually assess how different values of the λ parameter in the Box-Cox transformation affect the statistical properties of the time series data, such as linearity, normality, and homoscedasticity. Here, we will refer to some of the key graphical methods employed commonly. The first one is by using **Normal Probability Plot (Q-Q plot)**. This plot is used to visually assess if the data follows a normal distribution. The ordered data values are plotted against expected values from a standard normal distribution. If the data are normally distributed, the points will approximately lie on a straight line. Different λ values can be tested, and the one that results in the closest fit to a straight line is considered the best choice for achieving normality. The second approach is to visually assess the distribution of the data by plotting the **histogram** of the data. More specific, we transform the data with various λ values and plot the resulting histograms. The histogram shape can provide insights into skewness and kurtosis. A bell-shaped histogram indicates that the data is closer to a normal distribution.

4.4 κ-Logarithmic Transformation

The κ-Logarithmic transformation is a nonlinear, monotonic transformation from $\mathbb{R}_+ \to \mathbb{R}$. Therefore, it can be used like the Box-Cox transformation for the Gaussian anamorphosis of positive valued data.

4.4.1 Mathematical formulation

Given a time-series $y(t)$, we equivalently have a dataset of **strictly positive** scalars $Y = \{y_1, y_2, \ldots, y_n\}$. Henceforth, for simplicity, we will omit the variable t. The Box-Cox transformed series is defined as $z(t, \kappa)$, or equivalently $Z(\kappa) = \{z_1(\kappa), z_2(\kappa), \ldots, z_n(\kappa)\}$, and for each element is given by

$$z(\kappa) = \begin{cases} \frac{y^\kappa - y^{-\kappa}}{2\kappa} & , \quad \text{if} \quad \kappa \neq 0 \\ \log(y) & , \quad \text{if} \quad \kappa = 0 \end{cases} \tag{4.13}$$

where κ is the transformation parameter. The transformation ensures that for any value of κ, the transformed data will be continuous and will not contain any discontinuities or abrupt shifts.

The inverse κ-logarithmic transformation is given by

$$y(\kappa) = \begin{cases} (\sqrt{1 + \kappa^2 z^2} + \kappa z)^{\frac{1}{\kappa}} & , \quad \text{if} \quad \kappa \neq 0 \\ \exp(z) & , \quad \text{if} \quad \kappa = 0 \end{cases} \tag{4.14}$$

Figure 4.3 shows the result of applying κ-logarithmic transformation on data of various distributions.

4.4.2 Tuning the parameter κ

Once the data are transformed, we often assume that they follow a normal distribution. The normal probability density function is

$$f(z_i|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(z_i-\mu)^2}{2\sigma^2}\right)$$

where μ and σ^2 are the mean and variance of the transformed data, respectively. The Box-Cox transformation changes the scale of our data, so when you compute the likelihood on the transformed scale, it is not directly comparable to the likelihood on the original scale. The Jacobian of the transformation corrects for this by giving us the factor by which we should adjust the likelihood on the transformed scale to make it comparable to the original scale. The Jacobian matrix is a matrix of all first-order partial derivatives of a vector-valued function. In essence, it provides a linear approximation for how small changes in input variables can influence the multiple outputs of a function. Using the Jacobian in the context of transformations involves calculating the determinant of the Jacobian matrix J and applying it to adjust probabilities, densities, or likelihoods. Since $z(\kappa)$ is a scalar function of a scalar variable, J reduces to a single element. Here, it is given by

$$J(y,\kappa) = \frac{dz(\kappa)}{dy} = \frac{y^{\kappa-1}+y^{-\kappa-1}}{2} \quad , \quad \kappa \in \mathbb{R}$$

So, for the adjusted probability density function we have

$$f_{adj}(y(\kappa)) = f(z(\kappa))|J| = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right)\frac{y^{\kappa-1}+y^{-\kappa-1}}{2}$$

The probability density of $\mathbf{y}$, and also the likelihood of observing our i.i.d. data points $\mathbf{y}$ is the product of the individual densities

$$L_{adj}(\kappa,\mu,\sigma^2|\mathbf{y}) = f_{adj}(y_1|\mu,\sigma^2)\cdot f_{adj}(y_2|\mu,\sigma^2)\cdot\ldots\cdot f_{adj}(y_n|\mu,\sigma^2) = \prod_{i=1}^{n} f_{adj}(y_i|\mu,\sigma^2)$$

Substituting in our expression for $f_{adj}(y_i|\mu,\sigma^2)$ we get

$$L_{adj}(\kappa,\mu,\sigma^2|\mathbf{y}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i-\mu)^2}{2\sigma^2}\right)\frac{y_i^{\kappa-1}+y_i^{-\kappa-1}}{2}$$

The log-likelihood is simply the natural logarithm of the likelihood function. Taking the logarithm transforms the product of probabilities (in the likelihood) into a sum of logarithms, which is mathematically more tractable. Consequently, taking the logarithm we have

$$\begin{aligned}
\ell_{adj}(\kappa,\mu,\sigma^2|\mathbf{y}) &= \log\left(L_{adj}(\kappa,\mu,\sigma^2|\mathbf{y})\right)\\
&= \log\left(\prod_{i=1}^{n}\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_i-\mu)^2}{2\sigma^2}\right)\frac{y_i^{\kappa-1}+y_i^{-\kappa-1}}{2}\right)\\
&= \sum_{i=1}^{n}\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_i-\mu)^2}{2\sigma^2}\right)\frac{y_i^{\kappa-1}+y_i^{-\kappa-1}}{2}\right)\\
&= \sum_{i=1}^{n}\left[\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)+\log\left(\exp\left(-\frac{(y_i-\mu)^2}{2\sigma^2}\right)\right)+\log\left(\frac{y_i^{\kappa-1}+y_i^{-\kappa-1}}{2}\right)\right]\\
&= \sum_{i=1}^{n}\left[\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)\right]\\
&\quad+\sum_{i=1}^{n}\left[\log\left(\exp\left(-\frac{(y_i-\mu)^2}{2\sigma^2}\right)\right)\right]\\
&\quad+\sum_{i=1}^{n}\left[\log\left(\frac{y_i^{\kappa-1}+y_i^{-\kappa-1}}{2}\right)\right]\\
&\quad= -\frac{n}{2}\log\left(2\pi\sigma^2\right)-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu)^2+\sum_{i=1}^{n}\left[\log\left(\frac{y_i^{\kappa-1}+y_i^{-\kappa-1}}{2}\right)\right]
\end{aligned}$$

The negative log-likelihood function is given by

$$-\ell_{adj}(\kappa,\mu,\sigma^2|\mathbf{y}) = \frac{n}{2}\log\left(2\pi\sigma^2\right)+\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu)^2-\sum_{i=1}^{n}\left[\log\left(\frac{y_i^{\kappa-1}+y_i^{-\kappa-1}}{2}\right)\right] \tag{4.15}$$

For the values of μ and σ^2, we use the sample mean and variance estimations of $\mathbf{z}$. To find the optimal κ, we would differentiate $-\ell_{adj}(\kappa,\mu,\sigma^2|\mathbf{y})$ with respect to λ and set it to zero, and then solve for λ. This will give us the minimum negative log-likelihood estimate of κ. The actual differentiation can be quite complex. In practice, numerical optimization methods are often used to find the λ value that minimizes the negative log-likelihood function, or equivalently maximizes the log-likelihood function.

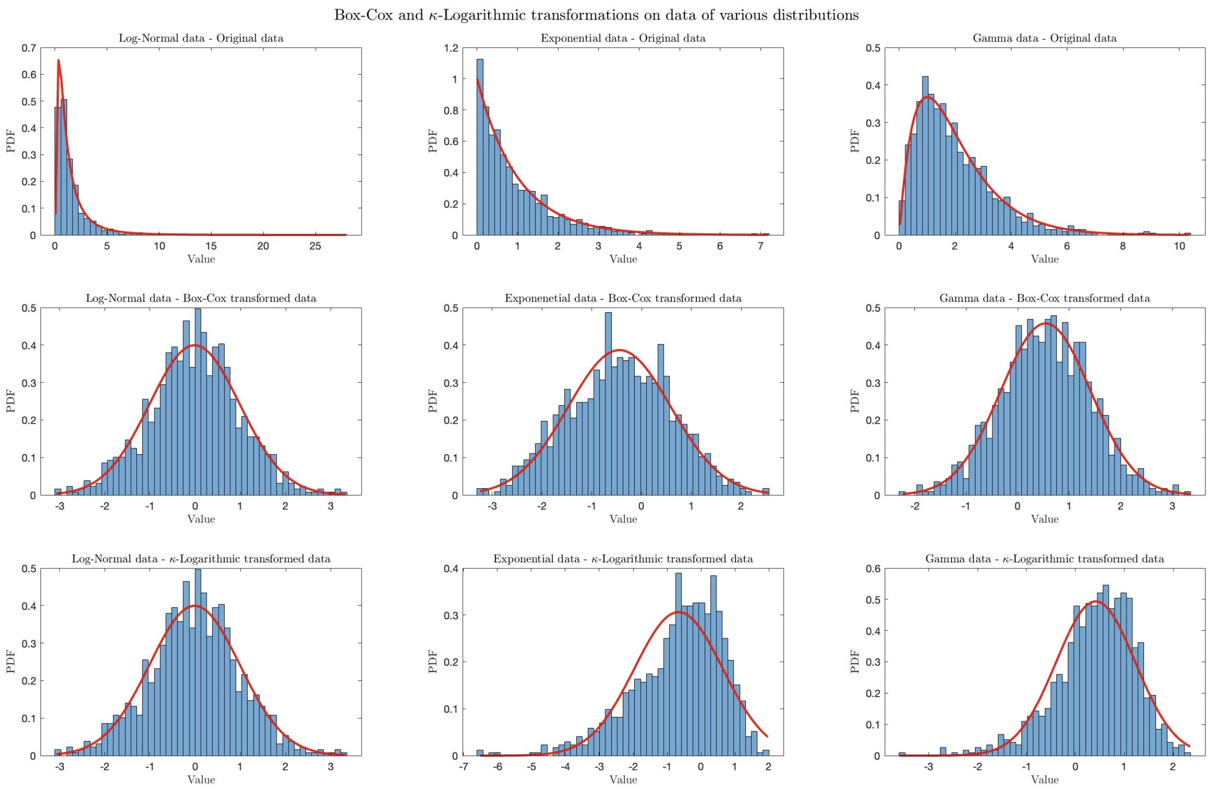


Figure 4.3: Box-Cox and κ-Logarithmic transformations on data of various distributions. The objective of both transformations is for the transformed data to conform to a normal distribution. The first row shows the histograms of the original data for various distributions. The PDFs are drawn with red. The second row shows the histograms of the Box-Cox transformed data in each case. The PDFs of the corresponding normal distributions are drawn with red. The third row shows histograms of the κ-Logarithmic transformed data on each case. The PDFs of the corresponding normal distributions are drawn with red.

5 Gaussian Processes

In the dynamic landscape of machine learning and statistical modeling, Gaussian processes (GPs) have emerged as a powerful paradigm that bridges the gap between data-driven predictions and uncertainty quantification. Rooted in the principles of probabilistic supervised learning, GPs offer an elegant framework for handling regression and classification tasks. Among its various applications, Gaussian process regression shines as a prominent technique that not only predicts target values but also provides estimates of the associated uncertainties.

At its core, a Gaussian process is more than just a predictive model; it is a distribution over functions. Unlike traditional point estimates, which provide single predictions for each input, GPs yield entire distributions of possible functions that could explain the underlying data. This not only furnishes predictions but also offers a measure of confidence or uncertainty associated with those predictions. Such a probabilistic perspective proves invaluable when dealing with noisy, limited, or irregularly sampled data, where quantifying uncertainty becomes paramount.

The terminology "Gaussian process" might evoke a connection to the Gaussian distribution. Indeed, a Gaussian process is a natural extension of the multivariate Gaussian distribution to an infinite-dimensional space of functions. This extension allows GPs to capture complex patterns and non-linear relationships within data, making them adept at handling diverse real-world problems.

In this exploration of Gaussian processes, our focus is on Gaussian process regression (GPR). This specialized application involves inferring the underlying relationship between input features and target variables, enabling us to make informed predictions about new, unseen instances. GPR transcends traditional regression by not only estimating a mean prediction, but also a covariance structure that characterizes the uncertainty associated with the prediction. Consequently, GPR is particularly useful in scenarios where decisions are made based on both accurate predictions and a keen awareness of the inherent unpredictability in data.

Throughout this discussion, we will delve into the foundations of Gaussian processes, demystifying concepts such as kernels, covariance functions, and hyperparameters. Furthermore, we will dissect the intricacies of Gaussian process regression and analyze the basic theoretical background in an attempt to showcase its application in various fields.

5.1 The Gaussian distribution

5.1.1 Univariate Gaussian distribution

A random variable X is Gaussian, and it follows a univariate normal distribution (UVN) with mean $\mu = \mathbb{E}[X] \in \mathbb{R}$ and variance $\sigma^2 = \mathrm{Var}(X) \in \mathbb{R}$, if its probability density function (PDF) is [42]

$$P_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{5.1}$$

Here, X represent the random variable and $x \in \mathbb{R}$ is the real argument. The normal distribution of X is usually represented by $X \sim \mathcal{N}(\mu, \sigma^2)$.

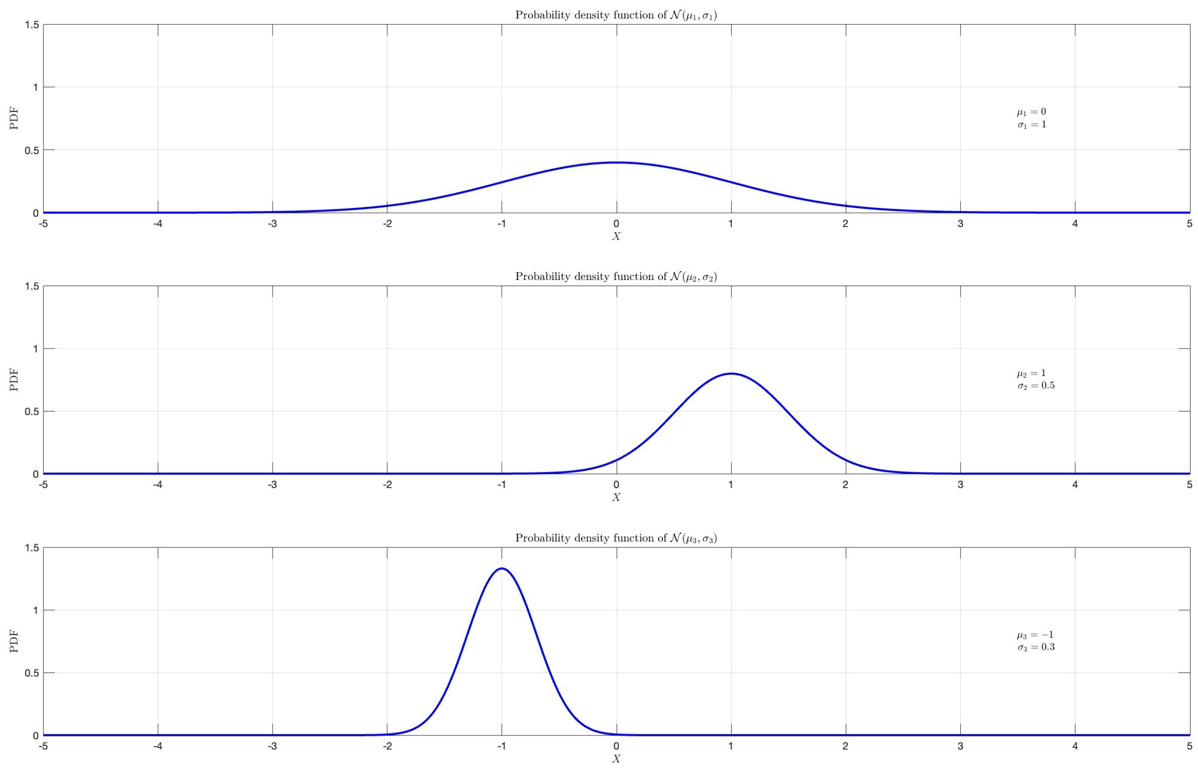


Figure 5.1: Illustrations of the probability density function for various univariate normal distributions.

5.1.2 Multivariate Gaussian distribution

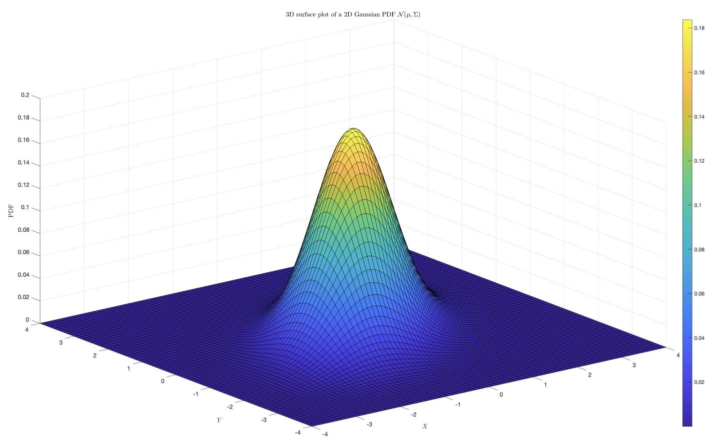
The multivariate normal distribution (MVN), or joint normal distribution, is a generalization of the univariate normal distribution to higher dimensions. A random variable is said to be D-variate normally distributed if every linear combination of its D components have a univariate normal distribution. Mathematically, $\mathbf{X} = (X_1, X_2 \ldots, X_D)^T$ has a multivariate Gaussian distribution if $Y = \sum_{i=1}^{D} \alpha_i X_i$ is normally distributed for any constant vector $\boldsymbol{\alpha} \in \mathbb{R}^D$. Note that if all D components are independent Gaussian random variables, then $\mathbf{X}$ must be multivariate Gaussian, because the sum of independent Gaussian random variables is always Gaussian. The PDF of a MVN with D dimensions is defined as [42]

$$P_{\mathbf{X}}(\mathbf{x}) = \frac{1}{(2\pi)^{D/2}[\det(\mathbf{\Sigma})]^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \tag{5.2}$$

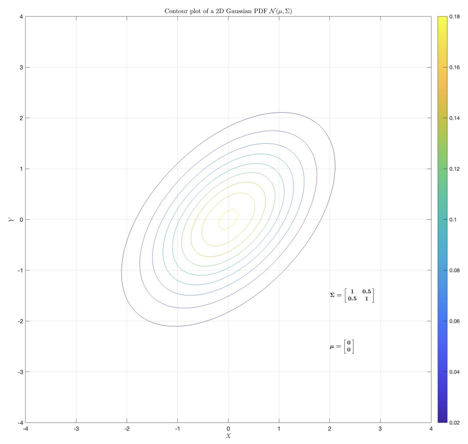
where D is the number of the dimension, $\mathbf{X}$ represent the random vector, $\mathbf{x} \in \mathbb{R}^D$ represents the real argument, $\boldsymbol{\mu} = \mathbb{E}[\mathbf{X}] \in \mathbb{R}^D$ is the mean vector, and $\mathbf{\Sigma} = \text{Cov}(\mathbf{X}, \mathbf{X}) \in \mathbb{R}^{D\times D}$ is the covariance matrix. $\mathbf{\Sigma}$ is a symmetric matrix that stores the pairwise covariance of all jointly modeled random variables, X_i, X_j, with $\Sigma_{ij} = \text{Cov}(X_i, X_j) \in \mathbb{R}$ as its (i, j) element.

5.1.3 Affine transformation

If $\mathbf{Y} = \mathbf{c} + \mathbf{B}\mathbf{X}$ is an affine transformation of $\mathbf{X} \ \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$, where $\mathbf{c}$ is an $M \times 1$ vector of constants and $\mathbf{B}$ is a constant $M \times N$ matrix, then $\mathbf{Y}$ has a multivariate normal



(a) Probability density function of a bivariate normal distribution.



(b) Contour plot of a bivariate normal distribution.

Figure 5.2: Illustrations of a multivariate (bivariate) normal distribution.

distribution with expected value $\mathbf{c} + \mathbf{B}\boldsymbol{\mu}$ and variance $\mathbf{B\Sigma B}^T$ [43], i.e.,

$$\mathbf{Y} \sim \mathcal{N}\left(\mathbf{c} + \mathbf{B}\boldsymbol{\mu}, \mathbf{B\Sigma B}^{\mathrm{T}}\right) \tag{5.3}$$

In particular, any subset of the X_i has a marginal distribution that is also multivariate normal. To see this, consider the following example: to extract the subset $(X_1, X_2, X_4)^T$, use

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & 0 & \dots & 0 \end{bmatrix}$$

which extracts the desired elements directly.

5.1.4 Bayes' theorem and the two rules of probability

Let X and Y be two continuous random variables. The **Bayes' theorem** is stated mathematically as the following equation

$$p_{X|Y}(x|y) = \frac{p_{Y|X}(y|x)p_X(x)}{p_Y(y)} = \frac{p_{X,Y}(x,y)}{p_Y(y)} \tag{5.4}$$

The **sum rule** states that [44]

$$p_X(x) = \int_Y p_{X,Y}(x,y)dy \tag{5.5}$$

and the **product rule** states that [44]

$$p_{X,Y}(x,y) = p_{X|Y}(x|y)p_Y(y) = p_{Y|X}(y|x)p_X(x) \tag{5.6}$$

In the remainder, we will see that a joint Gaussian distribution can be factorized into a conditional Gaussian and a marginal Gaussian distribution.

5.1.5 Marginalization

Given two continuous random variables X and Y whose joint distribution is known, then the marginal probability density function can be obtained by integrating the joint probability distribution, $p_{X,Y}(x,y)$, over Y, and vice versa. For the first case, that is

$$p_X(x) = \int_Y p_{X,Y}(x,y)dy = \int_Y p_{X|Y}(x|y)p_Y(y)dy \tag{5.7}$$

The first property of the "Gaussians", states that if we marginalize out variables in a multivariate Gaussian distribution, the result is still a Gaussian distribution. The Gaussian distribution is thus closed under marginalization. In Gaussian distributions, as stated above, we can perform the marginalization by applying the appropriate affine transformation on the multivariate random variable.

Let $\mathbf{X} = (X_1, X_2, \ldots, X_n)^T$ be a n-dimensional vector and $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_m)^T$ a m-dimensional vector, which both are jointly Gaussian distributed with covariance matrix $\mathbf{\Sigma} \in \mathbb{R}^{(n+m)\times(n+m)}$. Note that we can write $\mathbf{\Sigma}$ as a block matrix, i.e.,

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{\Sigma_{XX}} & \mathbf{\Sigma_{XY}} \\ \mathbf{\Sigma_{YY}} & \mathbf{\Sigma_{YX}} \end{bmatrix}$$

where $\mathbf{\Sigma_{XX}}$ and $\mathbf{\Sigma_{YY}}$ are the covariance matrices of $\mathbf{X}$ and $\mathbf{Y}$, respectively, and $\mathbf{\Sigma_{XY}} = (\mathbf{\Sigma_{YX}})^T$ gives the covariance between $\mathbf{X}$ and $\mathbf{Y}$. We have that [45]

$$\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{X}, \mathbf{\Sigma_{XX}}) \tag{5.8}$$

$$\mathbf{Y} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{Y}, \mathbf{\Sigma_{YY}})$$

5.1.6 Conditioning

Rather than disregarding information, which was the case when calculating marginal distributions earlier, our focus now shifts to integrating the information we possess regarding the other random variable, denoted as Y. Conditioning, in this context, signifies a process of acquiring knowledge: it pertains to understanding how our awareness that $Y = y$ influences our understanding of variable X. Conditioning essentially involves examining how the distribution of the variable X alters when the variable Y attains a specific value, denoted as y. Mathematically, we have

$$p_{\mathbf{X}|\mathbf{Y}}(\mathbf{x}|\mathbf{y}) = \frac{p_{\mathbf{X},\mathbf{Y}}(\mathbf{x},\mathbf{y})}{p_\mathbf{Y}(\mathbf{y})} \tag{5.9}$$

$$p_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) = \frac{p_{\mathbf{X},\mathbf{Y}}(\mathbf{x},\mathbf{y})}{p_\mathbf{X}(\mathbf{x})}$$

Note that Gaussian distributions are closed under conditioning. This fact means that, if we start with a Gaussian distribution and update our knowledge given the observed value of one of its components, then the resulting distribution is still Gaussian.

Let $\mathbf{X} = (X_1, X_2, \ldots, X_n)^T$ be a n-dimensional vector and $\mathbf{Y} = (Y_1, Y_2, \ldots, Y_m)^T$ a m-dimensional vector, which both are jointly Gaussian distributed with covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{(n+m)\times(n+m)}$. Note that we can write $\boldsymbol{\Sigma}$ as a block matrix, i.e.,

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{\mathbf{XX}} & \boldsymbol{\Sigma}_{\mathbf{XY}} \\ \boldsymbol{\Sigma}_{\mathbf{YY}} & \boldsymbol{\Sigma}_{\mathbf{YX}} \end{bmatrix}$$

where $\boldsymbol{\Sigma}_{\mathbf{XX}}$ and $\boldsymbol{\Sigma}_{\mathbf{YY}}$ are the covariance matrices of $\mathbf{X}$ and $\mathbf{Y}$, respectively, and $\boldsymbol{\Sigma}_{\mathbf{XY}} = (\boldsymbol{\Sigma}_{\mathbf{YX}})^T$ gives the covariance between $\mathbf{X}$ and $\mathbf{Y}$. We have that [45]

$$\mathbf{X}|\mathbf{Y} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{X}} + \Sigma_{\mathbf{XY}}\Sigma_{\mathbf{YY}}^{-1}(\mathbf{Y} - \boldsymbol{\mu}_{\mathbf{Y}}), \boldsymbol{\Sigma}_{\mathbf{XX}} - \Sigma_{\mathbf{XY}}\Sigma_{\mathbf{YY}}^{-1}\Sigma_{\mathbf{YX}}) \tag{5.10}$$

$$\mathbf{Y}|\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{Y}} + \Sigma_{\mathbf{YX}}\Sigma_{\mathbf{XX}}^{-1}(\mathbf{X} - \boldsymbol{\mu}_{\mathbf{X}}), \boldsymbol{\Sigma}_{\mathbf{YY}} - \Sigma_{\mathbf{YX}}\Sigma_{\mathbf{XX}}^{-1}\Sigma_{\mathbf{XY}})$$

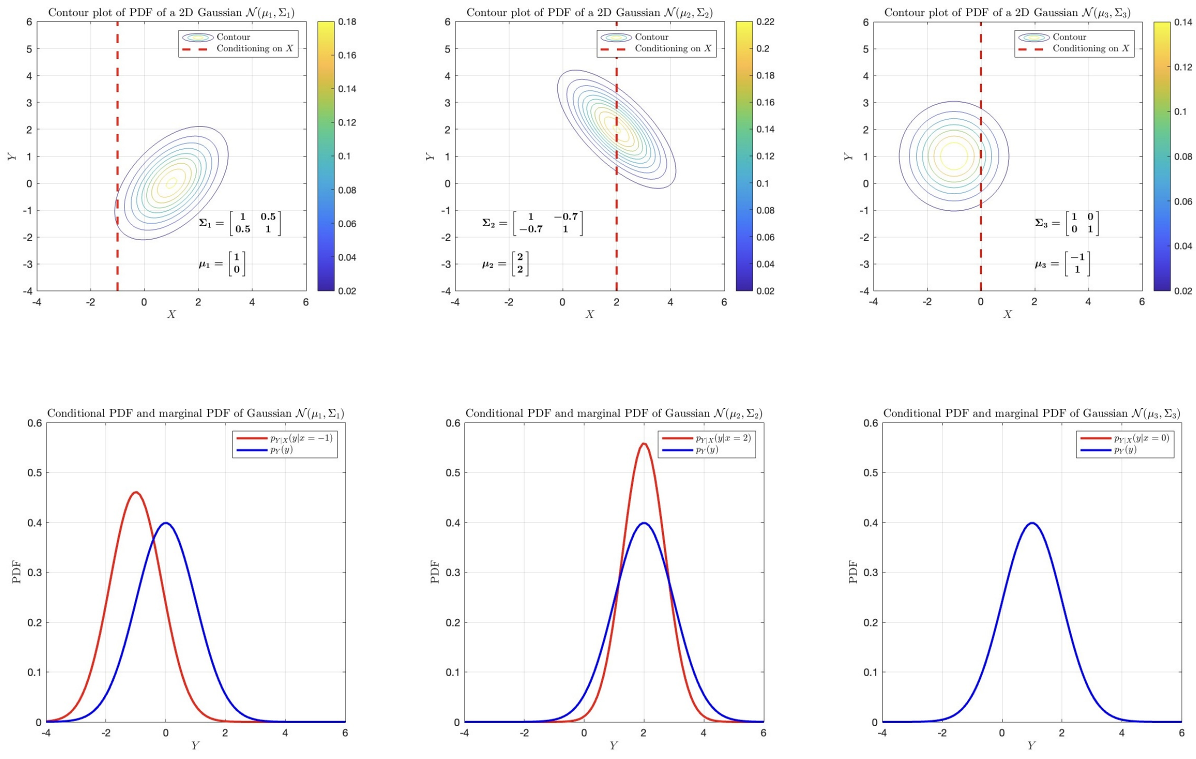


Figure 5.3: Illustrations of marginalization and conditioning in multivariate normal distributions. The top row showcases contour plots representing various 2D Gaussian distributions. In contrast, the bottom row illustrates the conditional and marginal distributions corresponding to the Gaussian distributions depicted above. It is noteworthy to observe the bottom-right plot, where the marginal and conditional probability density functions coincide. This equivalence arises due to a correlation of 0 between the variables X and Y, implying that the value of X provides no information about the value of Y.

5.1.7 Gaussian Process (GP)

We use a Gaussian process (GP) to describe a distribution over functions. Formally, a Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution [4]. A Gaussian process is completely specified by its mean function and covariance function. We define the mean function $m(\mathbf{x})$ and the covariance function $k(\mathbf{x}, \mathbf{x}')$ of a real process $f(\mathbf{x})$ as [4]

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})] \tag{5.11}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))] \tag{5.12}$$

and will write the Gaussian process as [4]

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{5.13}$$

In our case, the random variables represent the value of the function $f(\mathbf{x})$ at location $\mathbf{x}$. Often, Gaussian processes are defined over time, i.e. where the index set of the random variables is time. Here the index set $\mathcal{X}$ is the set of possible inputs, which could be more general, e.g. $\mathbb{R}^D$. For notational convenience, we use the (arbitrary) enumeration of the cases in the training set to identify the random variables such that $f_i \triangleq f(\mathbf{x}_i)$ is the random variable corresponding to the case $(\mathbf{x}_i, y_i)$ as would be expected [4].

From the definition of Gaussian processes we know that, for any finite set of input $\{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N\}$, we have the following multivariate normal distribution

$$\mathbf{f} \equiv (f_i, f_2, \ldots f_N)^T \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}) \tag{5.14}$$

where

$$\mu_i = m(\mathbf{x}_i) \quad , \quad i = 1, 2, \ldots, N$$

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) \quad , \quad i, j = 1, 2, \ldots, N$$

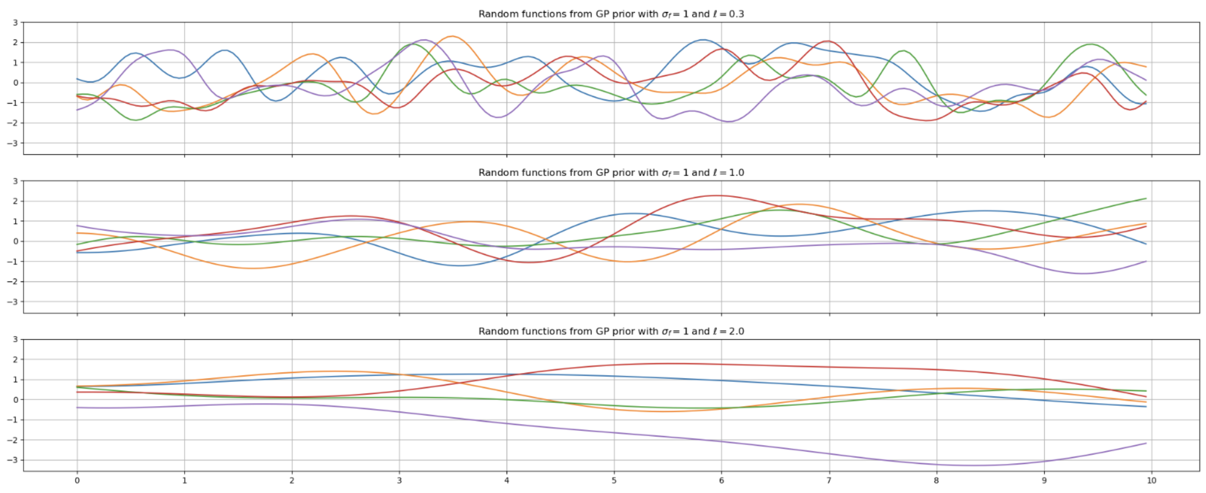


Figure 5.4: Random functions drawn from a Gaussian process $\mathcal{GP}(m(x), k(x, x'))$, where $m(x)$ is the zero-mean function and the covariance function $k(x, x')$ is the squared exponential kernel.

5.2 Gaussian Process Regression

Gaussian Process Regression (GPR) is a non-parametric probabilistic approach to obtain a function f, that is distributed according to a Gaussian Process (GP) and able to describe our underlying data. The entire function evaluation is drawn from a multivariate normal distribution.

The GP can be utilized as a prior probability distribution in Bayesian inference, which allows for function regression. Following the Bayesian methodology, new information is combined with existing information. Using Bayes' theorem, the prior is combined with new data to obtain a posterior distribution [46].

5.2.1 Parametric vs. non-parametric models

Parametric models necessitate the a priori specification of a set of parameters that define the underlying data distribution. This predefined structure empowers parametric models to make predictions based on a fixed number of parameters, regardless of the size of the training dataset. Notable examples of parametric models encompass linear regression, Lasso regression, Ridge regression, and logistic regression, among others [3].

On the other hand, the construction of **non-parametric models** does not entail explicit assumptions about the functional form, as seen in the case of parametric models like linear regression. Instead, non-parametric models can be viewed as approximations that closely adhere to the data points. These models refrain from relying on pre-established parameter configurations, thereby enabling them to adapt to intricate and irregular patterns present in the data. The advantage of non-parametric approaches lies in their ability to eschew assumptions about a specific functional form, such as linear modeling, allowing them to effectively capture a broader array of potential shapes for the actual or true function [3].

Aspect	**Parametric Models**	**Non-Parametric Models**
Definition	Require predefined parameter settings	Do not rely on specific parameters
Flexibility	Less flexible, assume fixed data structure	Highly flexible, adapt to complex patterns
Assumptions	Make strong assumptions about data distribution	Fewer assumptions about data distribution
Complexity	Simpler model structure	More complex model structure
Interpretability	More interpretable	Less interpretable
Performance	Efficient for large datasets with limited features	Perform well with high-dimensional data

Table 5.1: Comparison between parametric and non-parametric models [3]

5.2.2 Training set

We have a training set $\mathcal{D}$ of N observations, $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, N\}$, where $\mathbf{x}$ denotes an input vector of dimension D (independent variable) and y denotes a scalar output or target (dependent variable). The column vector inputs for all N cases are aggregated in the $D \times N$ design matrix $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)$, and the targets are collected in the vector $\mathbf{y} = (y_1, y_2, \dots, y_N)^T$, so we can write $\mathcal{D} = (\mathbf{X}, \mathbf{y})$ [4]. Within the context of regression, the target variables are continuous real values. Our primary objective revolves around drawing conclusions about the connection between the input variables and these target values, specifically, examining the conditional distribution of the targets concerning the inputs. Note that we are not interested in modelling the input distribution itself.

5.2.3 Noisy observations

We have assumed that any observed values y at input $\mathbf{x}$, i.e. $y(\mathbf{x})$, differ from the function value $f(\mathbf{x})$ by additive noise, and we will further assume that this noise follows an independent, identically distributed (i.i.d.) Gaussian distribution with zero mean and variance σ_n^2. It is [4]

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2)$$

and consequently

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_n^2 \mathbf{I}) \tag{5.15}$$

Finally, we have

$$y(\mathbf{x}) = f(\mathbf{x}) + \epsilon \tag{5.16}$$

5.2.4 Prior predictive distribution

Consider a finite training set $\mathcal{D}$ and a Gaussian process $f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$, which is the GP prior, where $m(\mathbf{x})$ is the prior mean function and $k(\mathbf{x}, \mathbf{x}')$ is the prior covariance (kernel) function. That implies that

$$\mathbf{f}|\mathbf{X} \sim \mathcal{N}(m(\mathbf{X}), K(\mathbf{X}, \mathbf{X})) = p(\mathbf{f}|\mathbf{X}) \tag{5.17}$$

where $\mathbf{f}$ is a vector with the function evaluation at all input vectors, i.e $\mathbf{X}$.

5.2.5 Likelihood

Given that observations $\mathbf{y}$ are equal to the function evaluations $\mathbf{f}$ with additive i.i.d. zero mean Gaussian noise, it is

$$\mathbb{E}(\mathbf{y}) = \mathbb{E}(\mathbf{f}) = m(\mathbf{X})$$

$$\mathrm{Cov}(\mathbf{f}, \mathbf{f}) = K(\mathbf{X}, \mathbf{X})$$

$$\mathrm{Cov}\,(\boldsymbol{\epsilon}, \boldsymbol{\epsilon}) = \sigma_n^2 \mathbf{I}$$

$$\mathrm{Cov}\,(\boldsymbol{\epsilon}, \mathbf{f}) = \mathrm{Cov}\,(\mathbf{f}, \boldsymbol{\epsilon}) = \mathbf{0}$$

$$\begin{aligned}\mathrm{Cov}(\mathbf{f},\mathbf{y}) = \mathrm{Cov}(\mathbf{y},\mathbf{f}) &= \mathrm{Cov}(\mathbf{f}+\boldsymbol{\epsilon},\mathbf{f})\\ &= \mathrm{Cov}(\mathbf{f},\mathbf{f}) + \mathrm{Cov}(\boldsymbol{\epsilon},\mathbf{f})\\ &= K(\mathbf{X},\mathbf{X}) + \mathbf{0}\\ &= K(\mathbf{X},\mathbf{X})\end{aligned}$$

$$\begin{aligned}\mathrm{Cov}(\mathbf{y},\mathbf{y}) &= \mathrm{Cov}(\mathbf{f}+\boldsymbol{\epsilon},\mathbf{f}+\boldsymbol{\epsilon})\\ &= \mathrm{Cov}(\mathbf{f},\mathbf{f}) + \mathrm{Cov}(\boldsymbol{\epsilon},\boldsymbol{\epsilon}) + 2\cdot\mathrm{Cov}(\boldsymbol{\epsilon},\mathbf{f})\\ &= K(\mathbf{X},\mathbf{X}) + \sigma_n^2\mathbf{I} + 2\cdot\mathbf{0}\\ &= K(\mathbf{X},\mathbf{X}) + \sigma_n^2\mathbf{I}\end{aligned}$$

We have the following joint Gaussian distribution of $\mathbf{y}$ and $\mathbf{f}$ given $\mathbf{X}$

$$\begin{bmatrix}\mathbf{y}\\ \mathbf{f}\end{bmatrix}\Bigg|\mathbf{X} \sim \mathcal{N}\left(\begin{bmatrix}m(\mathbf{X})\\ m(\mathbf{X})\end{bmatrix}, \begin{bmatrix}K(\mathbf{X},\mathbf{X})+\sigma_n^2\mathbf{I} & K(\mathbf{X},\mathbf{X})\\ K(\mathbf{X},\mathbf{X}) & K(\mathbf{X},\mathbf{X})\end{bmatrix}\right) = p\left(\begin{bmatrix}\mathbf{y}\\ \mathbf{f}\end{bmatrix}\Bigg|\mathbf{X}\right) \tag{5.18}$$

The conditional distribution $\mathbf{y}|\mathbf{f},\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{y}|\mathbf{f}}, \boldsymbol{\Sigma}_{\mathbf{y}|\mathbf{f}})$ is normal with

$$\boldsymbol{\mu}_{\mathbf{y}|\mathbf{f},\mathbf{X}} = m(\mathbf{X}) + K(\mathbf{X},\mathbf{X})\left[K(\mathbf{X},\mathbf{X})\right]^{-1}(\mathbf{f} - m(\mathbf{X})) = \mathbf{f} \tag{5.19}$$

$$\boldsymbol{\Sigma}_{\mathbf{y}|\mathbf{f},\mathbf{X}} = K(\mathbf{X},\mathbf{X}) + \sigma_n^2\mathbf{I} - K(\mathbf{X},\mathbf{X})\left[K(\mathbf{X},\mathbf{X})\right]^{-1}K(\mathbf{X},\mathbf{X})) = \sigma_n^2\mathbf{I} \tag{5.20}$$

So, we have that [4]

$$\mathbf{y}|\mathbf{f},\mathbf{X} \sim \mathcal{N}(\mathbf{f},\sigma_n^2\mathbf{I}) = p(\mathbf{y}|\mathbf{f},\mathbf{X}) \tag{5.21}$$

In the same way, we can derive the following distribution

$$\mathbf{f}|\mathbf{y},\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{f}|\mathbf{y},\mathbf{X}}, \boldsymbol{\Sigma}_{\mathbf{f}|\mathbf{y},\mathbf{X}}) = p(\mathbf{f}|\mathbf{y},\mathbf{X}) \tag{5.22}$$

where the mean function is

$$\boldsymbol{\mu}_{\mathbf{f}|\mathbf{y},\mathbf{X}} = m(\mathbf{X}) + K(\mathbf{X},\mathbf{X})\left[K(\mathbf{X},\mathbf{X}) + \sigma_n^2\mathbf{I}\right]^{-1}(\mathbf{y} - m(\mathbf{X})) \tag{5.23}$$

and the covariance function is

$$\boldsymbol{\Sigma}_{\mathbf{f}|\mathbf{y},\mathbf{X}} = K(\mathbf{X},\mathbf{X}) - K(\mathbf{X},\mathbf{X})\left[K(\mathbf{X},\mathbf{X}) + \sigma_n^2\mathbf{I}\right]^{-1}K(\mathbf{X},\mathbf{X}) \tag{5.24}$$

5.2.6 Marginal likelihood

Now, we will introduce the marginal likelihood (or evidence), which is the integral of the likelihood times the prior. The term marginal likelihood refers to the marginalization over the function values $\mathbf{f}$.

$$\begin{aligned}p(\mathbf{y}|\mathbf{X}) &= \int p(\mathbf{y}|\mathbf{f},\mathbf{X})p(\mathbf{f}|\mathbf{X})d\mathbf{f}\\ &= \int \mathcal{N}(\mathbf{f},\sigma_n^2\mathbf{I})\mathcal{N}(\mathbf{m},\mathbf{K})d\mathbf{f}\\ &= \mathcal{N}(\mathbf{m},\mathbf{K}+\sigma_n^2\mathbf{I})\\ &= \frac{1}{(2\pi)^{D/2}\left[\det(\mathbf{K}+\sigma_n^2\mathbf{I})\right]^{1/2}}\exp\left(\frac{1}{2}(\mathbf{y}-\mathbf{m})^T[\mathbf{K}+\sigma_n^2\mathbf{I}]^{-1}(\mathbf{y}-\mathbf{m})\right)\end{aligned} \tag{5.25}$$

The corresponding log marginal likelihood is given by

$$\begin{aligned}\log\left[p(\mathbf{y}|\mathbf{X})\right] =& \log\left(\frac{1}{(2\pi)^{D/2}\left[\det(\mathbf{K}+\sigma_n^2\mathbf{I})\right]^{1/2}}\exp\left(\frac{1}{2}(\mathbf{y}-\mathbf{m})^T[\mathbf{K}+\sigma_n^2\mathbf{I}]^{-1}(\mathbf{y}-\mathbf{m})\right)\right)\\ =& \underbrace{-\frac{1}{2}(\mathbf{y}-\mathbf{m})^T\left[\mathbf{K}+\sigma_n^2\mathbf{I}\right]^{-1}(\mathbf{y}-\mathbf{m})}_{\text{Data fit}}\\ & \underbrace{-\frac{1}{2}\log\left[\det\left(\mathbf{K}+\sigma_n^2\mathbf{I}\right)\right]}_{\text{Complexity penalty}}\\ & \underbrace{-\frac{n}{2}\log\left(2\pi\right)}_{\text{Constant term}}\end{aligned} \tag{5.26}$$

5.2.7 Posterior distribution

Given a set of training data $\mathcal{D}$, the GP regression model uses Bayesian inference to learn the distribution of $\mathbf{f}$ that is most likely to have generated the data. This involves computing the posterior distribution of $\mathbf{f}$ given the data, which is defined as

$$p(\mathbf{f}|\mathbf{y},\mathbf{X}) = \frac{p(\mathbf{y}|\mathbf{f},\mathbf{X})p(\mathbf{f})}{p(\mathbf{y}|\mathbf{X})} = \frac{p(\mathbf{y},\mathbf{f}|\mathbf{X})}{p(\mathbf{y}|\mathbf{X})} \tag{5.27}$$

where $p(\mathbf{y}|\mathbf{f},\mathbf{X})$ is the likelihood of the data given the function $\mathbf{f}$, $p(\mathbf{f})$ is the prior distribution of $\mathbf{f}$, $p(\mathbf{y}|\mathbf{X})$ is the marginal likelihood of the data, and $p(\mathbf{y},\mathbf{f}|\mathbf{X})$ is the joint distribution of $\mathbf{y}$ and $\mathbf{f}$ given $\mathbf{X}$.

5.2.8 Posterior predictive distribution

Once the posterior distribution of $\mathbf{f}$ has been learned, the model can make predictions at new test points $\mathbf{X}_*$ by computing the posterior predictive distribution, which is defined as

$$p(\mathbf{f}_*|\mathbf{y},\mathbf{X},\mathbf{X}_*) = \int p(\mathbf{f}_*|\mathbf{f},\mathbf{X}_*)p(\mathbf{f}|\mathbf{y},\mathbf{X})d\mathbf{f} \tag{5.28}$$

More specific, we are interested in inferring $\mathbf{f}_*$, given a set of new inputs $\mathbf{X}_*$. To infer $\mathbf{f}_*$ given observations $\mathbf{X}$, $\mathbf{y}$ and query $\mathbf{X}_*$, we write down the joint distribution of $\mathbf{y}$ and $\mathbf{f}_*$. By the definition of Gaussian process, it is a Gaussian distribution. Therefore, we have

$$\begin{bmatrix}\mathbf{y}\\ \mathbf{f}_*\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}m(\mathbf{X})\\ m(\mathbf{X}_*)\end{bmatrix}, \begin{bmatrix}K(\mathbf{X},\mathbf{X})+\sigma_n^2\mathbf{I} & K(\mathbf{X},\mathbf{X}_*)\\ K(\mathbf{X}_*,\mathbf{X}) & K(\mathbf{X}_*,\mathbf{X}_*)\end{bmatrix}\right) \tag{5.29}$$

Then we condition this multivariate Gaussian on the known training values, and we get the predictive distribution $p(\mathbf{f}_*|\mathbf{X}_*,\mathbf{X},\mathbf{y})$:

$$\mathbf{f}_*|\mathbf{X}_*,\mathbf{X},\mathbf{y} \sim \mathcal{N}(\bar{\mathbf{f}}_*, \operatorname{cov}(\mathbf{f}_*)) \tag{5.30}$$

where the posterior mean function is

$$\bar{\mathbf{f}}_* = m(\mathbf{X}_*) + K(\mathbf{X}_*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}]^{-1}(\mathbf{y} - m(\mathbf{X})) \tag{5.31}$$

and the posterior covariance function is

$$\mathrm{Cov}(\mathbf{f}_*) = K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X})[K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}]^{-1} K(\mathbf{X}, \mathbf{X}_*) \tag{5.32}$$

We can interpret the predictive formulas as follows

posterior mean $=$ prior mean $+$ correction conditioned on the observations

posterior uncertainty $=$ prior uncertainty - reduction in uncertainty

In case we predict for only one point, then $\mathbf{X}_* \equiv \mathbf{x}_*$. We have the predictive mean value $\bar{f}_* = \mu_{f_*}$, and its variance $\mathrm{Cov}(f_*) = \sigma_{f_*}^2$. The prediction interval at confidence level $(1-\alpha) \times 100\%$ for $0 < \alpha < 1$ is given by

$$\left[\mu_{f_*} - f_{\alpha/2}\sigma_{f_*}, \quad \mu_{f_*} + f_{1-\alpha/2}\sigma_{f_*}\right] \tag{5.33}$$

where $f_{\alpha/2} = \Phi^{-1}(\alpha/2)$ and $f_{1-\alpha/2} = \Phi^{-1}(1-\alpha/2)$ are, respectively, $(\alpha/2) \times 100\%$ and $(1-\alpha/2) \times 100\%$ quantiles of the standard normal distribution. Also, note that Φ denotes the CDF of the standard normal distribution.

5.3 Learning in Gaussian process models

Assume we have chosen a parameterized Gaussian prior $f(\mathbf{x}) \sim \mathcal{GP}(m_{\boldsymbol{\theta}}(\mathbf{x}), k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}'))$. The two main concerns in a GP model is to select that kernel function $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}')$ and tune its hyperparameters $\boldsymbol{\theta}$. These steps are incorporated in the term "model selection" and are crucial in order to ensure a well-fitted model.

5.3.1 Bayesian model comparison

One method for model selection is Bayesian model comparison, defined as follows

$$p(\mathcal{M}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathcal{M})p(\mathcal{M})}{\sum_{\mathcal{M}'} p(\mathcal{D}|\mathcal{M}')p(\mathcal{M}')} \propto p(\mathcal{D}|\mathcal{M})p(\mathcal{M}) \tag{5.34}$$

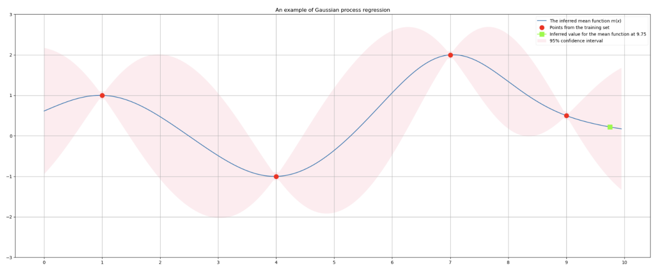
where $p(\mathcal{M})$ is the prior over models and $p(\mathcal{D}|\mathcal{M})$ is the marginal likelihood given by

$$p(\mathcal{D}|\mathcal{M}) = \int_{\boldsymbol{\theta}} p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M})p(\boldsymbol{\theta}|\mathcal{M})d\boldsymbol{\theta} \tag{5.35}$$

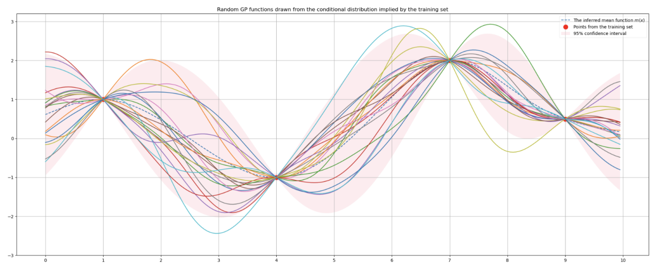
The optimal model $\mathcal{M}_*$ is given by

$$\mathcal{M}_* = \arg\max_{\mathcal{M}} \left[p(\mathcal{D}|\mathcal{M})p(\mathcal{M})\right] \tag{5.36}$$

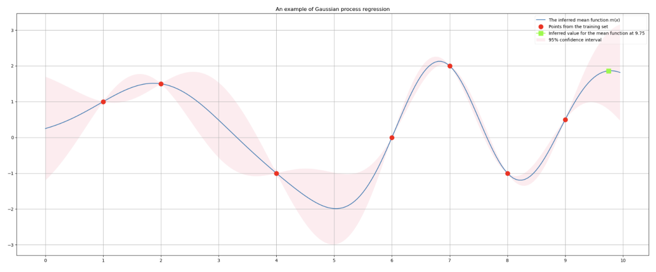
However, it does involve a very difficult integral (or sum in discrete case, as showed above) over the hyperparameters of our GP, which makes it impractical, and is also very sensitive to the prior we put over our hyperparameters.



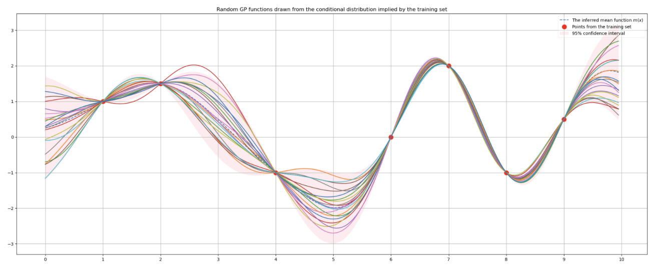
(a) Gaussian process regression on a training set of 4 points.



(b) Illustration of the posterior distribution over functions that the GP offers.



(c) Gaussian process regression on a training set of 7 points.



(d) Illustration of the posterior distribution over functions that the GP offers.

Figure 5.5: (a) It showcases Gaussian process regression with an initial set of 4 conditioning points, utilizing a GP prior of squared exponential kernel ($\sigma_f = 1, \ell = 1$) and a zero-mean function. (b) 20 samples are drawn from the GP posterior. (c) Enrichment of the model by introducing three additional observation points. This highlights the adaptive nature of the Gaussian process, as it refines predictions in response to increased data, evolving the predictive distribution accordingly. (d) 20 samples are drawn from the new GP posterior.

5.3.2 Maximum likelihood estimation

Another, more "friendly" approach to perform model selection is by experimenting with various kernel functions $k_{\boldsymbol{\theta}}$, and tune the hyperparameters $\boldsymbol{\theta}$ of each one using maximum likelihood estimation (MLE). Then you can compare the resulting likelihood values, or even test the final model's performance.

Under this approach, we will measure the quality of the fit to our training data $\mathcal{D}$ with the marginal likelihood, the probability of observing the given data under our prior. It is given by

$$\begin{aligned} p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) &= \int p(\mathbf{y}|\mathbf{f},\mathbf{X},\boldsymbol{\theta})p(\mathbf{f}|\mathbf{X},\boldsymbol{\theta})d\mathbf{f} \\ &= \mathcal{N}\left(\mathbf{m}_{\boldsymbol{\theta}}, \mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2\mathbf{I}\right) \\ &= \frac{1}{(2\pi)^{D/2}\left[\det(\mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2\mathbf{I})\right]^{1/2}} \exp\left(\frac{1}{2}(\mathbf{y}-\mathbf{m}_{\boldsymbol{\theta}})^T[\mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2\mathbf{I}]^{-1}(\mathbf{y}-\mathbf{m}_{\boldsymbol{\theta}})\right) \end{aligned} \tag{5.37}$$

We are interested in maximizing the marginal likelihood. So, in order to simplify the calculations, we can take equivalently the log marginal likelihood, which is given by

$$\begin{aligned} \log\left[p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})\right] = & \underbrace{-\frac{1}{2}(\mathbf{y}-\mathbf{m}_{\boldsymbol{\theta}})^T\left[\mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2\mathbf{I}\right]^{-1}(\mathbf{y}-\mathbf{m}_{\boldsymbol{\theta}})}_{\text{Data fit}} \\ & \underbrace{-\frac{1}{2}\log\left[\det\left(\mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2\mathbf{I}\right)\right]}_{\text{Complexity penalty}} \\ & \underbrace{-\frac{n}{2}\log\left(2\pi\right)}_{\text{Constant term}} \end{aligned} \tag{5.38}$$

The optimal hyperparameters $\boldsymbol{\theta}_*$ are given by

$$\boldsymbol{\theta}_* = \arg\max_{\boldsymbol{\theta}}\left[\log\left[p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})\right]\right] \tag{5.39}$$

Equivalently, we can minimize the negative log marginal likelihood, i.e

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}}\left[-\log\left[p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})\right]\right] \tag{5.40}$$

Optimizing the hyperparameters analytically involves the calculation of the partial differentials with respect to each component of $\boldsymbol{\theta}$, i.e. θ_j. We have

$$\begin{aligned} \frac{\partial}{\partial\theta_j}\log\left[p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})\right] &= \frac{1}{2}(\mathbf{y}-\mathbf{m}_{\boldsymbol{\theta}})^T\mathbf{K}_{\boldsymbol{\theta},\sigma_n}^{-1}\frac{\partial\mathbf{K}_{\boldsymbol{\theta},\sigma_n}}{\partial\theta_j}\mathbf{K}_{\boldsymbol{\theta},\sigma_n}^{-1}(\mathbf{y}-\mathbf{m}_{\boldsymbol{\theta}}) \\ &\quad - \frac{1}{2}\mathrm{Tr}\left(\mathbf{K}_{\boldsymbol{\theta},\sigma_n}^{-1}\frac{\partial\mathbf{K}_{\boldsymbol{\theta},\sigma_n}}{\partial\theta_j}\right) \\ &= \frac{1}{2}\mathrm{Tr}\left(\left(\mathbf{K}_{\boldsymbol{\theta},\sigma_n}^{-1}(\mathbf{y}-\mathbf{m}_{\boldsymbol{\theta}})(\mathbf{y}-\mathbf{m}_{\boldsymbol{\theta}})^T\left[\mathbf{K}_{\boldsymbol{\theta},\sigma_n}^T\right]^{-1} - \mathbf{K}_{\boldsymbol{\theta},\sigma_n}^{-1}\right)\frac{\partial\mathbf{K}_{\boldsymbol{\theta},\sigma_n}}{\partial\theta_j}\right) \end{aligned} \tag{5.41}$$

where $\mathbf{K}_{\boldsymbol{\theta},\sigma_n} = \mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2\mathbf{I}$. The essence of this method is to find values that make the observations seem probable. Gradients of the log marginal likelihood with respect to the hyperparameters can be computed, so we can use gradient-based optimizers. We cannot necessarily guarantee we will find a global optimum here and different solutions may lead to different interpretations of the data.

5.3.3 Maximum a posteriori estimation

By applying the Bayes rule, we obtain the posterior distribution of $\boldsymbol{\theta}$ given $\mathbf{y}$ and $\mathbf{X}$. We have

$$p(\boldsymbol{\theta}|\mathbf{y},\mathbf{X}) = \frac{p(\mathbf{y},\boldsymbol{\theta}|\mathbf{X})}{p(\mathbf{y}|\mathbf{X})} = \frac{p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{y}|\mathbf{X})} \propto p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{5.42}$$

where $p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})$ is the likelihood of the data given the hyperparameters $\boldsymbol{\theta}$, $p(\boldsymbol{\theta})$ is the prior distribution of $\boldsymbol{\theta}$, $p(\mathbf{y}|\mathbf{X})$ is the marginal likelihood of the data (which is ignored below as it does not depend on $\boldsymbol{\theta}$), and $p(\mathbf{y},\boldsymbol{\theta}|\mathbf{X})$ is the joint distribution of $\mathbf{y}$ and $\boldsymbol{\theta}$ given $\mathbf{X}$. The maximum a posteriori (MAP) estimate of $\boldsymbol{\theta}$ given that we have observed $\mathbf{Y} = \mathbf{y}$, is defined as follows

$$\begin{aligned}\boldsymbol{\theta}_* &= \arg\max_{\boldsymbol{\theta}} \left[p(\boldsymbol{\theta}|\mathbf{y},\mathbf{X})\right] \\ &= \arg\max_{\boldsymbol{\theta}} \left[\log\left[p(\boldsymbol{\theta}|\mathbf{y},\mathbf{X})\right]\right] \\ &= \arg\max_{\boldsymbol{\theta}} \left[\log\left[p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})p(\boldsymbol{\theta})\right]\right] \\ &= \arg\max_{\boldsymbol{\theta}} \left[\log\left[p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta})\right] + \log\left[p(\boldsymbol{\theta})\right]\right] \end{aligned} \tag{5.43}$$

Note that if we apply a **uniform prior distribution** over the hyperparameters $\boldsymbol{\theta}$, then MAP is **equivalent** to MLE.

5.3.4 Fully Bayesian inference and integral calculations

When trying to tune the hyperparameters of the GPR model, included in the vector $\boldsymbol{\theta}$, in the analysis above we used methods MLE and MAP, in a form that it is not required to do integral calculations. These methods can lead to overfitting.

A fully Bayesian approach is another way to avoid overfitting. It computes the posterior over a function value by integrating over all possible hyperparameters. This is stated by

$$p(\mathbf{f}_*|\mathbf{y},\mathbf{X},\mathbf{X}_*) = \int p(\mathbf{f}_*|\mathbf{y},\mathbf{X},\mathbf{X}_*,\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{y},\mathbf{X})d\boldsymbol{\theta} \tag{5.44}$$

For fully Bayesian inference, it is necessary to calculate marginal likelihoods analytically. This is often a intensive task and the above integral cannot be computed exactly. The first term in the integral is tractable. The second term is the posterior over hyperparameters, which comes from Bayes' rule, but requires approximation before we can compute the integral. There are methods that can be used for that task, like numerical integration, Markov Chain Monte Carlo, or variational methods. Although, we will not get further into this subject here.

5.4 Warped Gaussian Process Regression

In their simplest form, Gaussian processes (GPs) are constrained by their inherent simplicity, which involves an assumption that the target data follows a multivariate Gaussian distribution, with Gaussian noise affecting individual data points [47]. This simplicity facilitates the ease of making predictions through matrix manipulations, and it results in predictive distributions that also conform to the Gaussian distribution.

However, there are many situations where assuming Gaussian noise and modeling data as a GP may be impractical. For instance, the observations might consist of positive quantities spanning several orders of magnitude. In such cases, it might not be sensible to model these quantities directly while assuming homoscedastic Gaussian noise. To address this issue, it is a common practice in the statistics literature to apply transformations to the data. For example, taking the logarithm of the data is one such transformation. Subsequently, modeling proceeds with the assumption that this transformed data adheres to Gaussian noise, making it more amenable to modeling by a GP [47]. It is important to note that the logarithm is just one of many potential transformations that could be applied to the observation space to reshape the data into a form that can be effectively modeled using a GP. Such transformations are also the Box-Cox or the κ-logarithmic, which were discussed in a previous section. In essence, there exists a continuum of transformations that could be employed to align the data with the GP modeling framework.

In the view of applying a transformation on the data before building the GPR model, **"warped GPR"** is defined. This advanced framework allows for the vanilla GPR to be applied properly on data that are not Gaussianly distributed by transforming or "warping" the observation space. The "warped GPR" makes a transformation from a latent space to the observation, such that the data is best modelled by a GP in the latent space [47]. It can also be viewed as a generalization of the GP, since in observation space it is a non-Gaussian process, with non-Gaussian and asymmetric noise in general.

5.4.1 Making predictions

While working with warped GPR, we have defined a warping function g, that transform each observation y to its corresponding value z in the warped space. We train the model on the transformed data, and finally we are capable of making predictions in the warped space by obtaining the posterior predictive distribution as discussed above. At this point, assume that we have a point estimate μ_{z_*} and its corresponding variance $\sigma^2_{z_*}$ at $\mathbf{x}_*$. Transferring this prediction to the observation space is straightforward by means of the principle of "quantile invariance", which states that the quantiles of a probability distribution remain invariant under a monotonic transformation, i.e., if $\Phi(z_\alpha) = \alpha$ and $z_\alpha = g(y_\alpha)$, then it holds that $F_Y(y_\alpha) = \alpha$ [48]. Therefore, the predictive distribution in the observation space can be reconstructed from the respective distribution in the warped space by means of the function $g^{-1}(z)$. More precisely, the optimal prediction is given by [48]

$$\hat{y}_* = g^{-1}(\mu_{z_*}) \tag{5.45}$$

while the predictive interval at the confidence level $(1-\alpha) \times 100\%$ is given by [48]

$$\left[g^{-1}(\mu_{z_*} - f_{\alpha/2}\sigma_{z_*}), \quad g^{-1}(\mu_{z_*} + f_{1-\alpha/2}\sigma_{z_*})\right] \tag{5.46}$$

where $f_{\alpha/2} = \Phi^{-1}(\alpha/2)$ and $f_{1-\alpha/2} = \Phi^{-1}(1-\alpha/2)$ are, respectively, $(\alpha/2) \times 100\%$ and $(1-\alpha/2) \times 100\%$ quantiles of the standard normal distribution. Function Φ denotes the CDF of the standard normal distribution. Note that Equation (5.45) returns the median of the marginal predictive distribution in the observation space. This is due to the principle of quantile invariance, taking into account that the conditional mean μ_{z_*} is also the median of the latent variable's marginal conditional distribution [48].

For the warped space, the GPR gives a predictive distribution, which is Gaussian, and where the median and the mean lie at the same point. Transferring a prediction μ_{z_*} to the observation space, we get $\hat{y_*}$ by Equation (5.45), which is the **median** of the marginal predictive distribution in the observation space, as stated above. To calculate the **mean**, we need to calculate the following integral [47]:

$$E[y_*] = \int dz g^{-1}(z) \mathcal{N}_z(\mu_{z_*}, \sigma^2_{z_*}) \tag{5.47}$$

This is a simple one-dimensional integral under a Gaussian density, so Gauss-Hermite quadrature (see Appendix E) may be used to accurately compute it with a weighted sum of a small number of evaluations of the inverse function g^{-1} at appropriate places [47].

5.5 Mean Functions

Within the Gaussian process regression (GPR) framework, a critical initial decision pertains to the selection of the prior mean function. This choice assumes significance, as the prior mean function substantially influences the model's behavior and predictions. It serves as a foundational assumption regarding the inherent relationship between the input features and the target variable. It is worth noting that the mean function represents one of the two fundamental components of a Gaussian process (GP).

The prior mean function encapsulates the a priori beliefs or expectations we hold about the target variable's behavior before observing any data points. It serves as a foundational reference point for the GPR model to start making predictions. In essence, the prior mean function acts as an "initial best guess" or starting assumption regarding the average value of the target variable across the input space.

Two primary choices for the prior mean function are the zero mean and the constant mean. Let's delve into each of these choices and the implications they bring to the GPR model [4]:

- **Zero mean**:

$$m(\mathbf{X}) = 0 \tag{5.48}$$

  Opting for a zero mean function assumes that the expected value of the target variable is zero across the entire input space. This choice implies that the model begins with no inherent bias in its predictions. It is particularly useful when there is no prior information suggesting a specific average behavior of the target variable. A zero mean function is versatile and can adapt well to data with varying trends and patterns. Often, we prefer a zero mean function, as it offers simplification and has not a critical impact to the final model. GPs are flexible enough to model the mean even if it is set to an arbitrary value at the beginning.

- **Constant mean**:

$$m(\mathbf{X}) = c \tag{5.49}$$

  On the other hand, choosing a constant mean function involves specifying a fixed value as the mean across all inputs. This choice injects a degree of bias into the model's predictions, assuming a consistent average behavior across the dataset. A constant mean function can be advantageous when there is a strong prior belief that the target variable's mean is not zero and should be explicitly considered in the predictions.

The choice between a zero mean and a constant mean function should align with the insights you possess about the problem at hand. In GPR, the choice of the mean function does not have a significant impact on the model's behavior as the number of data points increases. The reason for this is that GPR is a flexible and adaptive method that relies on both the prior mean function and the covariance (kernel) function to capture patterns in the data. In fact, the predictive estimates converge towards the mean function as we move far away from observed data points, which is not really the case. This is a fundamental property of Gaussian processes. The covariance function, which encodes the relationships between data points, has a more pronounced influence on the shape of the predictions and the uncertainty estimates. In cases where there are a sufficient number of data points and the covariance function is well-tuned to the data, the predictive power of the model becomes increasingly reliant on the covariance function's ability to capture underlying patterns and relationships. This often makes the choice of the mean function less critical, as the model will adapt to the observed data and the covariance structure. We will see in a following subsection, that one can use a constant mean function, without much thought, consider its value as an additional model's hyperparameter and tune it through maximum likelihood estimation.

5.6 Kernel Functions

A kernel or covariance function describes the covariance of the Gaussian Process random variables. Letting $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$, the kernel function is denoted by $k(\mathbf{x}, \mathbf{x}') \in \mathbb{R}$ and it gives the covariance between $\mathbf{x}$ and $\mathbf{x}'$. Recall that the kernel function $k(\mathbf{x}, \mathbf{x}')$ together with the mean function $m(\mathbf{x})$ define the Gaussian process distribution:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

In general, what relates one observation to another is just the covariance function. We will use the popular kernel function named "squared exponential" to get a better understanding of its role.

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{(\mathbf{x} - \mathbf{x}')^2}{2l^2}\right)$$

where the maximum allowable covariance is defined as σ_f^2; this should be high for functions which cover a broad range on the y axis. If $\mathbf{x} \approx \mathbf{x}'$, then $k(\mathbf{x}, \mathbf{x}')$ approaches this maximum, meaning $f(\mathbf{x})$ is nearly perfectly correlated with $f(\mathbf{x}')$. This is good: for our function to look smooth, neighbours must be alike. Here, covariance decays exponentially fast as

$\mathbf{x}$ and $\mathbf{x}'$ become farther apart in the input space. Now, if $\mathbf{x}$ is distant from $\mathbf{x}'$, we have instead $k(\mathbf{x}, \mathbf{x}') \approx 0$, i.e., the two points cannot "see" each other. So, for example, during interpolation at new $\mathbf{x}$ values, distant observations will have negligible effect. How much effect this separation has will depend on the length parameter, l, so there is much flexibility built into ((5.55)) [49]. Not quite enough flexibility though: the data are often noisy as well, from measurement errors and so on. Consequently, we take the approach of folding the noise into $k(\mathbf{x}, \mathbf{x}')$, by writing

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{(\mathbf{x}-\mathbf{x}')^2}{2l^2}\right) + \sigma_n^2 \delta(\mathbf{x}, \mathbf{x}')$$

where $\delta(\mathbf{x}, \mathbf{x}')$ is the Kronecker delta function and σ_n^2 is the noise variance. We will explain further below.

5.6.1 Validity of a kernel

Suppose that we have a set $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) \in \mathbb{R}^{D\times N}$ of random variables. In order to be a valid kernel function the resulting kernel matrix $\mathbf{K}$ should be positive definite. That implies that the matrix should be symmetric and invertible. For us this means that if we define a covariance matrix $\mathbf{K}$, based on evaluating $k(\mathbf{x}_i, \mathbf{x}_j)$ at pairs of N $\mathbf{x}$-values, we must have that[50]

$$\mathbf{x}^T\mathbf{K}\mathbf{x} > 0 \quad \text{for all} \quad \mathbf{x} \neq \mathbf{0} \tag{5.50}$$

We intend to use as a covariance matrix in an MVN, and a positive (semi-)definite covariance matrix is required for MVN analysis. In that context, positive definiteness is the multivariate extension of requiring that a univariate Gaussian have positive variance parameter, σ^2.

The process of defining a new valid kernel from scratch it is not always trivial. Typically, predefined kernels are used to model a variety of processes. In what follows, we will visually explore some of these predefined kernels, and we will see how we can apply operations on them in order to construct new valid kernels.

5.6.2 White noise kernel

The white noise kernel represents independent and identically distributed noise added to the Gaussian process distribution. It has the form [46]

$$k(\mathbf{x}, \mathbf{x}') = \sigma_n^2 \delta(\mathbf{x}, \mathbf{x}') \tag{5.51}$$

where σ_n^2 is the variance of the noise and $\delta(\mathbf{x}, \mathbf{x}')$ is the Kronecker delta. This formula results in a covariance matrix with zeros everywhere except on the diagonal of the covariance matrix. This diagonal contains the variances of the individual random variables. All covariances between samples are zero because the noise is uncorrelated.

Figure 5.6 shows the white noise kernel as a function of the difference $\mathbf{x}-\mathbf{x}'$, and some samples drawn from a MVN with zero mean and white noise kernel for various values of the parameter σ_n, together with a visual representation of the corresponding covariance matrices.

5.6.3 Constant kernel

The equation for the constant kernel is given by [46]

$$k(\mathbf{x}, \mathbf{x}') = c \tag{5.52}$$

This kernel is mostly used in addition to other kernel functions. It depends on a single hyperparameter $c \geq 0$.

Figure 5.7 shows the constant kernel as a function of the difference $\mathbf{x} - \mathbf{x}'$, and some samples drawn from a MVN with zero mean and constant kernel for various values of the parameter c, together with a visual representation of the corresponding covariance matrices.

5.6.4 Linear kernel

The equation for the linear kernel is given by [46]

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \mathbf{x}^T \mathbf{x}' \tag{5.53}$$

The linear kernel is a dot-product kernel and thus, non-stationary. This kernel is often used in combination with the constant kernel to include a bias.

Figure 5.8 shows the linear kernel as a function of $\mathbf{x}$ and $\mathbf{x}'$, and some samples drawn from a MVN with zero mean and linear kernel for various values of the parameter σ_f, together with a visual representation of the corresponding covariance matrices.

5.6.5 Polynomial kernel

The equation for the polynomial kernel is given by [46]

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \left(\mathbf{x}^T \mathbf{x}' + c\right)^p \tag{5.54}$$

The polynomial kernel has an additional parameter $p \in \mathbb{R}$, that determines the degree of the polynomial. Since a dot-product is contained, the kernel is also non-stationary. The prior variance grows rapidly for $\|\mathbf{x}\|_2 \geq 1$ such that the usage for some regression problems is limited. It depends on a single hyperparameter $c \geq 0$.

Figure 5.9 shows the polynomial kernel as a function of $\mathbf{x}$ and $\mathbf{x}'$, and some samples drawn from a MVN with zero mean and polynomial kernel for various values of the parameter σ_f, together with a visual representation of the corresponding covariance matrices.

5.6.6 Squared exponential kernel

The squared exponential (or exponentiated quadratic, or radial basis function) kernel is one of the most popular kernels used in Gaussian process modelling. It has the form [46]

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\ell^2}\right) \tag{5.55}$$

where σ_f^2 is the scale factor, and ℓ is the lengthscale. The squared exponential kernel has become the de-facto default kernel for GPs. This is probably because it has some nice properties. It is universal, and you can integrate it against most functions that you need to. Every function in its prior has infinitely many derivatives. It also has only two parameters:

- The **lengthscale** ℓ determines the length of the "wiggles" in our function. In general, we will not be able to extrapolate more than l units away from our data.
- The **output variance** σ_f^2 determines the average distance of our function away from its mean. Every kernel has this parameter out in front; it is just a **scale factor**.

Figure 5.10 shows the squared exponential kernel as a function of the difference $\mathbf{x}-\mathbf{x}'$, and some samples drawn from a MVN with zero mean and squared exponential kernel for various values of the parameters σ_f, ℓ, together with a visual representation of the corresponding covariance matrices.

5.6.7 Squared exponential ARD kernel

The equation for the squared exponential ARD kernel is given by [46]

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-(\mathbf{x}-\mathbf{x}')^T \mathbf{P}^{-1} (\mathbf{x}-\mathbf{x}')\right) \tag{5.56}$$

where

$$\mathbf{P} = \operatorname{diag}(\ell_1, \ell_2, \ldots, \ell_D)$$

The automatic relevance determination (ARD) extension to the squared exponential kernel allows for independent lenghtscales, $\ell_1, \ell_2, \ldots, \ell_D > 0$, for each dimension of $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^D$. The individual lenghtscales are typically larger for dimensions which are irrelevant as the covariance will become almost independent of that input. We can interpret ℓ_j as the characteristic lengthscale of dimension j. If $\ell_j \to \infty$, then the corresponding dimension is ignored. If $\mathbf{P}$ is spherical, we get the isotropic kernel.

5.6.8 Exponential kernel

The exponential kernel has the form [51]

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|_2}{\ell}\right) \tag{5.57}$$

where σ_f^2 is the scale factor and ℓ is the lengthscale. This kernel is, also, called the Ornstein-Uhlenbeck kernel. Where the squared exponential function is smooth, the exponential kernel is only continuous — it is not differentiable. This has important implications in modeling, as the function approximations produced by kernel methods inherit the smoothness of the kernel. Hence, a smooth kernel (like the squared exponential) is good for fitting smooth functions, while a non-differentiable kernel (like the absolute exponential) may be a better choice for fitting non-differentiable (non-smooth) functions.

Figure 5.11 shows the exponential kernel as a function of the difference $\mathbf{x}-\mathbf{x}'$, and some samples drawn from a MVN with zero mean and exponential kernel for various values of the parameters σ_f, ℓ, together with a visual representation of the corresponding covariance matrices.

5.6.9 Rational quadratic kernel

The rational quadratic kernel is equivalent to adding together many squared exponential kernels with different lengthscales. So, GP priors with this kernel expect to see functions which vary smoothly across many length scales. It has the form [52]

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \left(1 + \frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\alpha\ell^2}\right)^{-\alpha} \tag{5.58}$$

where σ_f^2 is the scale factor, ℓ is the lengthscale and α is the scale-mixture factor. In particular, the parameter α determines the relative weighting of large-scale and small-scale variations. When $\alpha \to \infty$, the rational quadratic kernel is identical to the squared exponential kernel.

Figure 5.12 shows the rational quadratic kernel as a function of the difference $\mathbf{x} - \mathbf{x}'$, and some samples drawn from a MVN with zero mean and rational quadratic kernel for various values of the parameters σ_f, α, ℓ, together with a visual representation of the corresponding covariance matrices.

5.6.10 Periodic kernel

The periodic kernel has the form [52]

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{2\sin^2\left(\frac{\pi\|\mathbf{x}-\mathbf{x}'\|_2}{p}\right)}{\ell^2}\right) \tag{5.59}$$

where σ_f^2 is the scale factor, ℓ is the lengthscale, and p is the period factor. The periodic kernel allows one to model functions which repeat themselves exactly. Its parameters are easily interpretable:

- The **period** p simply determines the distance between repetitions of the function.
- The **lengthscale** ℓ determines the lengthscale in the same way as in the squared exponential kernel.

Figure 5.13 shows the periodic kernel as a function of the difference $\mathbf{x} - \mathbf{x}'$, and some samples drawn from a MVN with zero mean and periodic kernel for various values of the parameters σ_f, p, ℓ, together with a visual representation of the corresponding covariance matrices.

5.6.11 Cosine kernel

The cosine kernel has the form

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \cos\left(\frac{2\pi\|\mathbf{x} - \mathbf{x}'\|_2}{p}\right) \tag{5.60}$$

where σ_f^2 is the scale factor and p is the period factor. These parameters have the same interpretation as in the periodic kernel; cosine kernel is another form of a "periodic" covariance function.

Figure 5.14 shows the cosine kernel as a function of the difference $\mathbf{x} - \mathbf{x}'$, and some samples drawn from a MVN with zero mean and cosine kernel for various values of the parameters σ_f, p, together with a visual representation of the corresponding covariance matrices.

5.6.12 The Matérn kernel

The equation for the Matérn kernel is given by [4]

$$k(\mathbf{x}, \mathbf{x}') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\|\mathbf{x} - \mathbf{x}'\|_2}{\ell} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}\|\mathbf{x} - \mathbf{x}'\|_2}{\ell} \right) \tag{5.61}$$

with positive parameters ν and ℓ, where K_ν is a modified Bessel function, and *Gamma* is the gamma function.

Note that the scaling is chosen so that for $\nu \to \infty$ we obtain the SE covariance function. For the Matérn class, the process $f(\mathbf{x})$ is k-times MS differentiable if and only if $\nu > k$. The Matérn covariance functions become especially simple when ν is half-integer, i.e $\nu = p + 1/2$, where p is a non-negative integer. In this case the covariance function is a product of an exponential and a polynomial of order p, the general expression is given by [4]

$$k_{\nu=p+1/2}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\sqrt{2\nu}\|\mathbf{x} - \mathbf{x}'\|_2}{\ell} \right) \frac{\Gamma(p+1)}{\Gamma(2p+1)} \sum_{i=0}^{p} \frac{(p+i)!}{i!(p-i)!} \left(\frac{\sqrt{8\nu}\|\mathbf{x} - \mathbf{x}'\|_2}{\ell} \right)^{p-i} \tag{5.62}$$

It is possible that the most interesting cases for machine learning are $\nu = 3/2$ and $\nu = 5/2$, for which [4]

$$k_{\nu=3/2}(\mathbf{x}, \mathbf{x}') = \left(1 + \frac{\sqrt{3}\|\mathbf{x} - \mathbf{x}'\|_2}{\ell} \right) \exp\left(-\frac{\sqrt{3}\|\mathbf{x} - \mathbf{x}'\|_2}{} \right) \tag{5.63}$$

and

$$k_{\nu=5/2}(\mathbf{x}, \mathbf{x}') = \left(1 + \frac{\sqrt{5}\|\mathbf{x} - \mathbf{x}'\|_2}{\ell} + \frac{5\|\mathbf{x} - \mathbf{x}'\|_2^2}{3\ell^2} \right) \exp\left(-\frac{\sqrt{5}\|\mathbf{x} - \mathbf{x}'\|_2}{} \right) \tag{5.64}$$

since for $\nu = 1/2$ the process becomes very rough, and for $\nu \geq 7/2$, in the absence of explicit prior knowledge about the existence of higher order derivatives, it is probably very hard from finite noisy training examples to distinguish between values of $\nu \geq 7/2$ (or even to distinguish between finite values of ν and $\nu \to \infty$, the smooth squared exponential, in this case).

Figure 5.15 shows the Matérn kernel as a function of the difference $\mathbf{x} - \mathbf{x}'$, and some samples drawn from a MVN with zero mean and Matérn kernel for various values of the parameters ν, ℓ, together with a visual representation of the corresponding covariance matrices.

5.6.13 Making new kernels from old

Previously, we stated many commonly used covariance functions. Now, we will show the two main ways to combine existing kernel functions to make new ones.

The **sum of two kernels** is a valid kernel [4]. Proof: consider the random process $f(\mathbf{x}) = f_1(\mathbf{x}) + f_2(\mathbf{x})$, where $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ are independent. Then $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$. This construction can be used e.g. to add together kernels with different characteristic lengthscales. In Figure 5.19, an example of the sum of two kernels is shown.

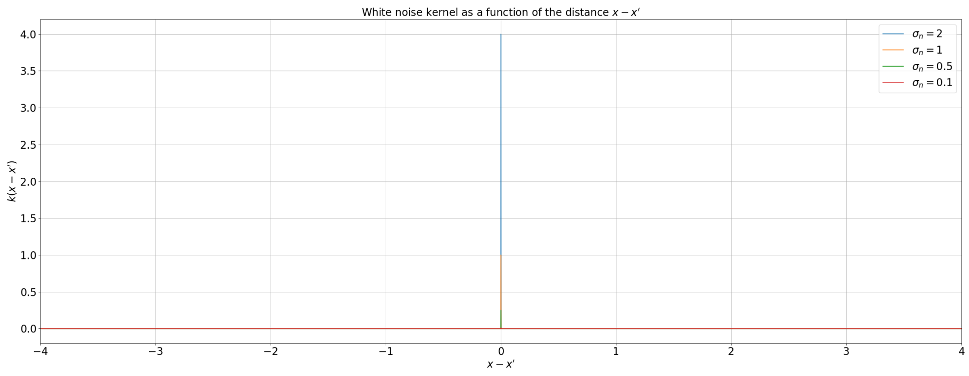
The **product of two kernels** is a valid kernel [4]. Proof: consider the random process $f(\mathbf{x}) = f_1(\mathbf{x})f_2(\mathbf{x})$, where $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ are independent. $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$. A simple extension of this argument means that $k^p(\mathbf{x}, \mathbf{x}')$ is a valid covariance function for $p \in \mathbb{N}$. In Figure 5.16, 5.17, and 5.18, three examples of the product of two kernels are shown.

5.7 A numerically stable GPR algorithm

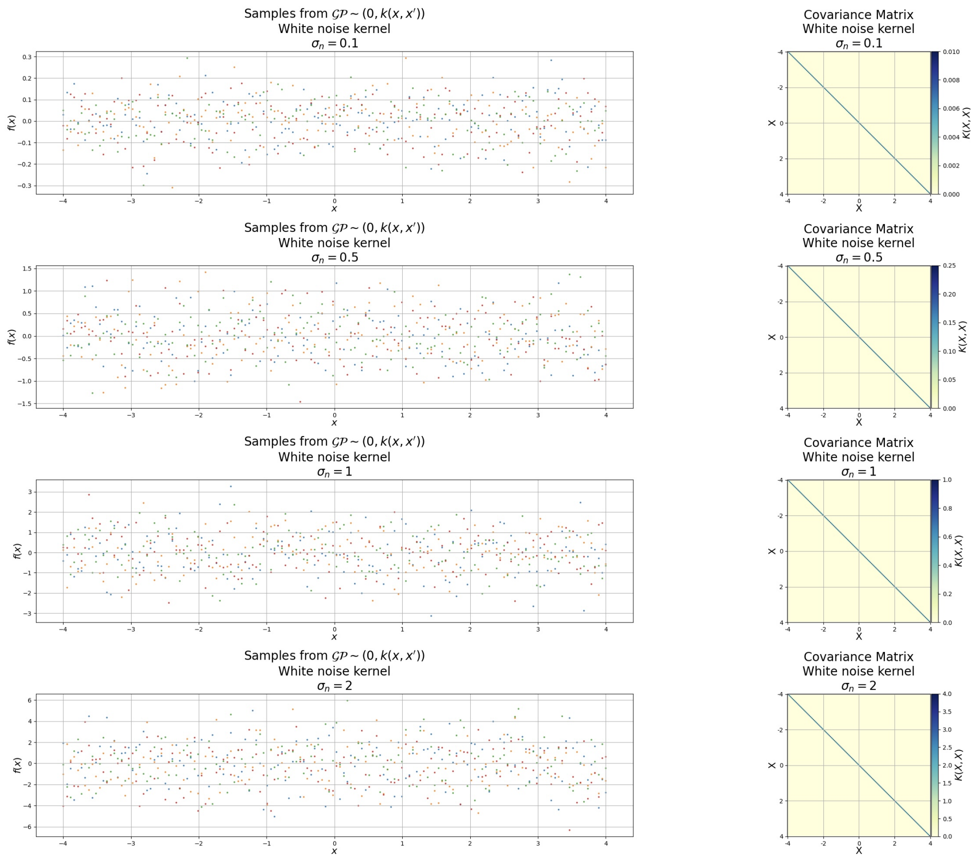
Algorithm 5.1 is an efficient algorithm for making predictions and calculating log marginal likelihood in a Gaussian process regression framework [4]. The implementation of that algorithm addresses the matrix inversion required by Equations (5.31) and (5.32) using Cholesky factorization (see Appendix D). For multiple test cases, lines 4-6 are repeated. The log determinant required in Equation (5.27) is computed from the Cholesky factor (for large n it may not be possible to represent the determinant itself). The computational complexity is $n^3/6$ for the Cholesky decomposition in line 2, and $n^2/2$ for solving triangular systems in lines 3-4 and (for each test case) in line 6.

Algorithm 5.1: An efficient Gaussian process regression algorithm [4]

```
Input: Training data: $\mathbf{X}$ (inputs), $\mathbf{y}$ (targets), Test input: $\mathbf{x}_*$, Mean function:
       $m_{\boldsymbol{\theta}}(\mathbf{x})$, Covariance function: $k_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{x}')$, and Kernel's hyperparameters $\boldsymbol{\theta}$
Output: Predictive mean: $\bar{f}_*$, Predictive variance: $\mathrm{Var}(f_*)$, and Log marginal
        likelihood: $\log[p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})]$
1 Compute covariance matrix: $\mathbf{K} = K_{\boldsymbol{\theta}}(\mathbf{X}, \mathbf{X}) + \sigma_n^2\mathbf{I}$
2 Perform Cholesky decomposition on matrix $\mathbf{K}$ and obtain factor $\mathbf{L}$: $\mathbf{K} = \mathbf{L}\mathbf{L}^T$
3 Solve $\mathbf{L}\mathbf{u} = \mathbf{y}$ for $\mathbf{u}$
4 Solve $\mathbf{L}^T\boldsymbol{\alpha} = \mathbf{u}$ for $\boldsymbol{\alpha}$
5 Compute kernel vector $\mathbf{k}_* = k(\mathbf{X}, \mathbf{x}_*)$
6 Solve $\mathbf{L}\mathbf{v} = \mathbf{k}_*$ for $\mathbf{v}$
7 Compute predictive mean $\bar{f}_* = \mathbf{k}_*^T\boldsymbol{\alpha}$
8 Compute predictive variance $\mathrm{Var}(f_*) = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^T\mathbf{v}$
9 Compute the log marginal likelihood
    $\log[p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})] = \frac{1}{2}\mathbf{y}^T\boldsymbol{\alpha} - \sum_i \log(L_{ii}) - \frac{n}{2}\log(2\pi)$
```



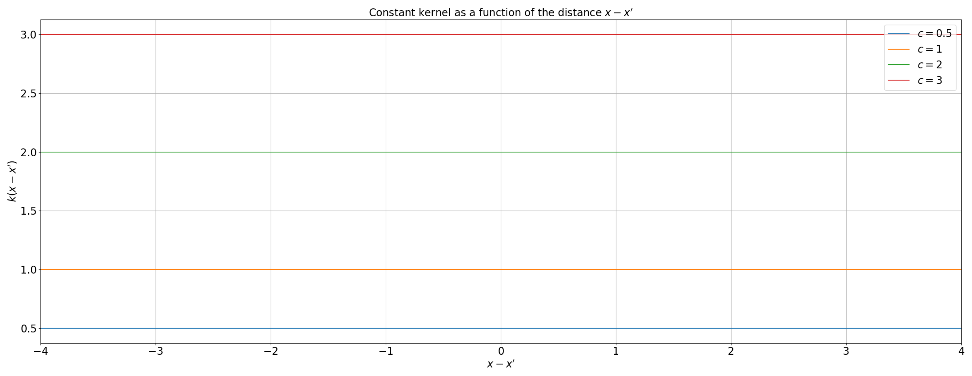
(a) White noise kernel as a function of the difference $x - x'$ for various values of the parameter σ_n.



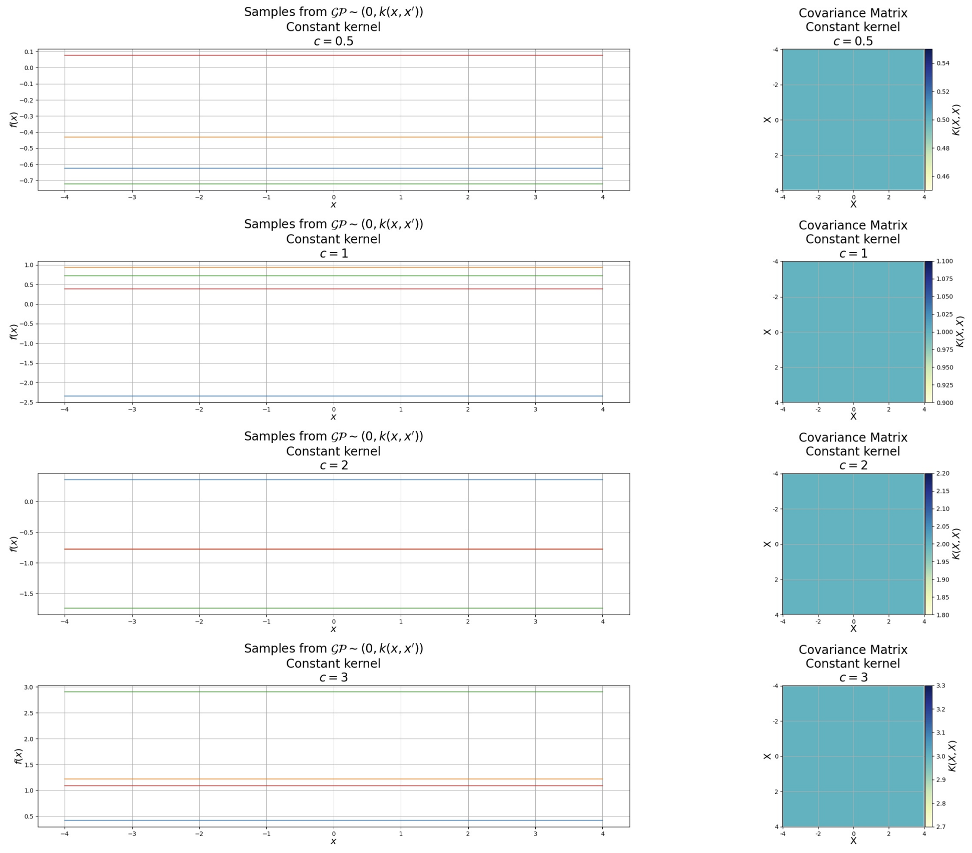
(b) (Left) Samples from a MVN [1]with zero mean and white noise kernel for various values of the parameter σ_n. (Right) Covariance matrices corresponding to the GP samples drawn on the left.

Figure 5.6: Visualization of the white noise kernel function.

[1]The covariance matrix was factorized using Cholesky decomposition.



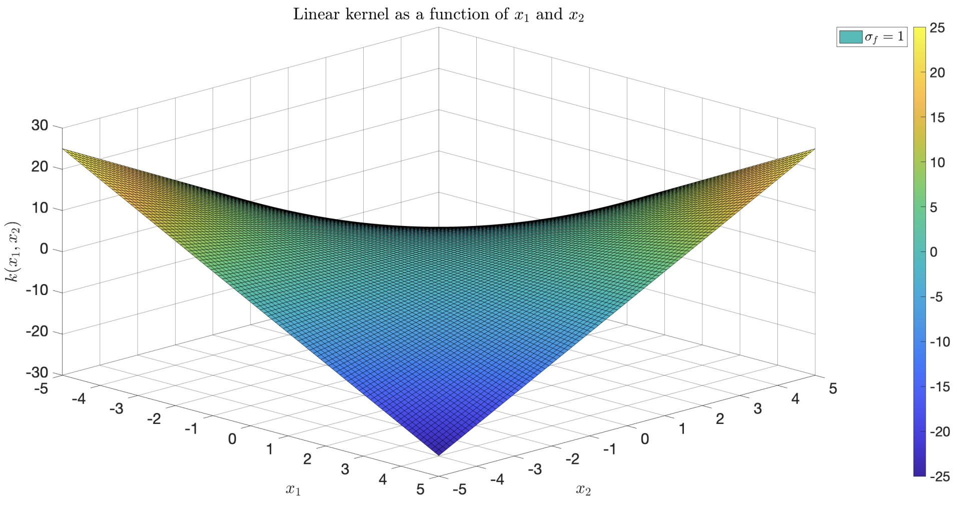
(a) Constant kernel as a function of the difference $x - x'$ for various values of the parameter c.



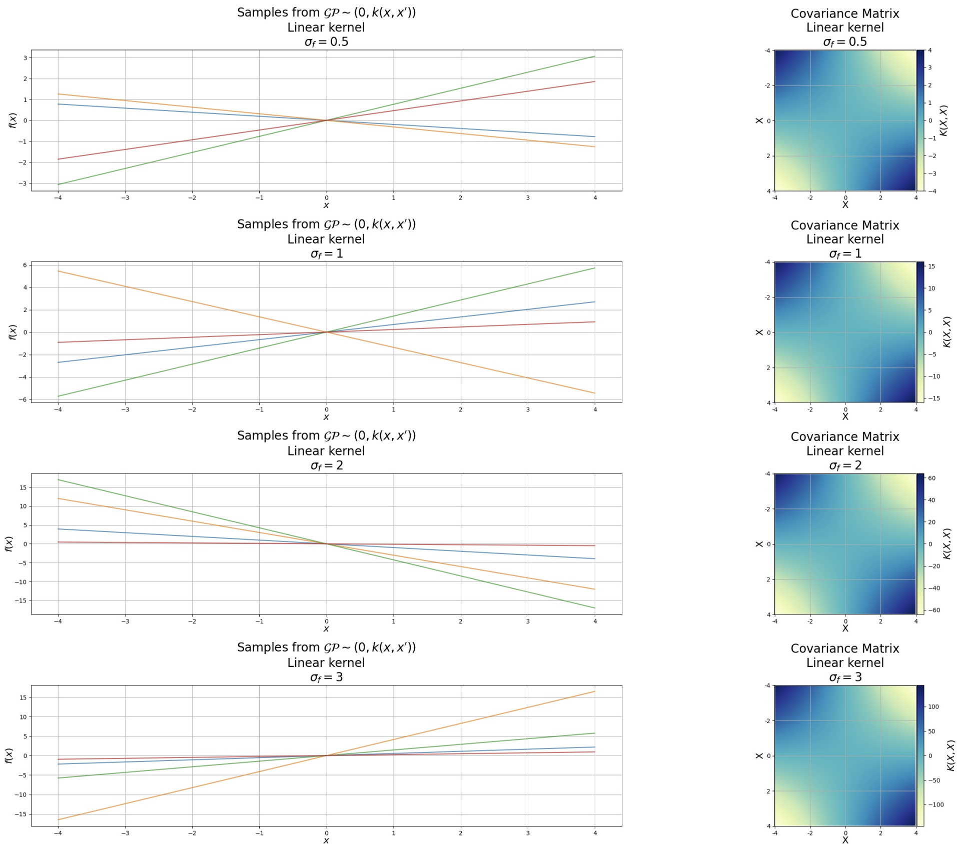
(b) (Left) Samples from a MVN [2]with zero mean and constant kernel for various values of the parameter c. (Right) Covariance matrices corresponding to the GP samples drawn on the left.

Figure 5.7: Visualization of the constant kernel function.

[2]The covariance matrix was factorized using Cholesky decomposition.



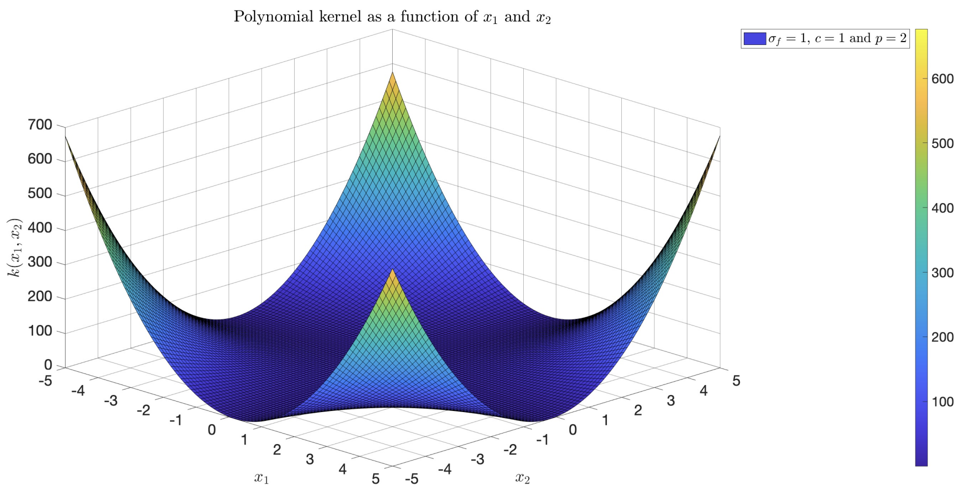
(a) Linear kernel as a function of x and x' for various values of the parameter σ_f.



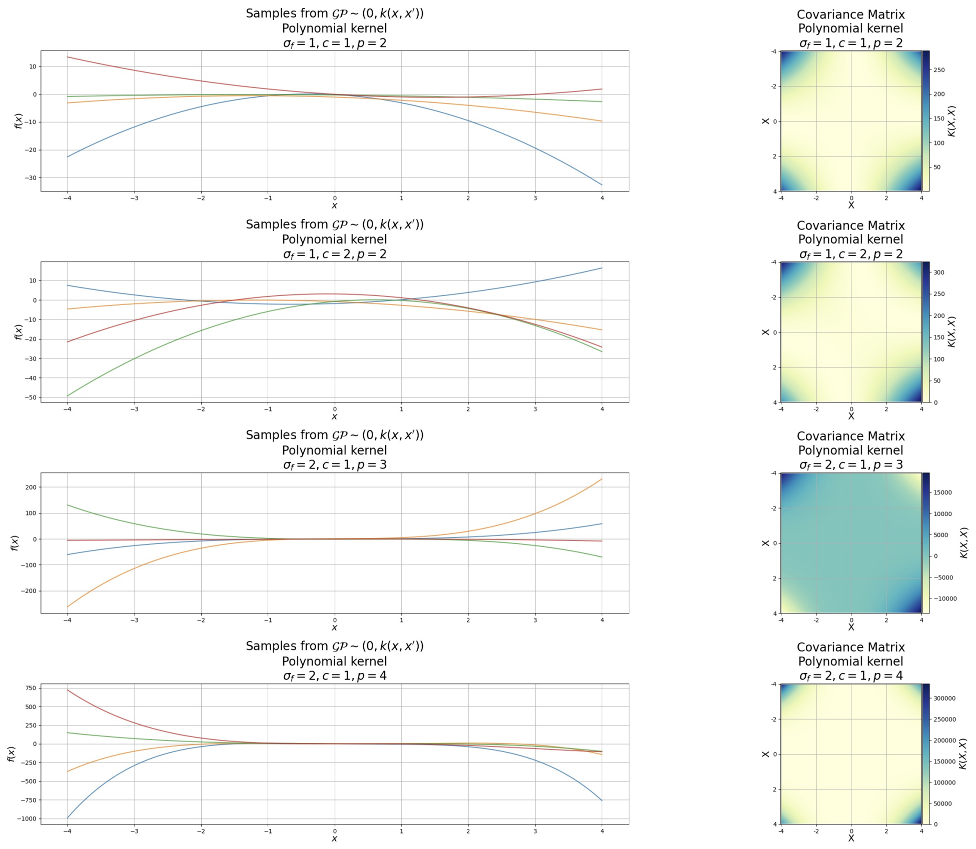
(b) (Left) Samples from a MVN [3]with zero mean and linear kernel for various values of the parameter σ_f. (Right) Covariance matrices corresponding to the GP samples drawn on the left.

Figure 5.8: Visualization of the linear kernel function.

[3]The covariance matrix was factorized using Cholesky decomposition.



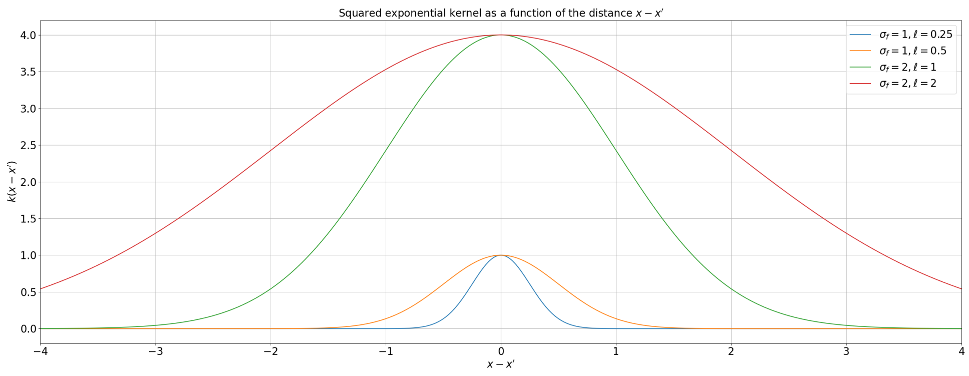
(a) Polynomial kernel as a function of x and x' for various values of the parameter σ_f.



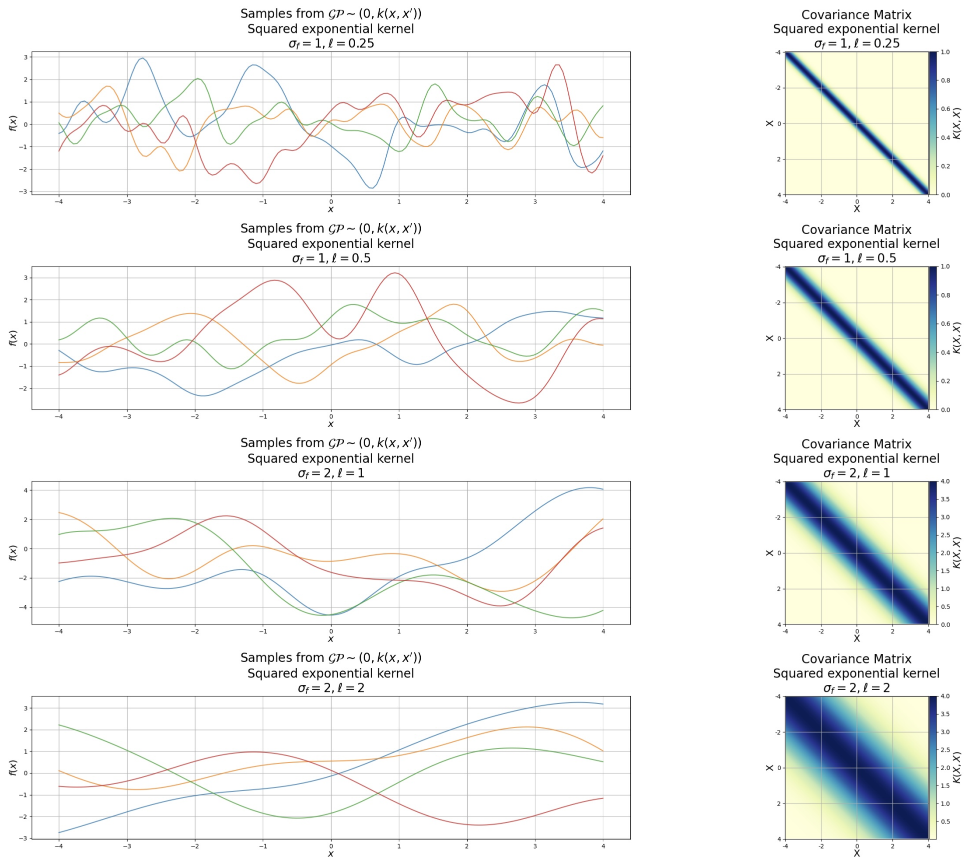
(b) (Left) Samples from a MVN [4]with zero mean and polynomial kernel for various values of the parameter σ_f. (Right) Covariance matrices corresponding to the GP samples drawn on the left.

Figure 5.9: Visualization of the polynomial kernel function.

[4]The covariance matrix was factorized using Cholesky decomposition.



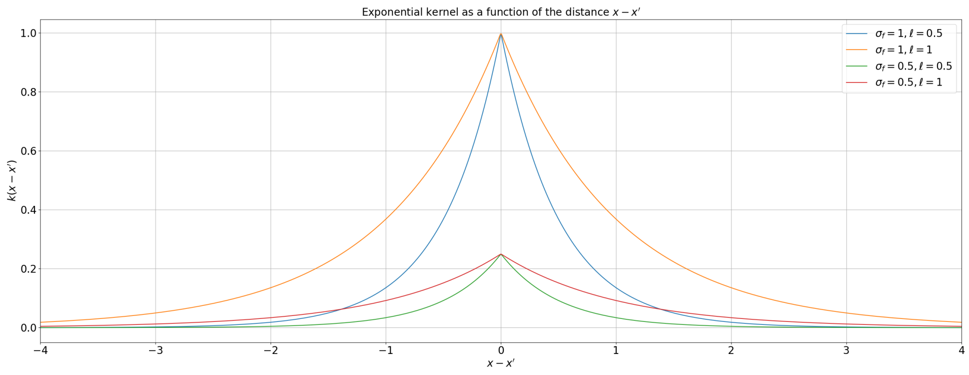
(a) Constant kernel as a function of the difference $x - x'$ for various values of the parameters σ_f and ℓ.



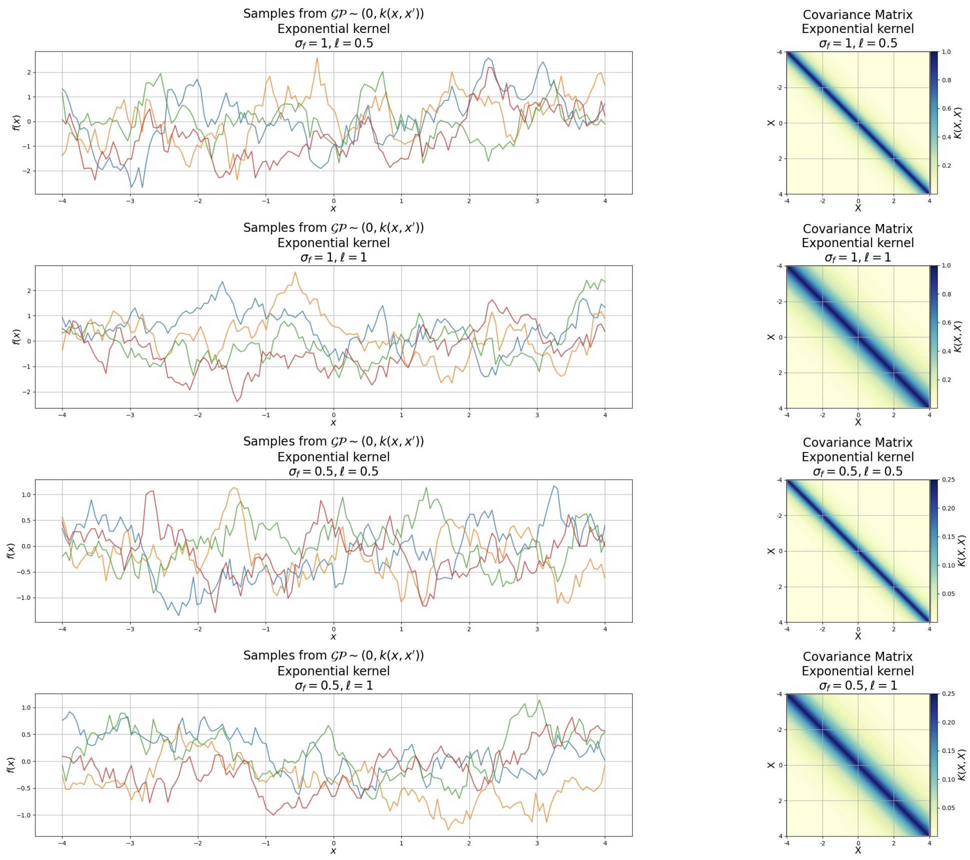
(b) (Left) Samples from a MVN [5]with zero mean and squared exponential kernel for various values of the parameters σ_f and ℓ. (Right) Covariance matrices corresponding to the GP samples drawn on the left.

Figure 5.10: Visualization of the squared exponential kernel function.

[5]The covariance matrix was factorized using Cholesky decomposition.



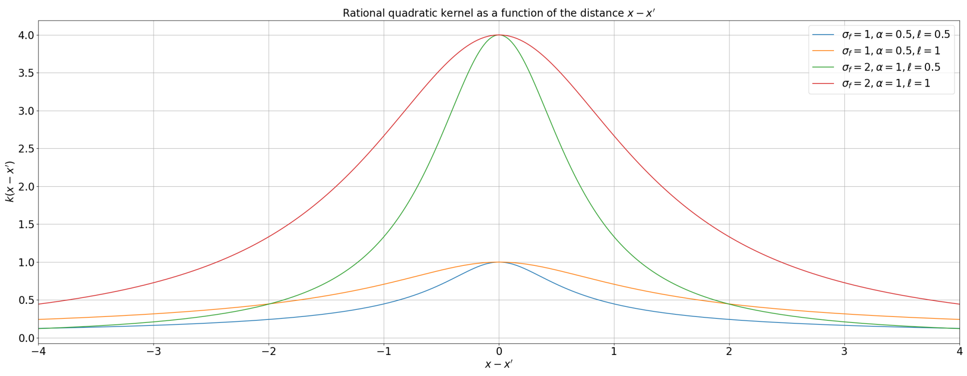
(a) Exponential kernel as a function of the difference $x - x'$ for various values of the parameters σ_f and ℓ.



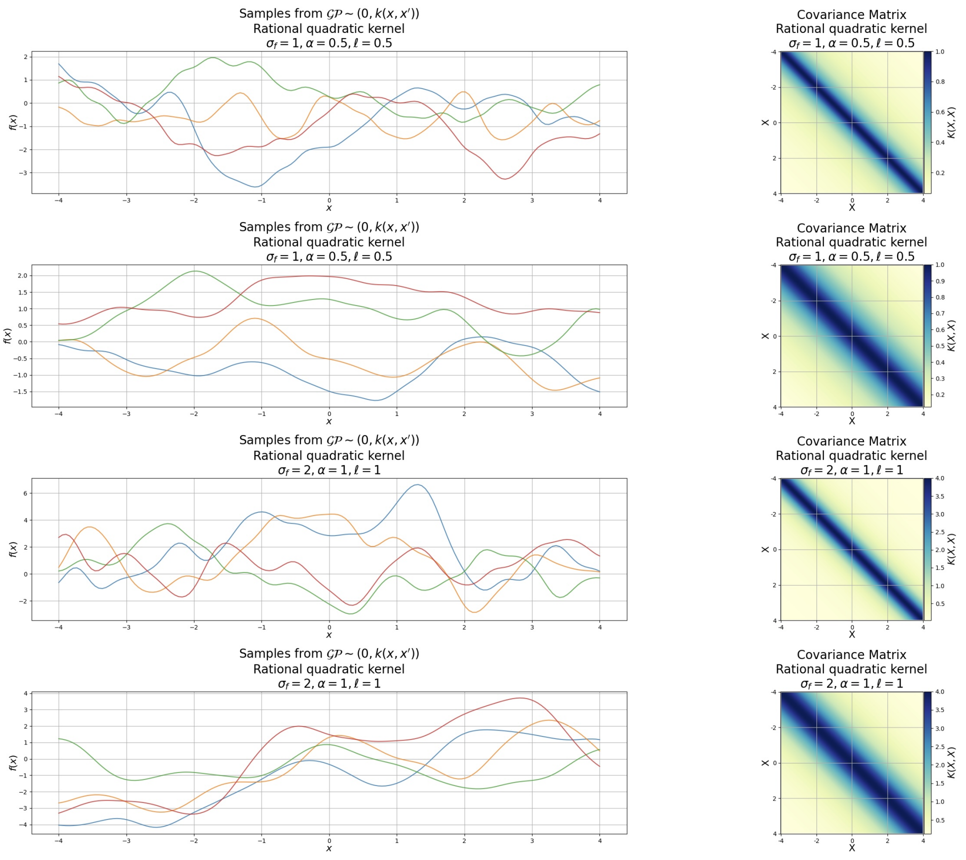
(b) (Left) Samples from a MVN [6]with zero mean and exponential kernel for various values of the parameters σ_f and ℓ. (Right) Covariance matrices corresponding to the GP samples drawn on the left.

Figure 5.11: Visualization of the exponential kernel function.

[6]The covariance matrix was factorized using Cholesky decomposition.



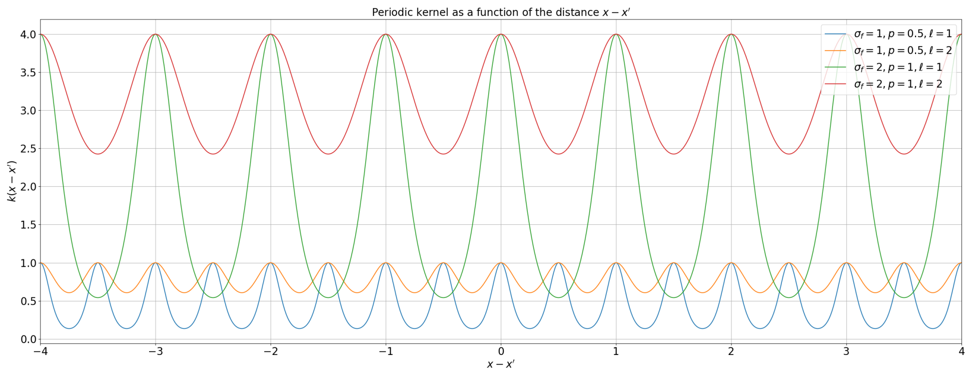
(a) Rational quadratic kernel as a function of the difference $x - x'$ for various values of the parameters σ_f, α, and ℓ.



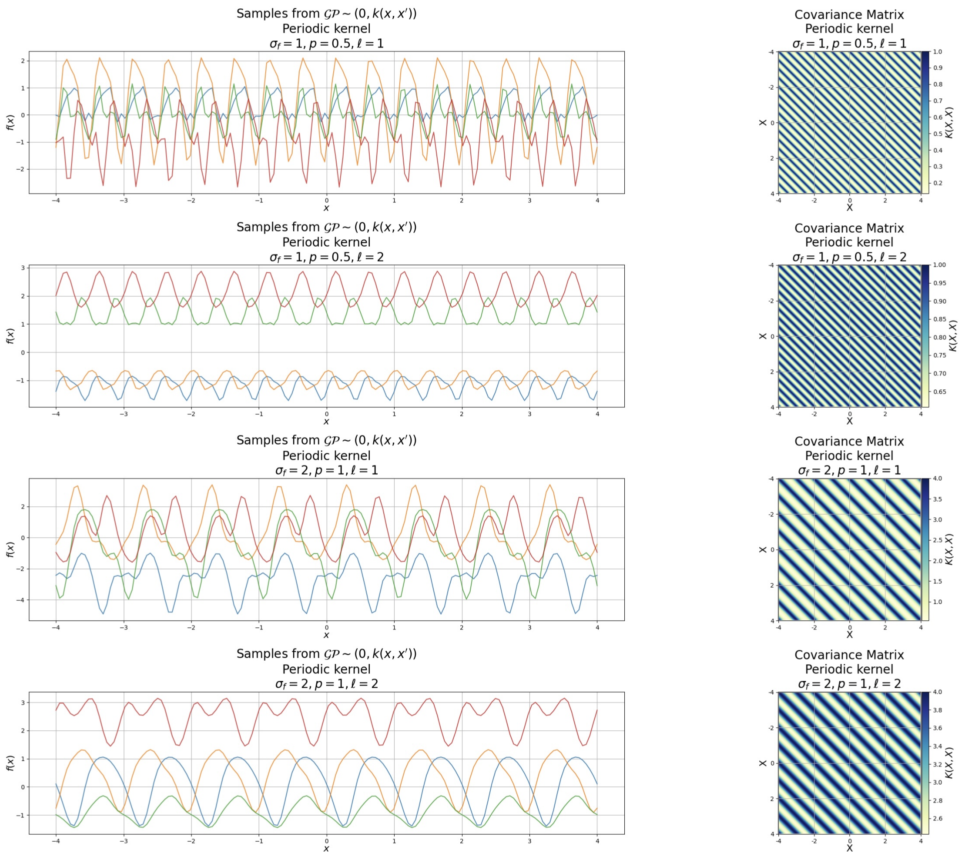
(b) (Left) Samples from a MVN[7] with zero mean and rational quadratic kernel for various values of the parameters σ_f, α, and ℓ. (Right) Covariance matrices corresponding to the GP samples drawn on the left.

Figure 5.12: Visualization of the rational quadratic kernel function.

[7]The covariance matrix was factorized using Cholesky decomposition.



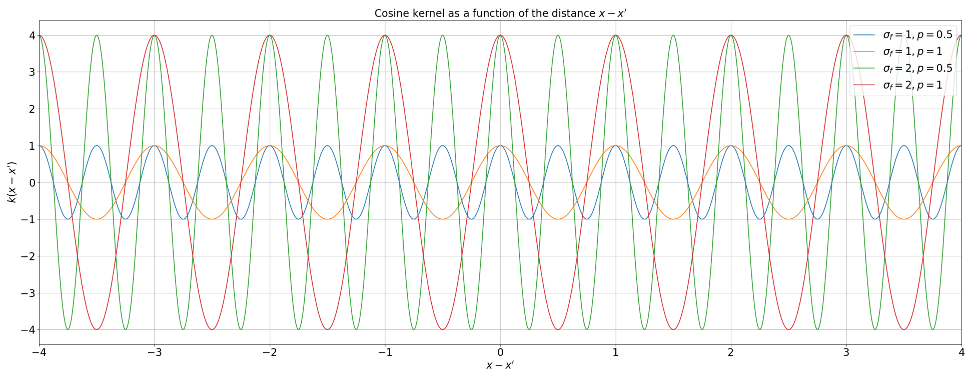
(a) Periodic kernel as a function of the difference $x - x'$ for various values of the parameters σ_f, p, and ℓ.



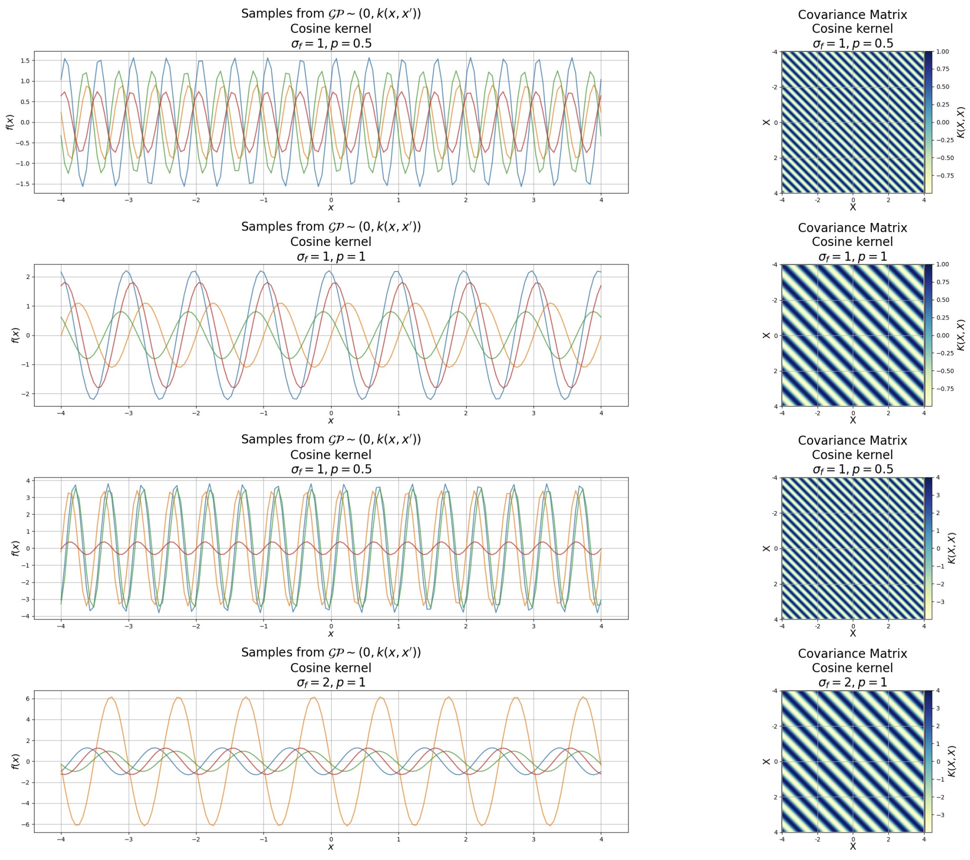
(b) (Left) Samples from a MVN [8]with zero mean and periodic kernel for various values of the parameters σ_f, p, and ℓ. (Right) Covariance matrices corresponding to the GP samples drawn on the left.

Figure 5.13: Visualization of the periodic kernel function.

[8]The covariance matrix was factorized using Cholesky decomposition.



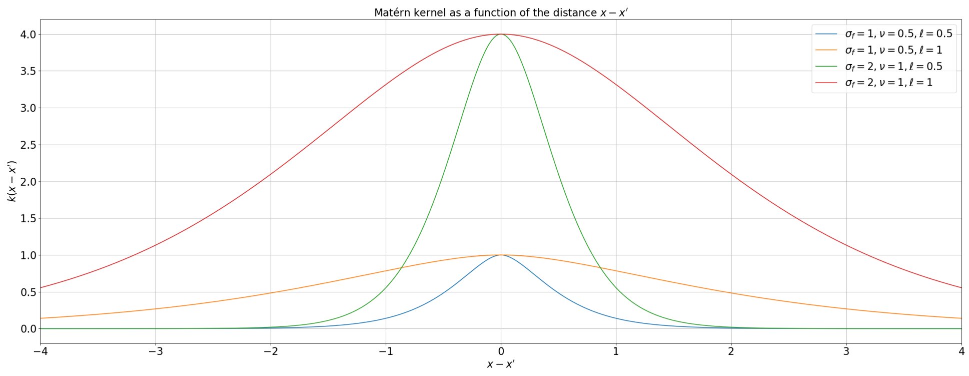
(a) Cosine kernel as a function of the difference $x - x'$ for various values of the parameters σ_f, p, and ℓ.



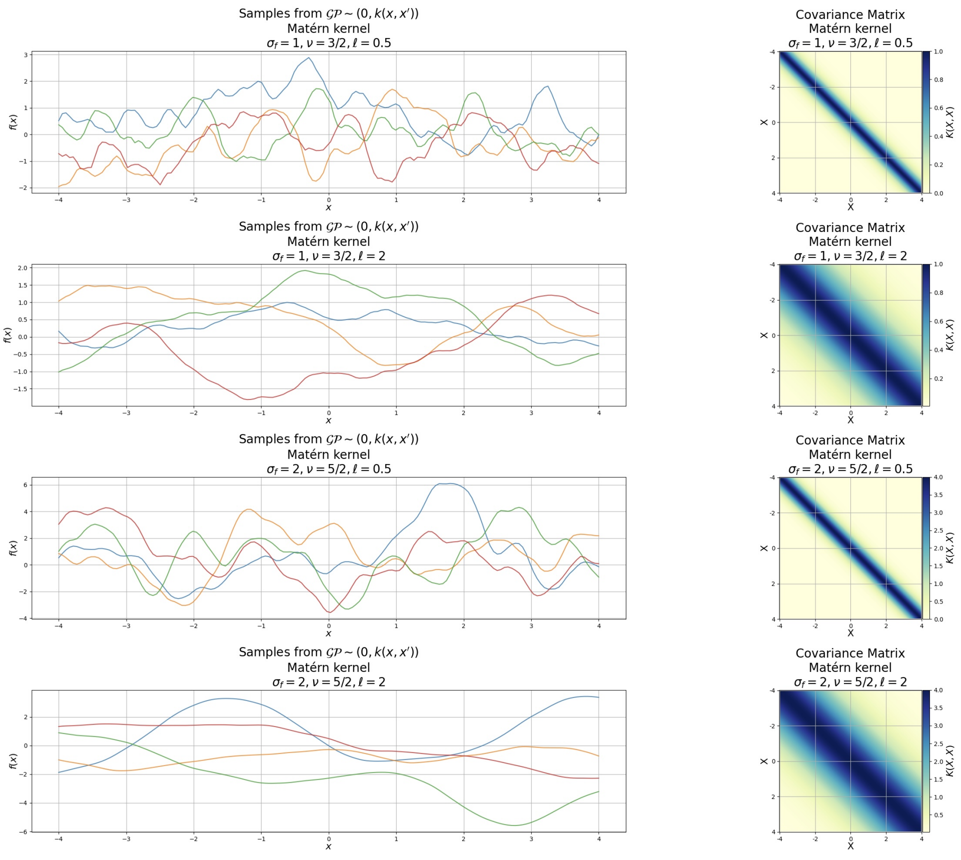
(b) (Left) Samples from a MVN [9]with zero mean and cosine kernel for various values of the parameters σ_f, p, and ℓ. (Right) Covariance matrices corresponding to the GP samples drawn on the left.

Figure 5.14: Visualization of the cosine kernel function.

[9]The covariance matrix was factorized using Cholesky decomposition.



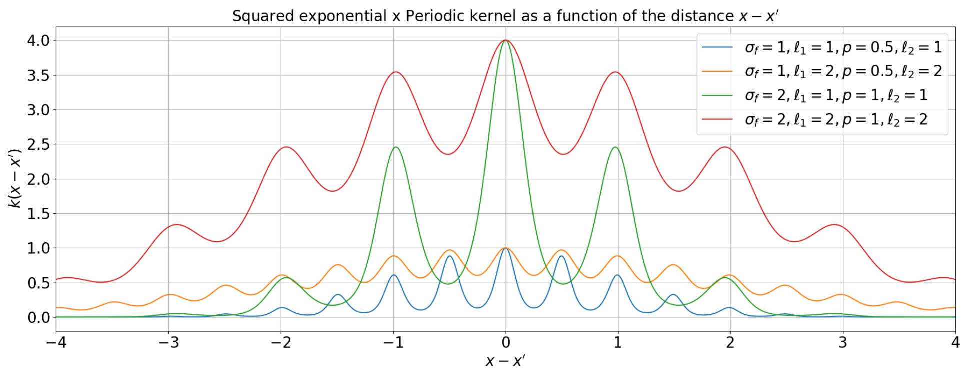
(a) Matérn kernel as a function of the difference $x - x'$ for various values of the parameters σ_f, p, and ℓ.



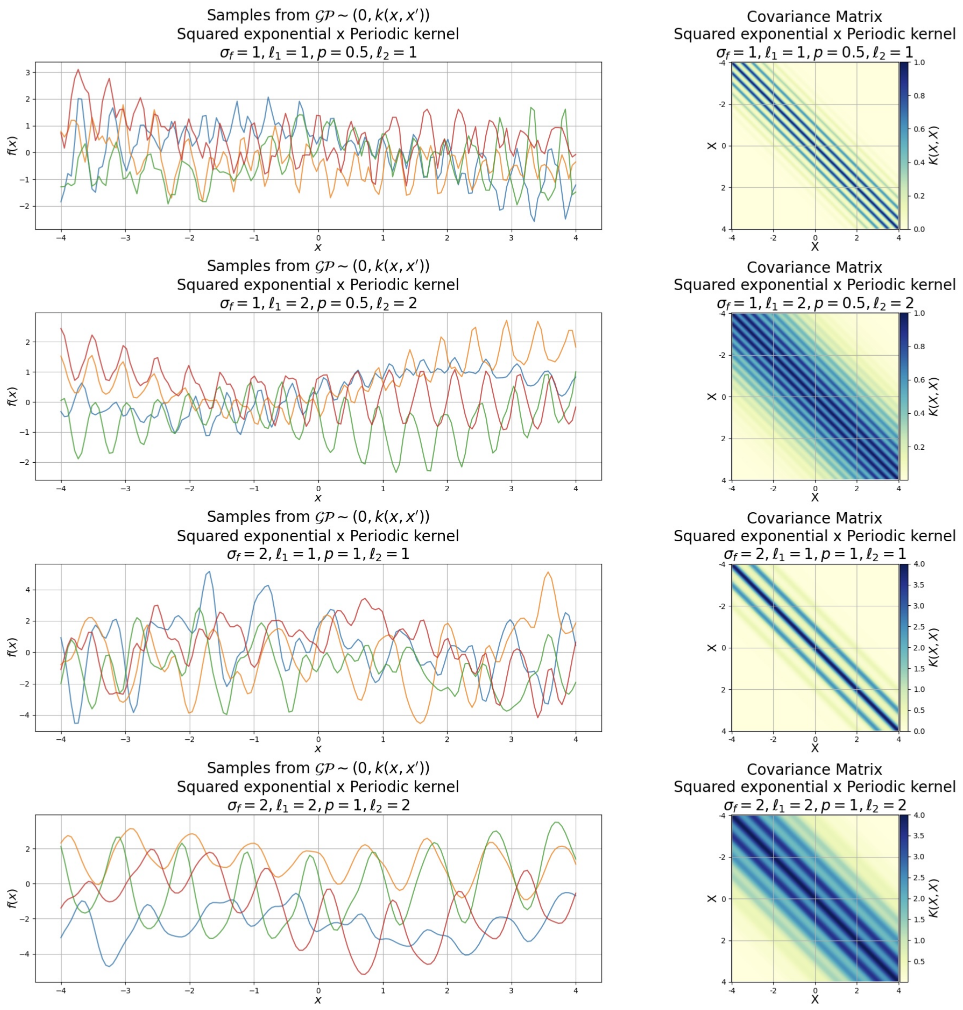
(b) (Left) Samples from a MVN [10]with zero mean and Matérn kernel for various values of the parameters σ_f, p, and ℓ. (Right) Covariance matrices corresponding to the GP samples drawn on the left.

Figure 5.15: Visualization of the Matérn kernel function.

[10]The covariance matrix was factorized using Cholesky decomposition.



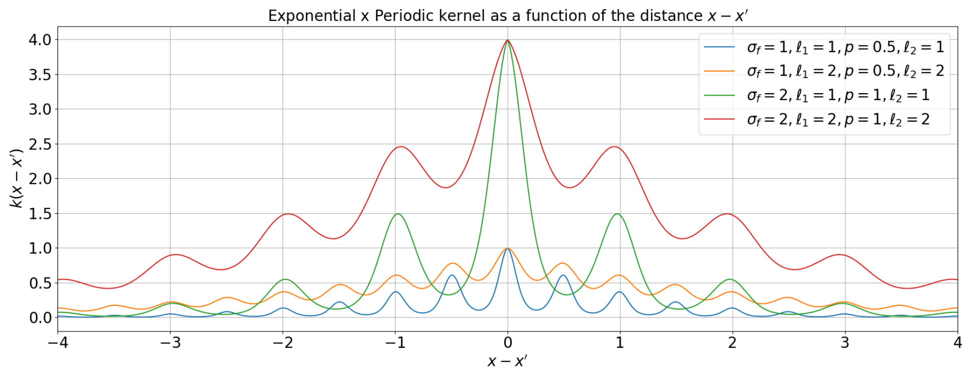
(a) Squared exponential × Periodic kernel as a function of the difference $x - x'$ for various values of the parameters σ_f, ℓ_1, p, and ℓ_2.



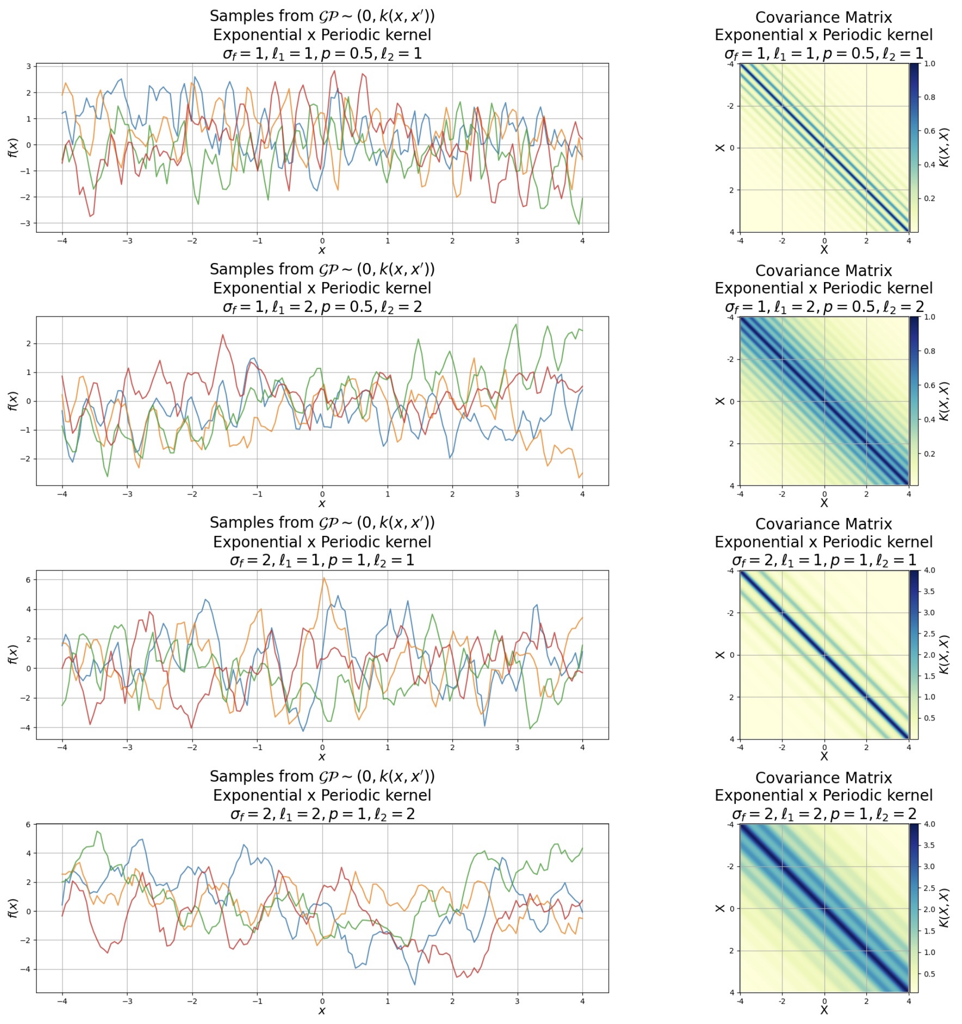
(b) (Left) Samples from a MVN [11]with zero mean and Squared exponential × Periodic kernel for various values of the parameters σ_f, ℓ_1, p, and ℓ_2. (Right) Covariance matrices corresponding to the GP samples drawn on the left.

Figure 5.16: Visualization of the Squared exponential × Periodic kernel function.

[11]The covariance matrix was factorized using Cholesky decomposition.



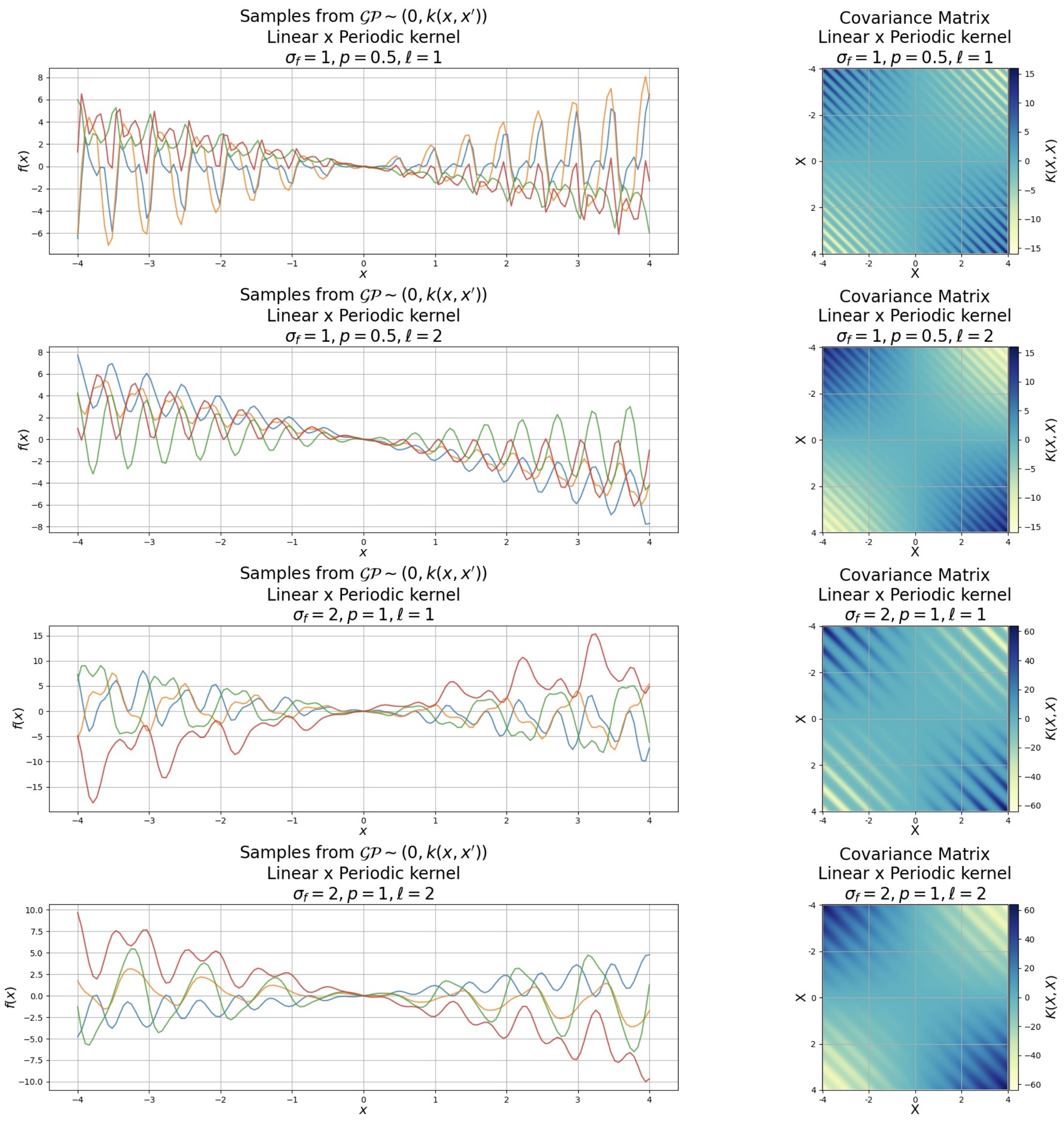
(a) Exponential $\times$ Periodic kernel as a function of the difference $x - x'$ for various values of the parameters σ_f, ℓ_1, p, and ℓ_2.



(b) (Left) Samples from a MVN [12]with zero mean and Exponential $\times$ Periodic kernel for various values of the parameters σ_f, ℓ_1, p, and ℓ_2. (Right) Covariance matrices corresponding to the GP samples drawn on the left.

Figure 5.17: Visualization of the Exponential $\times$ Periodic kernel function.

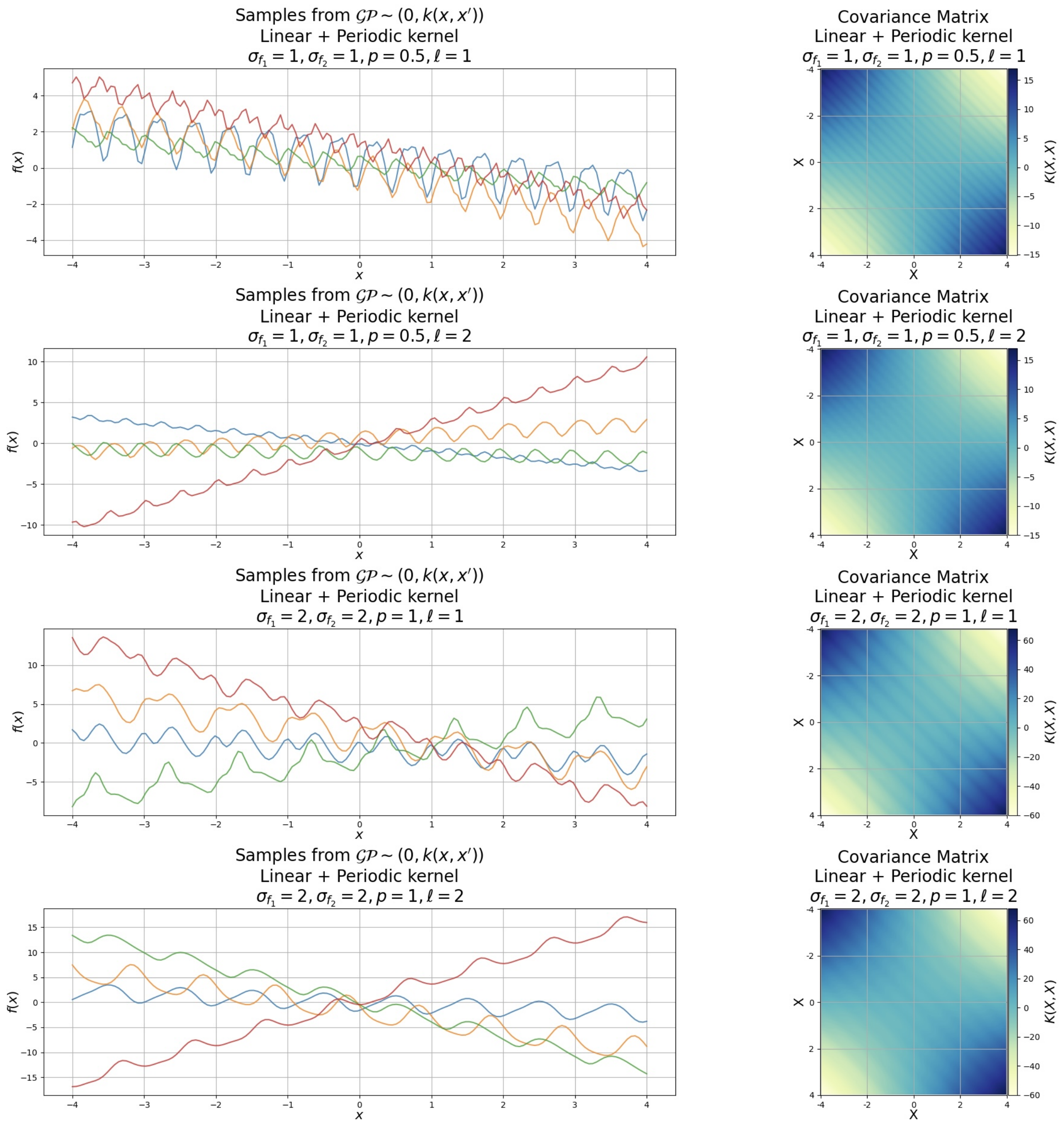
[12]The covariance matrix was factorized using Cholesky decomposition.



(a) (Left) Samples from a MVN [13]with zero mean and Linear × Periodic kernel for various values of the parameters σ_f, p, and ℓ. (Right) Covariance matrices corresponding to the GP samples drawn on the left.

Figure 5.18: Visualization of the Linear × Periodic kernel function.

[13]The covariance matrix was factorized using Cholesky decomposition.



(a) (Left) Samples from a MVN [14]with zero mean and Linear + Periodic kernel for various values of the parameters σ_{f_1}, σ_{f_2}, p, and ℓ. (Right) Covariance matrices corresponding to the GP samples drawn on the left.

Figure 5.19: Visualization of the Linear + Periodic kernel function.

[14]The covariance matrix was factorized using Cholesky decomposition.

5.8 Computational cost and limitations

Gaussian processes demonstrate the remarkable versatility of constructing models and managing uncertainty using the unassuming Gaussian distribution. In the realm of machine learning, they find primary application in modeling functions that are computationally expensive. Nevertheless, their utility extends across a diverse range of applications in various fields. It is important to acknowledge that the primary drawback associated with straightforward Gaussian process models is their limited scalability when confronted with large datasets. More specific [53]:

- The $\mathcal{O}(N^3)$ computational cost usually takes the blame, required to factor the covariance matrix or do the covariance matrix inversion, so that we can evaluate the marginal likelihood and make predictions. The Cholesky approach is noticeable more numerically stable.
- That cost is not the only story. Computing the kernel matrix costs $\mathcal{O}(DN^2)$ and uses $\mathcal{O}(N^2)$ memory. Sometimes the covariance computations can be significant, and running out of memory places a hard limit on problem sizes.

There is a large literature on special cases and approximations of GPs that can be evaluated more efficiently.

It is crucial to recognize that Gaussian processes are not universally capable of representing all functions. For example, the probability of a function being strictly monotonic under any Gaussian process is essentially zero. Additionally, Gaussian processes serve as valuable components in constructing various other models. However, when our observation process deviates from a Gaussian distribution or when the underlying noise is not assumed to be Gaussian, conducting inference becomes a more complex task, necessitating additional effort and approximation techniques.

5.9 Multi-output Gaussian Process Regression

In the analysis above, the GPR allows functions with scalar outputs. For the extension to vector-valued outputs, multiple approaches exist [46]:

(i) Extending the kernel to multivariate outputs

(ii) Adding the output dimension as training data

(iii) Using separated GPR for each output

While the first two approaches set a prior on the correlation between the output dimensions, the latter disregards a correlation without loss of generality. Following the approach in (iii), let each target $\mathbf{y}$ be a vector of dimension M, then the previous definition of the training set $\mathcal{D}$ is extended to a vector-valued output with $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N) \in \mathbb{R}^{D\times N}$ and $\mathbf{Y} = (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N)^T \in \mathbb{R}^{N\times M}$. We write $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1, \dots, N\}$ or just $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$. In that case, we define the vector-valued GP as follows [46]

$$\boldsymbol{f}_{\text{GP}}(\mathbf{x}) \sim \begin{cases} \mathcal{GP}\left(m_1(\mathbf{x}), k_1\left(\mathbf{x}, \mathbf{x}'\right)\right) \\ \mathcal{GP}\left(m_2(\mathbf{x}), k_2\left(\mathbf{x}, \mathbf{x}'\right)\right) \\ \vdots \\ \mathcal{GP}\left(m_M(\mathbf{x}), k_M\left(\mathbf{x}, \mathbf{x}'\right)\right) \end{cases} \tag{5.65}$$

6 Decision trees

One of the most effective approaches in the realm of **supervised learning**, catering to both **classification** and **regression** tasks, is the decision tree. This method constructs a tree-like structure resembling a flowchart, where each internal node signifies a test performed on an attribute, each branch represents a test outcome, and each leaf node, also known as a terminal node, corresponds to a class label or a continuous value. This construction process entails repeatedly partitioning the training data into subsets based on attribute values, with stopping criteria such as the maximum tree depth or the minimum number of samples required for node splitting [54].

6.1 Types of decision trees

Tree-based algorithms form a prominent group of closely related techniques in non-parametric supervised learning, applicable to both regression and classification tasks. For those unfamiliar with supervised learning, it is a subset of machine learning algorithms that involve model development using labeled data, comprising both input and output values. Decision trees can be categorized into two types based on their target variables [55]:

- **Categorical variable decision trees**: These are employed when the algorithm aims to predict a categorical target variable. For instance, consider predicting the relative price of a computer, categorized as low, medium, or high. Features like monitor type, speaker quality, RAM, and SSD may be considered. The decision tree learns from these features, guiding each data point through various nodes until it reaches a leaf node corresponding to one of the three categorical targets: low, medium, or high.
- **Continuous variable decision trees**: In this scenario, the features provided as input to the decision tree, such as the characteristics of a house, are used to predict a continuous output, such as the price of the house.

6.2 Structure of a decision tree

Let's explore the appearance and operational principles of a decision tree when making predictions with new input data [55]. Figure 6.1 provides a visual representation illustrating the fundamental structure of a decision tree. Each decision tree starts with a root node that serves as the entry point for incoming inputs. This root node subsequently branches into groups of decision or internal nodes, which make conditional assessments based on various outcomes and observations. This process of segmenting a single node into multiple nodes is known as **splitting**. Conversely, if a node doesn't further split, it is referred to as a "leaf node" or "terminal node". A segment of a decision tree is commonly referred to as a "branch" or "sub-tree", as exemplified by the grey box in Figure 6.1.

Another concept worth considering is the opposite of splitting, where we remove decision rules from the tree when necessary. This process is referred to as **pruning** and serves the purpose of reducing the algorithm's complexity or preventing overfitting [55].

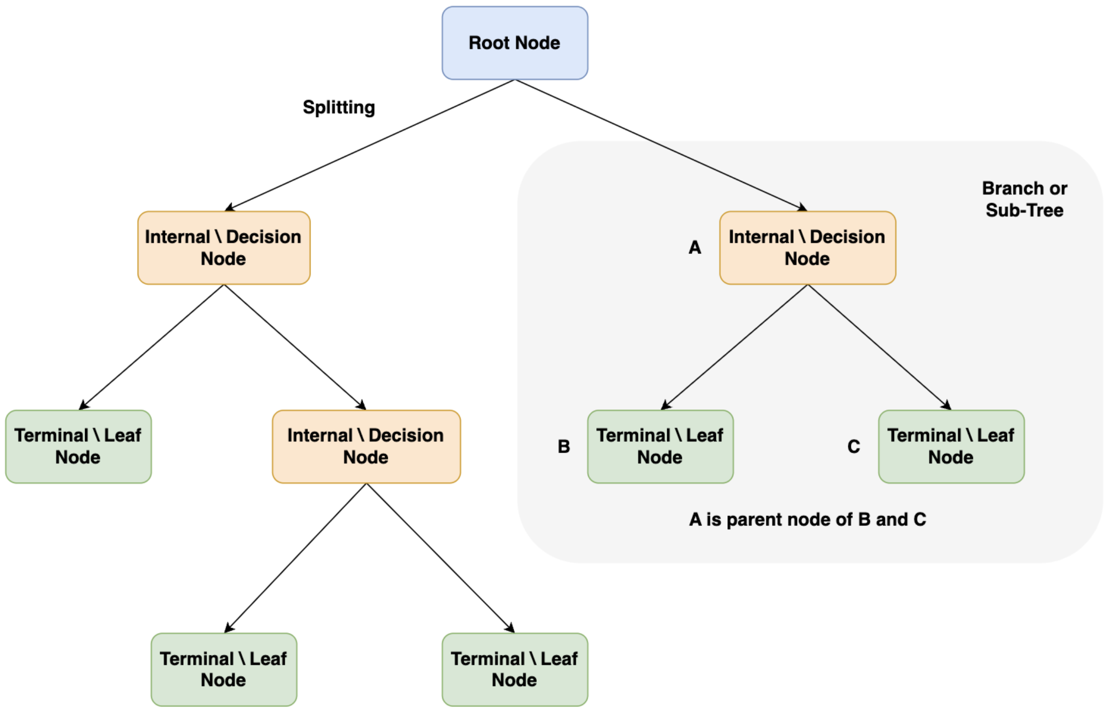


Figure 6.1: Basic structure of a decision tree.

6.3 Terminology in decision trees

Here are some commonly used terms in decision trees [56]:

- **Root node**: This represents the topmost node in the tree, encompassing the entire dataset. It serves as the starting point for the decision-making process.
- **Decision or internal node**: These nodes symbolize choices regarding input features. Internal nodes branch off, connecting to leaf nodes or other internal nodes.
- **Leaf or terminal node**: These nodes have no child nodes and indicate a class label (for classification) or a numerical value (for regression).
- **Splitting**: This is the process of dividing a node into two or more sub-nodes using a split criterion and a chosen feature.
- **Branch or sub-tree**: A subsection of the decision tree that starts at an internal node and ends at leaf nodes.
- **Parent node**: The node that divides into one or more child nodes.
- **Child node**: Nodes that emerge when a parent node is split.
- **Impurity**: A measure of how homogeneous the target variable is in a subset of data. It reflects the level of randomness or uncertainty in a set of examples. Common impurity measurements in decision trees for classification tasks include the Gini index and entropy.

- **Variance**: Variance measures the variation between predicted and target variables in different data samples. It is used in regression problems within decision trees. Measures like Mean Squared Error (MSE), Mean Absolute Error (MAE), Friedman MSE, or Half Poisson Deviance are employed to gauge variance in regression tasks.
- **Information Gain**: Information gain quantifies the reduction in impurity achieved by splitting a dataset based on a particular feature in a decision tree. The feature that provides the greatest information gain determines the splitting criterion. Information gain is utilized to identify the most informative feature for splitting at each node, with the aim of creating purer subsets.
- **Pruning**: Pruning is the process of removing branches from the tree that do not contribute additional information or may lead to overfitting.

Next, we will focus on **regression trees** and the basic algorithm for constructing them.

6.4 Theoretical background of regression trees

In this subsection, the analysis has been extracted from [1].

> Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (like a constant) in each one. They are conceptually simple yet powerful. Next, we will describe a popular method for tree-based regression and classification called CART (Classification And Regression Tree).
>
> Let's consider a regression problem with continuous response Y and inputs X_1 and X_2, each taking values in the unit interval. The top-left panel of Figure 6.2 shows a partition of the feature space by lines that are parallel to the coordinate axes. In each partition element, we can model Y with a different constant. However, there is a problem: although each partitioning line has a simple description like $X_1 = c$, some of the resulting regions are complicated to describe.
>
> To simplify matters, we restrict attention to recursive binary partitions like that in the top-right panel of Figure 6.2. We first split the space into two regions and model the response by the mean of Y in each region. We choose the variable and split-point to achieve the best fit. Then one or both of these regions are split into two more regions, and this process is continued until some stopping rule is applied. For example, in the top right panel of Figure 6.2, we first split at $X_1 = t_1$. Then the region $X_1 \leq t_1$ is split at $X_2 = t_2$ and the region $X_1 > t_1$ is split at $X_1 = t_3$. Finally, the region $X_1 > t_3$ is split at $X_2 = t_4$. The result of this process is a partition into the five regions $R_1, R_2, \ldots, R_5$ shown in the Figure 6.2. The corresponding regression model predicts Y with a constant c_m in region R_m, that is
>
> $$\hat{f}(\mathbf{X}) = \sum_{m=1}^{5} c_m I\{(X_1, X_2) \in R_m\} \tag{6.1}$$

This same model can be represented by the binary tree in the bottom left panel of Figure 6.2. The full dataset sits at the top of the tree. Observations satisfying the condition at each junction are assigned to the left branch, and the others to the right branch. The terminal nodes or leaves of the tree correspond to the regions $R_1, R_2, \ldots, R_5$. The bottom-right panel of Figure 6.2 is a perspective plot of the regression surface from this model. For illustration, we chose the node means $c_1 = -5$, $c_2 = -7$, $c_3 = 0$, $c_4 = 2$, $c_5 = 4$ to make this plot.

A key advantage of the recursive binary tree is its interpretability. The feature space partition is fully described by a single tree. With more than two inputs, partitions like that in the top right panel of Figure 6.2 are difficult to draw, but the binary tree representation works in the same way. This representation is also popular among medical scientists, perhaps because it mimics the way that a doctor thinks. The tree stratifies the population into strata of high and low outcome, on the basis of patient characteristics.

6.5 Basic regression trees

In this subsection, the analysis has been extracted from [1].

We now turn to the question of how to grow a regression tree. Our data consists of p inputs and a response, for each of N observations: that is, $(\mathbf{x}_i, y_i)$ for $i = 1, 2, \ldots, N$, with $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^T$. The algorithm needs to automatically decide on the splitting variables and split points, and also what topology (shape) the tree should have. Suppose first that we have a partition into M regions $R_1, R_2, \ldots, R_M$, and we model the response as a constant c_m in each region:

$$f(\mathbf{x}) = \sum_{m=1}^{M} c_m I(\mathbf{x} \in R_m) \tag{6.2}$$

If we adopt as our criterion the minimization of the sum of squares $(y_i - f(\mathbf{x}_i))^2$, it is easy to see that the best $\hat{c}_m$ is just the average of y_i in region R_m:

$$\hat{c}_m = \operatorname{avg}(y_i | \mathbf{x}_i \in R_m) \tag{6.3}$$

Now finding the best binary partition in terms of minimum sum of squares is generally computationally infeasible. Hence, we proceed with a greedy algorithm. Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j, s) = \{\mathbf{X} | X_j \le s\} \tag{6.4}$$

$$R_2(j, s) = \{\mathbf{X} | X_j > s\} \tag{6.5}$$

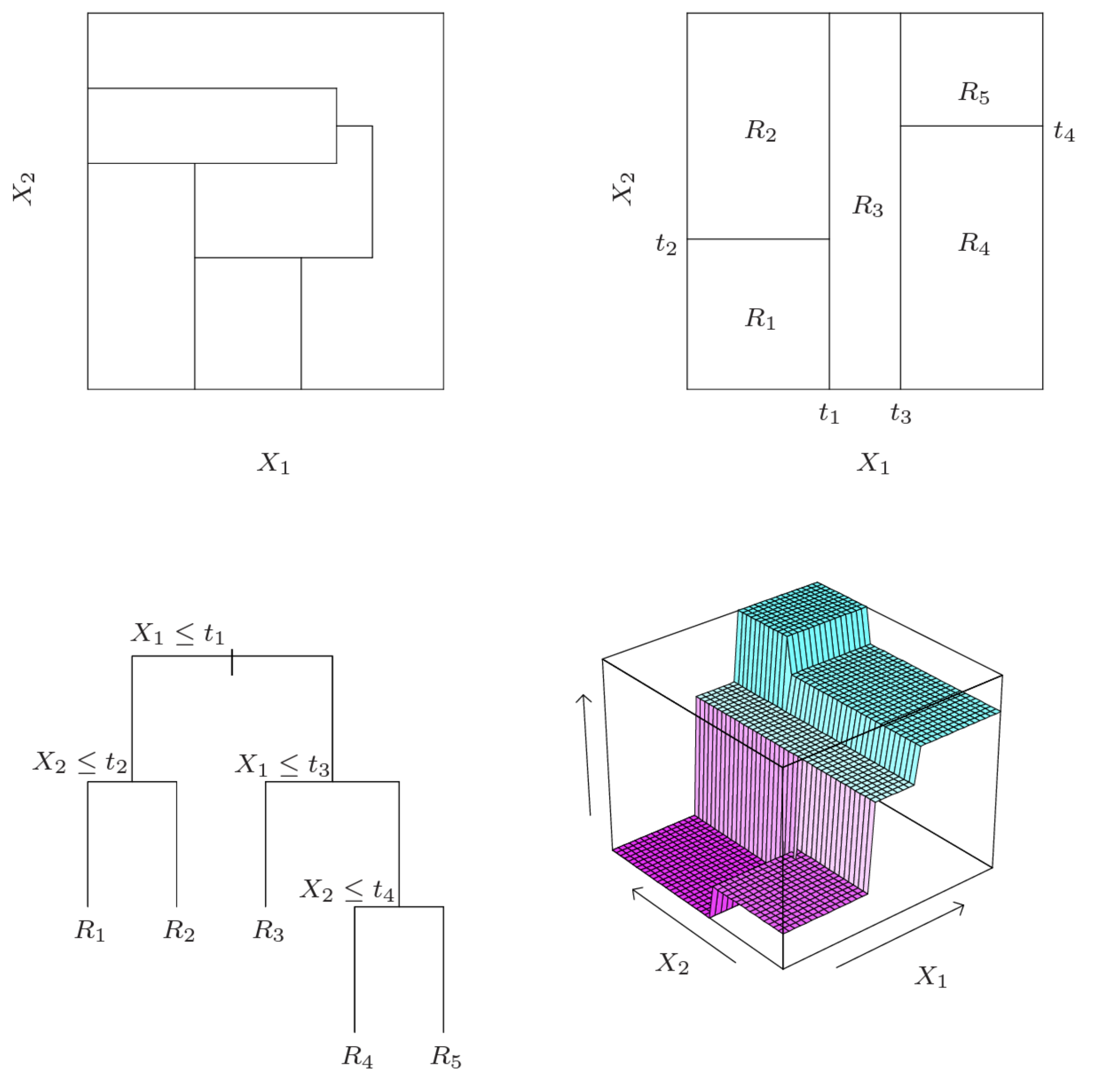


Figure 6.2: Partitions and CART. Top right panel shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some synthetic data. Top left panel shows a general partition that cannot be obtained from recursive binary splitting. Bottom left panel shows the tree corresponding to the partition in the top right panel, and a perspective plot of the prediction surface appears in the bottom right panel. Credit: [1].

Then we seek the splitting variable j and split point s that solve:

$$\min_{j,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2\right] \tag{6.6}$$

For any choice j and s, the inner minimization is solved by

$$\hat{c}_1=\operatorname{avg}(y_i|\mathbf{x}_i\in R_1(j,s)) \tag{6.7}$$

$$\hat{c}_2 = \text{avg}(y_i|\mathbf{x}_i \in R_2(j,s)) \tag{6.8}$$

For each splitting variable, the determination of the split point s can be done very quickly and hence by scanning through all of the inputs, determination of the best pair (j, s) is feasible.

Having found the best split, we partition the data into the two resulting regions and repeat the splitting process on each of the two regions. Then this process is repeated on all of the resulting regions.

How large should we grow the tree? Clearly, a **very large tree** might **overfit** the data, while a small tree might not capture the important structure. Tree size is a tuning parameter governing the model's complexity, and the optimal tree size should be adaptively chosen from the data. One approach would be to split tree nodes only if the decrease in sum-of-squares due to the split exceeds some threshold. This strategy is too short-sighted, however, since a seemingly worthless split might lead to a very good split below it. The preferred strategy is to grow a large tree T_0, stopping the splitting process only when some minimum node size is reached. Then this large tree is pruned using **cost-complexity pruning**, which we now describe.

We define a subtree $T \subset T_0$ to be any tree that can be obtained by pruning T_0, that is, collapsing any number of its internal (non-terminal) nodes. We index terminal nodes by m, with node m representing region R_m. Let $|T|$ denote the number of terminal nodes in T. Letting

$$N_m = \#\{\mathbf{x}_i \in R_m\} \tag{6.9}$$

$$\hat{c}_m = \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} y_i \tag{6.10}$$

$$Q_m(T) = \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} (y_i - \hat{c}_m)^2 \tag{6.11}$$

we define the cost complexity criterion

$$C_a(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T| \tag{6.12}$$

The idea is to find, for each α, the sub-tree $T_\alpha \subseteq T_0$ to minimize $C_a(T)$. The tuning parameter $\alpha \geq 0$ governs the trade-off between tree size and its goodness of fit to the data. Large values of α result in smaller trees T_α, and conversely for smaller values of α. As the notation suggests, with $\alpha = 0$ the solution is the full tree T_0. We discuss how to adaptively choose α below. For each α, one can show that there is a unique smallest sub-tree T_α that minimizes $C_a(T)$. To find T_α, we use **weakest link pruning**: we successively collapse the internal node that produces the smallest per-node increase in $\sum_m N_m Q_m(T)$, and continue until we produce the single-node (root) tree. This gives a (finite) sequence of sub-trees, and one can show this sequence must contain T_α. See Breiman *et al.* (1984) [57] or Ripley (1996) [58] for details. Estimation of α is achieved by five- or ten-fold cross-validation: we choose the value $\hat{\alpha}$ to minimize the cross-validated sum of squares. Our final tree is $T_{\hat{\alpha}}$.

6.6 Bagging vs. Boosting

6.6.1 Ensemble learning methods

Ensemble learning models are rooted in the concept that the fusion of multiple models can yield formidable results. This technique harnesses the power of multiple models, often termed "weak learners", to achieve outcomes that surpass those of individual constituent models in terms of performance, stability, and predictive accuracy [59]. Machine learning predictions and classification errors typically stem from factors such as noise, bias, and variance. Ensemble learning techniques are specifically designed to mitigate these issues and enhance the overall robustness and reliability of the model.

Ensemble learning techniques can be applied to tackle both **classification** and **regression** problems, consistently demonstrating superior performance compared to other machine learning approaches. This superiority arises from the fact that the final prediction is derived by amalgamating results from numerous models [59]. While there are countless possibilities for creating ensembles in predictive modeling, two techniques stand out as predominant in ensemble learning: **bagging** and **boosting**. In the following sections, we will delve into the underlying principles of bagging and boosting techniques.

6.6.2 Bagging

Bootstrap aggregating, commonly referred to as bagging, is an ensemble learning method designed to enhance the stability and accuracy of machine learning algorithms employed in statistical classification and regression tasks. It addresses issues such as variance reduction and overfitting that can arise in individual models. This technique is typically applied in the context of decision tree algorithms.

Bagging achieves its goal by constructing an ensemble of diverse learners through variations in the training dataset. Instead of training a single model on the entire dataset, bagging generates multiple weak learners or base models by training them on subsets of the original data. The specifics of the ensemble, including the number of models and the size of the subsets, are determined by the data scientist developing the model. The subsets of data utilized to train the weak learners are formed through random sampling with replacement for each new model trained. This implies that a data subset for training may contain duplicate entries. Bagging employs similar learners on small sample populations of the training data, and subsequently aggregates the predictions of these learners. For classification problems, the final model output is determined through majority voting, whereas for regression problems, it involves computing an average of the predictions. The steps involved in bagging are the following [59]:

1. **Subset creation**: Multiple subsets are generated from the training dataset by selecting observations with replacement, a technique known as bootstrapping. Each subset represents a random sample from the original data.

2. **Base model generation**: A base model, often referred to as a weak model, is constructed separately for each subset. These base models are trained independently on their respective subsets.

3. **Parallel processing**: The base models operate concurrently and independently of each other. This parallelization allows for efficient model training on different data samples.

4. **Predictive aggregation**: The predictions made by all the base models are aggregated or combined to determine the final outcome. This aggregation process can involve methods such as averaging or majority voting, depending on the specific problem.

Figure 6.3 shows a diagrammatic representation of the bagging process.

In essence, bagging mitigates variance-related errors and significantly improves the overall accuracy of the model. By introducing diversity in the training process and leveraging the collective insights of multiple models, bagging ensures a more robust and reliable ensemble learning approach.

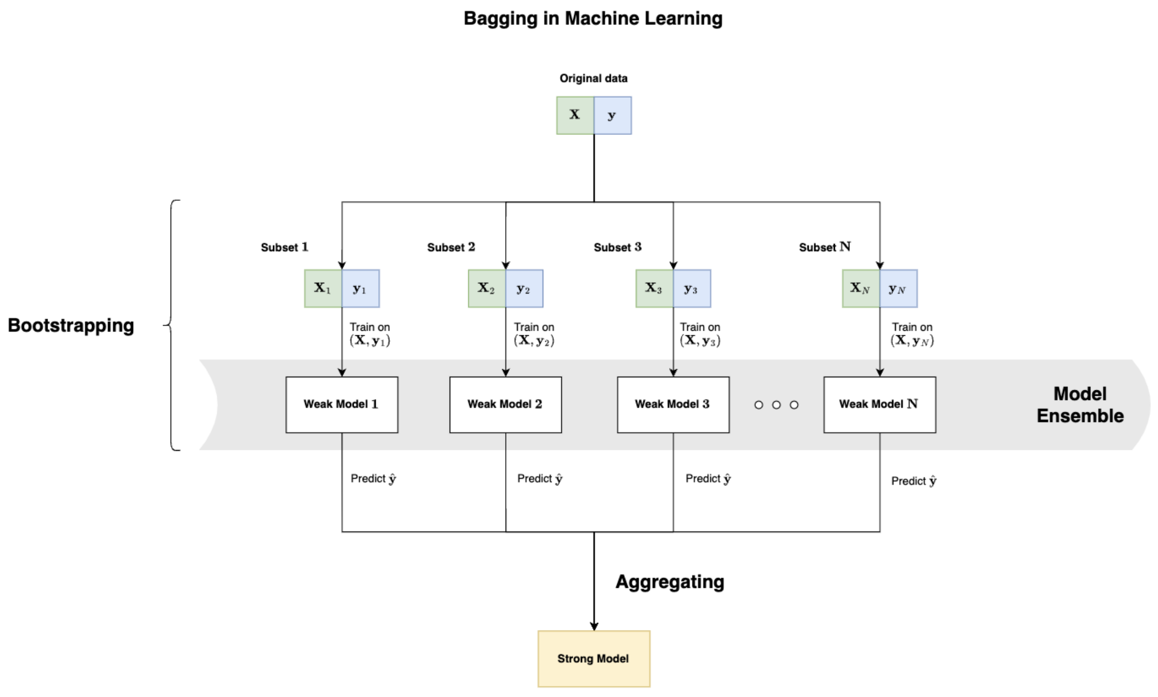

Bagging in Machine Learning
Original data
$\mathbf{X}$
$\mathbf{y}$
Bootstrapping
Subset 1
Subset 2
Subset 3
Subset N
$\mathbf{X}_1$ $\mathbf{y}_1$
$\mathbf{X}_2$ $\mathbf{y}_2$
$\mathbf{X}_3$ $\mathbf{y}_3$
$\mathbf{X}_N$ $\mathbf{y}_N$
Train on $(\mathbf{X}, \mathbf{y}_1)$
Train on $(\mathbf{X}, \mathbf{y}_2)$
Train on $(\mathbf{X}, \mathbf{y}_3)$
Train on $(\mathbf{X}, \mathbf{y}_N)$
Weak Model 1
Weak Model 2
Weak Model 3
Weak Model N
Model Ensemble
Predict $\hat{y}$
Predict $\hat{y}$
Predict $\hat{y}$
Predict $\hat{y}$
Aggregating
Strong Model

Figure 6.3: Diagrammatic representation of the bagging process.

6.6.3 Boosting

Boosting stands as an ensemble modeling technique that aims to create a robust model by integrating a multitude of weak models. This process involves sequentially building models using weak models. The initial step entails constructing a model using the training data. Next, each subsequent model is developed to rectify errors present within the previous model's predictions. This iterative approach continues until either the entire training dataset is accurately predicted or the maximum allowable number of models is reached.

Distinct from the bagging methodology, boosting is an ensemble technique that brings alterations to the training data and dynamically adjusts the importance of observations based on previous predictions. In contrast to the independent nature of weak learners in bagging, boosting introduces a sense of interdependence among them. Each weak learner takes into consideration the outcomes of its predecessor and adapts the weights assigned to data points. This transformative process works to elevate the weak learners into a realm of strong learners. Following are the steps involved in the boosting technique [55]:

1. **Equal weight subset**: A subset is created from the training dataset where each data point is assigned equal weight. This initial subset serves as the starting point for the boosting process.

2. **Base model creation**: A base model is constructed using the initial dataset, and this model is used to make predictions on the entire dataset.

3. **Error calculation**: Errors are computed by comparing the predicted values from the base model with the actual values. Any observation that the model predicts incorrectly is assigned a higher weight, emphasizing the misclassified data points.

4. **Next model creation**: A new model is created as part of the boosting process, with the goal of correcting the errors made by the previous model. This new model is designed to focus on the observations that the prior model misclassified.

5. **Iterative error correction**: The boosting process continues iteratively, with each subsequent model built to rectify the errors made by the previous model. This iterative approach aims to gradually improve the overall predictive performance.

6. **Final model**: The final model that emerges from this process is a robust and strong learner. It's essentially the weighted mean or combination of all the models (weak learners) created during the boosting process.

Figure 6.4 shows a diagrammatic representation of the boosting process.

Boosting orchestrates adjustments to observation weights, particularly those that were incorrectly predicted, by enhancing their importance. This emphasis on the incorrectly predicted observations aims to rectify their subsequent prediction. While boosting serves to diminish bias-related errors, it does bear the potential of inducing overfitting within the training dataset. This duality underscores the necessity of striking a balance during the boosting process, as the technique's endeavors to enhance accuracy can inadvertently lead to an excessive specialization to the training data.

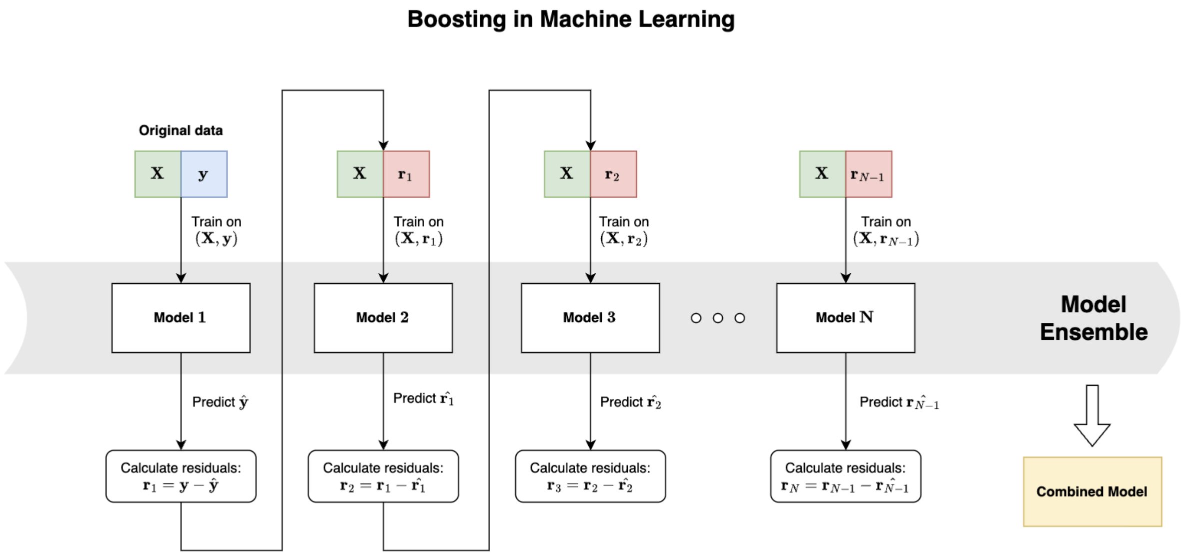

Boosting in Machine Learning
Original data
X
y
Train on $(\mathbf{X}, \mathbf{y})$
Model 1
Predict $\hat{\mathbf{y}}$
Calculate residuals: $\mathbf{r}_1 = \mathbf{y} - \hat{\mathbf{y}}$
X
$\mathbf{r}_1$
Train on $(\mathbf{X}, \mathbf{r}_1)$
Model 2
Predict $\hat{\mathbf{r}_1}$
Calculate residuals: $\mathbf{r}_2 = \mathbf{r}_1 - \hat{\mathbf{r}_1}$
X
$\mathbf{r}_2$
Train on $(\mathbf{X}, \mathbf{r}_2)$
Model 3
Predict $\hat{\mathbf{r}_2}$
Calculate residuals: $\mathbf{r}_3 = \mathbf{r}_2 - \hat{\mathbf{r}_2}$
X
$\mathbf{r}_{N-1}$
Train on $(\mathbf{X}, \mathbf{r}_{N-1})$
Model N
Predict $\hat{\mathbf{r}_{N-1}}$
Calculate residuals: $\mathbf{r}_N = \mathbf{r}_{N-1} - \hat{\mathbf{r}_{N-1}}$
Model Ensemble
Combined Model

Figure 6.4: Diagrammatic representation of the boosting process.

6.6.4 Comparison

Now that we've gained a foundational understanding of both algorithms, let's delve into the distinctions and commonalities between these two methods by drawing comparisons. Here are some of the similarities between bagging and boosting:

- **Multiple learners**: Both bagging and boosting ensemble techniques create an ensemble of multiple learners from a single learner.
- **Random sampling**: Both techniques employ random sampling to generate multiple training datasets, introducing diversity into the learning process.
- **Combining learner outputs**: In both bagging and boosting, the final prediction is typically made by averaging the results of the individual learners or through majority voting.
- **Reducing variance**: Bagging and boosting serve to decrease model variance and enhance the stability of the models.

Let's explore some of the differences between bagging and boosting algorithms:

- **Nature of combination**: Bagging and boosting may both involve N learners, but they fundamentally differ in how they combine predictions. Bagging combines predictions from the same type of models, whereas boosting combines predictions from different types of models.
- **Parallel vs. sequential**: Bagging operates as a parallel ensemble learning method, while boosting is a sequential ensemble learning method, with each learner built on the insights of its predecessor.
- **Independence vs. interaction**: In bagging, each model is constructed independently of the others, with no interaction between them. In contrast, boosting relies on the results of previously built models to influence the creation of new models, creating a sequential and interdependent process.

- **Weighting**: Bagging assigns equal weight to each model in the ensemble, treating them uniformly. In the boosting technique, the contribution of new models is weighted based on their performance, with better-performing models receiving greater importance.

- **Training data subsets**: In boosting, new subsets of training data are formed by including observations that the previous model misclassified, aiming to correct errors. Bagging, on the other hand, uses randomly generated training data subsets without considering previous model performance.

- **Effect on variance and bias**: Bagging primarily aims to reduce variance, making it effective at mitigating overfitting. In contrast, boosting is designed to reduce bias, addressing issues related to underfitting.

6.7 Random forests

6.7.1 Architecture of random forests

Random forests, also known as random decision forests, stand as a robust **ensemble learning** algorithm employed for a range of tasks such as classification and regression. This method implements the **bagging** technique and operates by constructing a multitude of decision trees during the training phase. In essence, a random forest is an ensemble of decision trees brought together in a distinctive manner [60].

For classification endeavors, the output generated by the random forest corresponds to the class that the majority of trees have chosen. Conversely, when dealing with regression tasks, the final prediction is derived from the mean or average predictions of the individual trees. This approach addresses a significant pitfall of decision trees, which tend to overfit their training data. Random decision forests mitigate this by introducing randomness and variety into the process.

A key feature of the random forest is that it comprises multiple decision trees, and each tree is constructed from a distinct subset of the data rows. Additionally, at each node of these trees, a distinct subset of features is chosen for the splitting process. This diversity in data samples and feature selection is pivotal to the success of a random forest. Each of these trees generates its own prediction, and these individual predictions are then aggregated through averaging [61].

This aggregation process is what sets random forests apart from a single decision tree, rendering them more accurate and less prone to overfitting. This is particularly evident in the context of the random forest regressor, where the final prediction is obtained as an average of the predictions produced by all trees within the forest. While it is noteworthy that random forests often exhibit superior performance compared to individual decision trees, their accuracy can fall short of that achieved by gradient boosted trees. Nonetheless, it is important to recognize that the performance of random forests is influenced by the characteristics of the data being processed.

6.7.2 Hyperparamaters of random forests

When writing code regarding a Random Forest model, the following are some of the most common **hyperparameters** that someone has to tune:

- **n_estimators**: This hyperparameter specifies the number of decision trees to be included in the random forest. Increasing n_estimators typically improves the model's performance, but it can also lead to longer training times.

- **max_depth**: It controls the maximum depth of each individual decision tree in the forest. Setting a higher value allows trees to be deeper, which can lead to overfitting if not carefully tuned.

- **min_samples_split**: This hyperparameter determines the minimum number of samples required to split an internal node during the tree-building process. A lower value makes the trees more sensitive to noise and can result in overfitting.

- **min_samples_leaf**: Specifies the minimum number of samples required to be in a leaf node. Similar to min_samples_split, a lower value increases the risk of overfitting.

- **max_features**: Determines the maximum number of features to consider when making a split decision. You can set it as a fixed number or a fraction of the total number of features. It introduces randomness and helps prevent overfitting.

- **bootstrap**: A binary parameter that indicates whether the training data should be bootstrapped (sampled with replacement). It controls whether each tree in the forest is trained on a different subset of the data.

- **random_state**: Set for reproducibility. It controls the random seed for random number generation during the construction of the trees.

- **criterion**: Specifies the function used to measure the quality of a split. Common options include "gini" for Gini impurity and "entropy" for information gain.

- **oob_score**: A binary parameter indicating whether to use out-of-bag samples to estimate the model's accuracy. Out-of-bag samples are data points that were not included in the bootstrapped training set for each tree.

- **n_jobs**: Specifies the number of CPU cores to use during training. Setting it to -1 will use all available cores.

- **warm_start**: A binary parameter that allows you to reuse the existing model and continue training with additional estimators. Useful for incremental learning.

6.8 Adaptive Boosting (AdaBoost)

6.8.1 Architecture of AdaBoost

The initial breakthrough in **boosting** algorithms that achieved remarkable practical success was realized through Adaptive Boosting, often abbreviated as AdaBoost. This technique involves fitting a series of weak learners to the dataset, subsequently according greater weight to incorrect predictions while assigning relatively lesser weight to the accurate ones. This strategic weighting mechanism allows the algorithm to concentrate its efforts on predicting the more challenging observations. Ultimately, the conclusive outcome is derived from a majority vote in classification tasks or an averaging process in regression scenarios [62].

The default approach of the algorithm involves employing decision trees as foundational estimators, which are considered weak learners. These decision trees consist of only one split, often referred to as **decision stumps** due to their concise nature. Both the foundational estimators and the decision tree's parameters are adjustable, offering an avenue for enhancing the overall model's performance through careful tuning.

6.8.2 Hyperparamaters of AdaBoost

When writing code regarding an AdaBoost model, the following are some of the most common **hyperparameters** that someone has to tune:

- **n_estimators**: This hyperparameter specifies the number of weak learners (usually decision trees) to be combined. Increasing n_estimators typically improves model performance, but it can also lead to longer training times.

- **base_estimator**: AdaBoost can work with different types of base classifiers. By default, it uses a decision tree with a depth of 1 (a stump), but you can specify a different base estimator, such as a decision tree with greater depth or even other types of classifiers.

- **learning_rate**: This hyperparameter controls the contribution of each weak learner to the final ensemble. Lower values make the learning process more gradual, and the model may require more weak learners to reach good performance.

- **loss**: Specifies the loss function to be optimized during training. For regression tasks, common options include "linear" (least squares loss) and "exponential" (exponential loss).

- **random_state**: Set for reproducibility. It controls the random seed for random number generation during training.

- **n_jobs**: Specifies the number of CPU cores to use during training. If set to -1, it will use all available cores.

6.9 Gradient boosting

Gradient boosting is one of the most powerful techniques for building predictive models. **Gradient boosting** involves three elements [62]:

1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.

6.9.1 Loss function

The choice of the loss function relies on the nature of the problem under consideration. It is necessary for the function to be mathematically smooth (differentiable), and while there are established loss functions available, you also have the flexibility to create your own. To illustrate, in regression tasks, a squared error could be adopted, while in classification problems, logarithmic loss might be more appropriate. An advantage of the gradient boosting framework is that you don't need to develop a new boosting technique for every potential loss function. Instead, it provides a versatile framework that can accommodate any loss function with smooth mathematical properties [62].

6.9.2 Weak learner

Gradient boosting employs decision trees as its weak learners. More specifically, regression trees are employed. These trees produce real values for splits and their outcomes can be summed up, enabling subsequent model outputs to be combined in order to "adjust" the discrepancies in predictions. The tree-building process is carried out greedily, where the best split points are chosen based on measures of purity like Gini index or to minimize the overall loss [62]. In the initial stages, similar to how it is done in AdaBoost, very short decision trees were utilized. These trees had just one split, often termed a "decision stump". For more comprehensive trees, generally with 4 to 8 levels, they can be adopted. To ensure the weak nature of the learners while still facilitating the greedy construction, certain limitations are often imposed on the weak learners. These restrictions could involve a maximum number of layers, nodes, splits, or leaf nodes. The objective is to maintain the weak character of the learners while allowing their construction in a greedy fashion.

6.9.3 Additive model

The model introduces trees sequentially, with each new tree being **added** individually and the existing trees remaining **unchanged**. A gradient descent technique is applied to minimize the loss while incorporating these trees. In conventional usage, gradient descent is employed to decrease a set of parameters, like the coefficients in a regression equation or the weights in a neural network. Once the error or loss is computed, the weights are adjusted to minimize that error.

However, in this context, instead of dealing with parameters, we work with weak learner sub-models, particularly decision trees. Following the loss calculation, in order to execute the gradient descent procedure, a new tree needs to be incorporated into the model that reduces the loss (essentially following the gradient). To achieve this, we parameterize the tree, then modify its parameters to move in the correct direction (aiming to reduce the residual loss). This approach is generally referred to as functional gradient descent or gradient descent with functions [62].

The outcome generated by the new tree is subsequently combined with the output of the existing sequence of trees, with the aim of rectifying or enhancing the final model output. A predetermined number of trees are included, or the training process halts when the loss attains an acceptable level, or when further improvements on an external validation dataset are no longer observed.

6.9.4 Intuitive introduction

Similar to other boosting techniques, gradient boosting assembles weak learners into a robust single learner through an iterative process. This is best understood within the context of least-squares regression, where the objective is to instruct a model F to predict values of the form $\hat{y} = F(\mathbf{x})$, achieved by minimizing the average squared error $\frac{1}{n}\sum i(\hat{y}i - y_i)^2$. In this equation, the index i runs through a training set of actual output values y of size n. It is:

- $\hat{y}_i$ is the predicted value $F(x_i)$,
- y_i is the observed value, and
- n is the number of samples in $\mathbf{y}$

Let's delve into a gradient boosting algorithm comprising M stages. At each stage m, $1 \leq m \leq M$, of the gradient boosting process, let's assume there's a somewhat imperfect model F_m in place. For lower values of m, this model might simply return $\hat{y}_i = \bar{\mathbf{y}}$, where the right-hand side represents the mean of $\mathbf{y}$. To enhance F_m, our algorithm aims to introduce a new estimator, denoted as $h_m(\mathbf{x})$. Consequently,

$$F_{m+1}(\mathbf{x}_i) = F_m(\mathbf{x}_i) + h_m(\mathbf{x}_i) = y_i \tag{6.13}$$

or, put differently,

$$h_m(\mathbf{x}_i) = y_i - F_m(\mathbf{x}_i) \tag{6.14}$$

Therefore, gradient boosting endeavors to fit h_m to the residual $y_i - F_m(\mathbf{x}_i)$. Just like other versions of boosting, each subsequent F_{m+1} aims to rectify the errors made by its predecessor F_m. This concept's generalization, encompassing loss functions beyond the squared error, as well as tackling classification and ranking problems, stems from the observation that residuals $h_m(\mathbf{x}_i)$ for a given model are proportionate to the negative gradients of the mean squared error (MSE) loss function (in relation to $F(\mathbf{x}_i)$). Specifically, it can be expressed as:

$$L_{\mathrm{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - F(\mathbf{x}_i))^2 \tag{6.15}$$

$$-\frac{\partial L_{\mathrm{MSE}}}{\partial F(\mathbf{x}_i)} = \frac{2}{n}(y_i - F(\mathbf{x}_i)) = \frac{2}{n}h_m(\mathbf{x}_i) \tag{6.16}$$

This underscores the possibility of specializing gradient boosting into a gradient descent algorithm. Furthermore, its extension involves integrating a different loss function and its corresponding gradient.

6.9.5 Gradient boosting algorithm

In numerous cases of supervised learning, there exists a set of input vectors $\mathbf{x}$ and corresponding output variables y, linked by a certain probabilistic distribution. The objective is to determine a function $\hat{F}(\mathbf{x})$ that effectively approximates the output variable based on the input values. This is formalized by introducing a loss function $L(y, F(\mathbf{x}))$ and seeking to minimize its expected value:

$$\hat{F} = \arg\min_{F} \mathbb{E}_{\mathbf{x},y}[L(y, F(\mathbf{x}))] \tag{6.17}$$

The gradient boosting method assumes a real-valued y. It aims to approximate $\hat{F}(\mathbf{x})$ by composing a weighted sum of M functions $h_m(\mathbf{x})$ from a defined class $\mathcal{H}$, referred to as base (or weak) learners:

$$\hat{F}(\mathbf{x}) = \sum_{m=1}^{M} \gamma_m h_m(\mathbf{x}) + \mathrm{const} \tag{6.18}$$

Typically, a known training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ is provided, consisting of observed sample values of $\mathbf{x}$ and their corresponding y values. Adhering to the principle of empirical risk minimization, the method aims to find an approximation $\hat{F}(\mathbf{x})$ that minimizes the average loss function value over the training set, thereby minimizing the empirical risk. It achieves this by initiating with a model comprising a constant function $F_0(\mathbf{x})$, and then progressively expanding it in a step-by-step manner:

$$F_0(\mathbf{x}) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma) \tag{6.19}$$

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \left(\arg\min_{h_m \in \mathcal{H}} \left[\sum_{i=1}^{n} L(y_i, F_{m-1}(\mathbf{x}_i) + h_m(\mathbf{x}_i)) \right] \right) (\mathbf{x}) \tag{6.20}$$

for $m \geq 1$, where $h_m \in \mathcal{H}$ represents a base learner function. However, the optimal selection of the function h_m for an arbitrary loss function L is often computationally unfeasible. Thus, we simplify our approach while preserving the core idea. We apply a steepest descent approach to this optimization problem (functional gradient descent). This involves iteratively modifying $F_{m-1}(\mathbf{x})$ to find a local minimum of the loss function:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) - \gamma \sum_{i=1}^{n} \nabla_{F_{m-1}} L(y_i, F_{m-1}(\mathbf{x}_i)) \tag{6.21}$$

This change is governed by a small positive value γ. For sufficiently small γ, this guarantees that $L(y_i, F_m(\mathbf{x}_i)) \leq L(y_i, F_{m-1}(\mathbf{x}_i))$. Furthermore, we can optimize γ by finding the γ value that minimizes the loss function:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, F_m(\mathbf{x}_i)) = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(\mathbf{x}_i) - \gamma \nabla_{F_{m-1}} L(y_i, F_{m-1}(\mathbf{x}_i))\right) \tag{6.22}$$

In the scenario where we examine the continuous case—meaning that $\mathcal{H}$ encompasses all arbitrary differentiable functions on $\mathbb{R}^d$—the model's update process follows these equations:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) - \gamma_m \sum_{i=1}^{n} \nabla_{F_{m-1}} L(y_i, F_{m-1}(\mathbf{x}_i)) \tag{6.23}$$

In these equations, γ_m represents the step length, and its value is determined by:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(\mathbf{x}_i) - \gamma \nabla_{F_{m-1}} L(y_i, F_{m-1}(\mathbf{x}_i))\right) \tag{6.24}$$

However, in the discrete case, specifically when the set $\mathcal{H}$ is finite, we opt for the candidate function h that is closest to the gradient of L. Then, by using line search on the aforementioned equations, we can compute the coefficient γ. It's important to note that this approach is heuristic, and as a result, it provides an approximation rather than an exact solution to the given problem. In pseudo-code, the generic gradient boosting method is shown in Algorithm 6.1.

6.9.6 Gradient tree boosting

Gradient boosting is commonly employed with decision trees, particularly fixed-size trees like CARTs, used as base learners. In this context, Friedman suggests an enhancement to the gradient boosting method that refines the fit quality of each base learner. In the standard gradient boosting approach, at the m-th step, a decision tree $h_m(\mathbf{x})$ is fitted to pseudo-residuals. Let J_m denote the number of leaves in this tree. The tree divides the input space into J_m distinct regions $R_{1m}, \ldots, R_{J_m m}$ and provides a constant prediction within each region. By using indicator notation, the output of $h_m(\mathbf{x})$ for input $\mathbf{x}$ can be expressed as a sum:

$$h_m(\mathbf{x}) = \sum_{j=1}^{J_m} b_{jm} \mathbf{1}_{R_{jm}}(\mathbf{x}) \tag{6.25}$$

Here, b_{jm} represents the predicted value within region R_{jm}. These coefficients b_{jm} are then multiplied by a value γ_m, determined through line search to minimize the loss function. The model is updated as follows:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \gamma_m h_m(\mathbf{x}) \tag{6.26}$$

Algorithm 6.1: A generic gradient boosting algorithm [5]

```
Input: Training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, loss function $L(y, F(\mathbf{x}))$, and number of
       iterations $M$
Output: Final boosted model $F_M(\mathbf{x})$
1 Initialize the model with a constant value:
      $$F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$$
2 for $m = 1$ to $M$ do
3     Compute so-called pseudo-residuals:
          $$r_{im} = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \quad \text{for} \quad i = 1, \dots, n$$
4     Fit a base learner (or weak learner, e.g. tree) closed under scaling $h_m(\mathbf{x})$ to
        pseudo-residuals, i.e. train it using the training set $\{(\mathbf{x}_i, r_{im})\}_{i=1}^n$
5     Compute multiplier $\gamma_m$ by solving the following one-dimensional optimization
        problem:
          $$\gamma_m = \arg\min_\gamma \sum_{i=1}^{n} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \gamma h_m(\mathbf{x}_i)\right)$$
6     Update the model:
          $$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \gamma_m h_m(\mathbf{x})$$
7 Output the final model: $F_M(\mathbf{x})$
```

$$\gamma_m = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma h_m(\mathbf{x}_i)) \tag{6.27}$$

Friedman proposes a modification to this algorithm, referred to as “TreeBoost”, where instead of a single γ_m for the entire tree, it selects a distinct optimal value γ_{jm} for each region of the tree. Consequently, the coefficients b_{jm} derived from the tree-fitting process can be disregarded, leading to a revised model update rule:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}_{R_{jm}}(\mathbf{x}) \tag{6.28}$$

$$\gamma_{jm} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma) \tag{6.29}$$

6.10 Improvements to basic gradient boosting

Gradient boosting follows a greedy approach, which makes it susceptible to rapid overfitting of a training dataset. To counteract this, regularization techniques can be employed, aiming to penalize specific components of the algorithm. By and large, these methods enhance the algorithm's performance by mitigating overfitting. In this subsection, we will explore four enhancements that can be applied to the fundamental gradient boosting process [62]:

- Tree constraints
- Shrinkage
- Random sampling
- Penalized learning

6.10.1 Tree constraints

It's crucial for the weak learners to possess some level of competence while still maintaining their inherent weakness. There exist various methods to constrain the decision trees. A useful rule of thumb is that the more restricted the process of tree creation, the greater the number of trees necessary in the model. Conversely, if individual trees are subject to fewer constraints, a smaller number of trees will suffice. The subsequent **constraints** can be enforced during the decision tree construction: [62]:

- **Number of trees**: It's generally safe to augment the number of trees in the model, as it's usually slow to lead to overfitting. The guiding principle is to continue adding trees until no further improvement is observed.
- **Tree depth**: Opting for shallower trees is advisable, as deeper trees tend to be more intricate. The range of 4 to 8 levels often yields favorable outcomes.
- **Number of nodes or leaves**: This constraint governs the size of the tree. It doesn't enforce a symmetrical structure unless other constraints are concurrently applied.
- **Number of observations per split**: This constraint sets a minimum requirement for the volume of training data at a particular node before a split can be contemplated.
- **Minimum improvement to loss**: This acts as a restriction on the enhancement achieved by any split that is introduced to a tree.

6.10.2 Weighted updates

The predictions made by each tree are sequentially accumulated. To control the learning pace of the algorithm, the contribution of each tree can be adjusted through weighting. This adjustment is known as **shrinkage** or a **learning rate**. Essentially, this technique slows down the learning process, necessitating the inclusion of more trees in the model. Consequently, the training duration is extended, creating a trade-off between the quantity of trees and the learning rate [62]. It's typical to employ small values within the range of 0.1 to 0.3, and even values below 0.1, for effective results.

6.10.3 Stochastic gradient boosting

A significant revelation in bagging ensembles and random forests was the ability to construct trees greedily using sub-samples from the training dataset. This advantageous concept can also be harnessed to decrease the correlation among trees within the sequence in gradient boosting models. This adaptation of boosting is referred to as stochastic gradient boosting. There are a few different variations of stochastic boosting that can be employed [62]:

- Sub-sample rows before creating each tree.
- Sub-sample columns before creating each tree.
- Sub-sample columns before considering each split.

In general, adopting a more assertive sub-sampling approach, such as opting for only 50% of the available data, has proven to yield advantageous outcomes.

6.10.4 Penalized gradient boosting

Supplementary restrictions can be applied to the parameterized trees aside from defining their structure. In the context of gradient boosting, classical decision trees like CART are replaced with a modified version known as a regression tree. These regression trees have numeric values assigned to their leaf nodes, which are sometimes referred to as weights in certain literature. Consequently, the leaf weight values of these trees can be subjected to regularization using well-known regularization functions, such as: [62]:

- L1 regularization of weights.
- L2 regularization of weights.

6.11 Extreme Gradient Boosting (XGBoost)

XGBoost, short for Extreme Gradient Boosting, represents a resilient machine-learning algorithm [63]. It stands as a parallelized and meticulously optimized variant of the gradient boosting algorithm. The parallelization of the entire boosting process significantly enhances training efficiency. Instead of aiming to construct the best possible model using the entire dataset, as is common in traditional methods, XGBoost takes a different approach. It trains numerous models on diverse subsets of the training dataset and subsequently selects the best-performing model through a voting mechanism [64]. In many instances, XGBoost surpasses conventional gradient boosting algorithms in terms of performance. It is worth noting that in XGBoost, the **layer-wise** tree growth strategy is being used.

6.11.1 Features of XGBoost

Now, we will discuss some features of XGBoost that make it a so interesting and widespread implementation of gradient boosting [64, 65]:

- **Regularization**: XGBoost provides the option to penalize complex models using both L1 and L2 regularization methods. This regularization aids in preventing overfitting.
- **Handling sparse data**: Dealing with sparse data, which may result from missing values or data processing steps like one-hot encoding, is a crucial aspect. XGBoost employs a sparsity-aware split finding algorithm to effectively manage various types of sparsity patterns within the data.
- **Weighted quantile sketch**: Unlike most existing tree-based algorithms, which typically handle data of equal weights using quantile sketch algorithms, XGBoost is equipped to handle weighted data effectively and find split points through its distributed weighted quantile sketch algorithm.
- **Block structure for parallel learning**: XGBoost can leverage multiple CPU cores for faster computation. This is achievable due to a block structure in its system design. Data is organized and stored in in-memory units called blocks. This approach enables the reuse of data layout by subsequent iterations, as opposed to recomputing it. It also aids in tasks like split finding and column sub-sampling.
- **Cache awareness**: XGBoost optimizes memory access for gradient statistics by row index, which often involves non-continuous memory access. To achieve this, XGBoost allocates internal buffers in each thread for storing gradient statistics, thus making optimal use of hardware.
- **Out-of-core computing**: This feature is designed to handle large datasets that don't fit into memory efficiently. It optimizes disk space usage while processing extensive datasets.
- **Non-linearity**: XGBoost is capable of detecting and learning from non-linear patterns within the data, making it adaptable to a wide range of scenarios.
- **Scalability**: XGBoost can run in distributed environments using servers and clusters like Hadoop and Spark. This scalability enables the processing of vast amounts of data. Moreover, XGBoost is available in several programming languages, including C++, Java, and Python, enhancing its accessibility and usability across various platforms.

6.11.2 Algorithm of XGBoost

Next, we give an intuitive explanation of the XGBoost algorithm:

1. **Initialization**: XGBoost starts with initializing the ensemble model with a simple estimate. This initial estimate is often the mean (for regression) or the log-odds (for classification) of the target values.

2. **Boosting iterations**: The algorithm proceeds in a series of boosting iterations, where each iteration improves the model by adding a new decision tree (weak learner) to the ensemble.

3. **Calculation of residuals**: For each boosting iteration, the algorithm calculates the difference between the actual target values and the current ensemble's predictions. These differences are called residuals.

4. **Building decision trees**: A new decision tree is added to the ensemble in each boosting iteration. This tree is trained to predict the residuals calculated in the previous step. The goal of the new tree is to capture the patterns that were not well-modeled by the existing ensemble.

5. **Regularization**: To prevent overfitting and improve generalization, XGBoost applies regularization techniques to each new tree. These techniques include max depth constraints on the trees, minimum child weight constraints, and column subsampling (selecting a subset of features for each tree).

6. **Calculating leaf outputs**: Once a decision tree is trained, its outputs (predictions for individual instances) are assigned to the leaves of the tree. These outputs are determined based on the distribution of residuals that fall into each leaf during training.

7. **Updating ensemble predictions**: The predictions of the new tree are then added to the predictions made by the existing ensemble, updating the overall prediction of the model.

8. **Learning rate**: Each new tree's contribution to the ensemble's prediction is controlled by a parameter called the learning rate. A lower learning rate makes the model more robust by shrinking the contribution of each new tree, reducing the risk of overfitting.

9. **Repeat**: Steps 3-8 are repeated for a predefined number of boosting iterations or until a stopping criterion is met.

10. **Final prediction**: The final prediction of the XGBoost model is the sum of predictions from all the individual trees in the ensemble, adjusted by the learning rate.

In Algorithm 6.2, a generic unregularized XGBoost algorithm is shown.

6.11.3 Hyperparamaters of XGBoost

When writing code regarding an XGBoost model, the following are some of the most common **hyperparameters** that someone has to tune:

- **n_estimators**: This hyperparameter specifies the number of boosting rounds or decision trees to build. It controls the number of weak learners in the ensemble. Typically, higher values may lead to better performance but can also increase training time.

- **learning_rate (or eta)**: The learning rate determines the step size at each iteration while moving toward a minimum of a loss function. Lower values make the optimization process more robust but require more iterations to converge.
- **max_depth**: It controls the maximum depth of each decision tree in the ensemble. Increasing this value can lead to more complex trees, but it may also increase the risk of overfitting.
- **min_child_weight**: This hyperparameter sets the minimum sum of instance weight (hessian) needed in a child. It can be used to control overfitting. Larger values make the algorithm more conservative.
- **subsample**: It determines the fraction of randomly sampled training data to use for growing trees during each boosting round. Setting it to a value less than 1.0 can help prevent overfitting.
- **colsample_bytree**: This controls the fraction of features to be randomly sampled when building each tree. It can be used to introduce randomness and reduce overfitting.
- **gamma (or min_split_loss)**: It specifies a regularization term that penalizes larger tree structures. Increasing gamma can make the algorithm more conservative and prevent overfitting.
- **lambda (reg_lambda)**: This is the L2 regularization term on weights. It adds a penalty term to the loss function based on the magnitude of the weights, discouraging large weights.
- **alpha (reg_alpha)**: This is the L1 regularization term on weights. Like lambda, it adds a penalty term to the loss function, encouraging sparsity in the feature weights.
- **objective**: It defines the learning task and corresponding objective function. Common values include "reg:squarederror" for regression tasks and "binary:logistic" for binary classification.
- **eval_metric**: This specifies the evaluation metric to be used during training. Common options include "rmse" for regression and "logloss" for classification.
- **early_stopping_rounds**: It allows you to stop training if the model's performance on a validation set does not improve for a specified number of consecutive rounds.
- **scale_pos_weight**: Used in imbalanced classification tasks, it controls the balance of positive and negative weights. It can help the model handle class imbalance.

6.12 Light Gradient Boosting Machine (LightGBM)

In this subsection, the analysis has been extracted from [7].

Algorithm 6.2: A generic unregularized XGBoost algorithm [6]

```
Input: Training set: $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, Loss function: $L(y, F(\mathbf{x}))$, Number of weak
       learners: $M$, and Learning rate: $\alpha$
Output: Final boosted model $\hat{F}_M(\mathbf{x})$
1 Initialize the model with a constant value:
      $$\hat{F}_{(0)}(\mathbf{x}) = \arg\min_{\theta} \sum_{i=1}^{N} L(y_i, \theta)$$
2 for $m = 1$ to $M$ do
3     Compute the "gradients" and "hessians":
          $$\hat{g}_m(\mathbf{x}_i) = \left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(x)=\hat{F}_{(m-1)}(\mathbf{x})}$$
          $$\hat{h}_m(\mathbf{x}_i) = \left[\frac{\partial^2 L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)^2}\right]_{f(x)=\hat{F}_{(m-1)}(\mathbf{x})}$$
4     Fit a base learner (or weak learner, e.g. tree) using the training set
      $\left\{\mathbf{x}_i, -\frac{\hat{g}_m(\mathbf{x}_i)}{\hat{h}_m(\mathbf{x}_i)}\right\}_{i=1}^{N}$ by solving the following optimization problem:
          $$\hat{\phi}_m = \arg\min_{\phi \in \Phi} \sum_{i=1}^{N} \frac{1}{2}\hat{h}_m(\mathbf{x}_i)\left[\phi(\mathbf{x}_i) - \frac{\hat{g}_m(\mathbf{x}_i)}{\hat{h}_m(\mathbf{x}_i)}\right]^2$$
5     Update the model:
          $$\hat{F}_m(\mathbf{x}) = \alpha\hat{\phi}_m(\mathbf{x})$$
6     Update the model:
          $$\hat{F}_{(m)}(\mathbf{x}) = \hat{F}_{(m-1)}(\mathbf{x}) + \hat{F}_m(\mathbf{x})$$
7 Output the final model: $\hat{F}(\mathbf{x}) = \hat{F}_{(M)}(\mathbf{x}) = \sum_{m=0}^{M} \hat{F}_m(\mathbf{x})$;
```

Gradient boosting decision tree (GBDT) is a widely-used machine learning algorithm, due to its efficiency, accuracy, and interpretability. GBDT achieves state-of-the-art performances in many machine learning tasks, such as multi-class classification, click prediction, and learning to rank. In recent years, with the emergence of big data (in terms of both the number of features and the number of instances), GBDT is facing new challenges, especially in the trade-off between accuracy and efficiency. Conventional implementations of GBDT need to, for every feature, scan all the data instances to estimate the information gain of all the possible split points. Therefore, their computational complexities will be proportional to both the number of features and the number of instances. This makes these implementations very time-consuming when handling big data.

> To tackle this challenge, a straightforward idea is to reduce the number of data instances and the number of features. However, this turns out to be highly non-trivial. For example, it is unclear how to perform data sampling for GBDT. While there are some works that sample data according to their weights to speed up the training process of boosting, they cannot be directly applied to GBDT.

As an answer to these challenges, the algorithm of **LightGBM** [7] was introduced, which is a gradient-boosting framework based on decision trees to increase the efficiency of the model and reduces memory usage. It uses two novel techniques:

- **Gradient-based One Side Sampling (GOSS)**
- **Exclusive Feature Bundling (EFB)**

These techniques fulfill the limitations of the histogram-based algorithm that is primarily used in all GBDT frameworks. The two techniques of GOSS and EFB described below form the characteristics of the LightGBM algorithm. They comprise together to make the model work efficiently and provide it a cutting edge over other GBDT frameworks. It is worth noting that in LightGBM, the **leaf-wise** tree growth strategy is being used.

6.12.1 Gradient-based One-Side Sampling (GOSS)

The following analysis has been extracted from [7].

> While there is no native weight for data instances in GBDT, different data instances have varied roles in the computation of information gain. The instances with larger gradients (i.e., under-trained instances) will contribute more to the information gain. GOSS keeps those instances with large gradients (e.g., larger than a predefined threshold, or among the top percentiles), and only randomly drops those instances with small gradients to retain the accuracy of information gain estimation. This treatment can lead to a more accurate gain estimation than uniformly random sampling, with the same target sampling rate, especially when the value of information gain has a large range.
>
> In particular, in order to compensate for the influence to the data distribution, when computing the information gain, GOSS introduces a constant multiplier for the data instances with small gradients (see Algorithm 6.3). Specifically, GOSS firstly sorts the data instances according to the absolute value of their gradients and selects the top $a \times 100\%$ instances. Then it randomly samples $b \times 100\%$ instances from the rest of the data. After that, GOSS amplifies the sampled data with small gradients by a constant $\frac{1-a}{b}$ when calculating the information gain. By doing so, we put more focus on the under-trained instances without changing the original data distribution by much.

Algorithm 6.3: Gradient-based One-Side Sampling (GOSS) [7]

```
Input: Training data: $I$, Number of iterations: $d$, Sampling ratio of large
       gradient data: $a$, Sampling ratio of small gradient data: $b$, Loss function:
       $loss$, and Weak learner: $L$
1  models $\leftarrow \{\}$
2  fact $\leftarrow \frac{1-a}{b}$
3  topN $\leftarrow a \times \text{len}(I)$
4  randN $\leftarrow b \times \text{len}(I)$
5  for $i = 1$ to $d$ do
6      preds $\leftarrow$ models.predict($I$)
7      g $\leftarrow loss(I, \text{preds})$
8      w $\leftarrow \{1, 1, ...\}$
9      sorted $\leftarrow$ GetSortedIndices(abs(g))
10     topSet $\leftarrow$ sorted[1 : topN]
11     randSet $\leftarrow$ RandomPick(sorted[topN : len($I$)], randN)
12     usedSet $\leftarrow$ topSet + randSet
13     w[randSet]$\times =$ fact {Comment: Assign weight fact to the small gradient data}
14     newModel $\leftarrow L(I[\text{usedSet}], -\text{g}[\text{usedSet}], \text{w}[\text{usedSet}])$
15     models.append(newModel)
```

6.12.2 Exclusive Feature Bundling (EFB)

The following analysis has been extracted from [7].

> High-dimensional data are usually very sparse which provides us the possibility of designing a nearly lossless approach to reduce the number of features. Specifically, in a sparse feature space, many features are mutually exclusive, i.e., they never take non-zero values simultaneously. The exclusive features can be safely bundled into a single feature, called an Exclusive Feature Bundle. So, an efficient algorithm was designed by reducing the optimal bundling problem to a graph coloring problem (by taking features as vertices and adding edges for every two features if they are not mutually exclusive), and solving it by a greedy algorithm with a constant approximation ratio. Hence, the complexity of histogram building changes from $\mathcal{O}(\#\text{data} \times \#\text{feature})$ to $\mathcal{O}(\#\text{data} \times \#\text{bundle})$, while $\#\text{bundle} \ll \#\text{feature}$. Hence, the speed of the training framework is improved without hurting accuracy.
>
> Addressing the initial concern, it has been established that determining the optimal bundling strategy is NP-Hard, indicating the inherent challenge of finding an exact solution within polynomial time. Seeking a viable alternative, researchers have approached the problem by converting the optimal bundling issue into a graph coloring problem, with features represented as vertices and edges added between non-mutually exclusive pairs. Following this reduction, a pragmatic approach employs a greedy algorithm that offers reasonable results with a constant approximation ratio for graph coloring, ultimately leading to the creation of the feature bundles.

Moreover, it is worth noting that in many scenarios, a substantial number of features display a lack of complete mutual exclusivity. These features seldom assume zero values simultaneously. In light of this, a potential enhancement lies in permitting a slight fraction of conflicts within the algorithm. This adjustment could yield an even smaller collection of feature bundles, thereby further refining the computational efficiency of the process. Quantitative analysis demonstrates that introducing a controlled level of conflict-achieved through a calculated introduction of randomness—would impact training accuracy by a maximum of $\mathcal{O}([(1-\gamma)n]^{2/3})$. Here, γ represents the maximal conflict rate within each bundle. By opting for a relatively modest γ, a harmonious balance between accuracy and efficiency can be achieved.

Drawing on the previously discussed ideas, an algorithm is formulated for exclusive feature bundling, as outlined in Algorithm 6.4. Initially, a graph is constructed with weighted edges, where the weights correspond to the total conflicts between features. Subsequently, features are sorted by their degrees within the graph, in a descending order. The features in the ordered list are then evaluated; they are either assigned to an existing bundle with minimal conflict (controlled by γ), or a new bundle is created. The time complexity of Algorithm 6.4 is $\mathcal{O}(\#\text{feature}^2)$, and it is executed only once before the training phase. While this complexity is manageable for scenarios with a moderate number of features, it may encounter challenges when dealing with an extensive feature set, such as millions of features. To further enhance computational efficiency, an alternative strategy is proposed, which bypasses the graph construction step. This strategy entails ordering features based on the count of nonzero values, mirroring the concept of ordering by degrees, as more nonzero values often correlate with a higher likelihood of conflicts. As this approach primarily modifies ordering strategies in Algorithm 6.4, the specifics of the new strategy are omitted to prevent redundancy.

The second aspect revolves around the necessity of a proficient technique for merging features within the same bundle, aimed at reducing the associated training complexity. Central to this process is ensuring that the values of original features remain discernible within the feature bundles. Given that the histogram-based algorithm operates on discrete bins instead of continuous feature values, the construction of a feature bundle involves situating exclusive features in separate bins. This is achieved by introducing offsets to the original feature values. To illustrate, consider a scenario where two features reside in a feature bundle. Initially, feature A spans values within $[0, 10)$, while feature B spans values within $[0, 20)$. By adding an offset of 10 to the values of feature B, the refined feature takes on values from $[10, 30)$. Consequently, the merger of features A and B becomes feasible, utilizing a feature bundle spanning $[0, 30]$ to replace the original features. Further details of this process are presented in Algorithm 6.5.

The EFB algorithm effectively bundles numerous exclusive features into a more compact set of dense features, thereby mitigating unnecessary computational overhead for zero-valued features. It is important to note that an optimization approach can also be applied to the fundamental histogram-based algorithm. This optimization involves excluding zero feature values by employing a table to record data with non-zero values for each feature. By scanning this table, the cost associated with histogram building for a feature transitions from $\mathcal{O}$(#data) to $\mathcal{O}$(#non_zero_data). It's worth mentioning that this optimization necessitates additional memory allocation and computational resources to manage per-feature tables throughout the entirety of the tree growth process. This enhancement has been incorporated into LightGBM as a basic function. Importantly, this optimization coexists harmoniously with EFB and can be employed when the bundles exhibit sparsity.

Algorithm 6.4: Greedy Bundling [7]

```
Input: Features: $F$, and Max conflict count: $K$
Output: $bundles$
1  Construct graph $G$
2  $searchOrder \leftarrow G$.sortByDegree()
3  $bundles \leftarrow \{\}$
4  $bundlesConflict \leftarrow \{\}$
5  for $i$ in $searchOrder$ do
6      $needNew \leftarrow$ True;
7      for $j = 1$ to $len(bundles)$ do
8          $cnt \leftarrow$ ConflictCnt($bundles[j], F[i]$)
9          if $cnt \leftarrow bundlesConflict[i] \leq K$ then
10             $bundles[j]$.add($F[i]$)
11             $needNew \leftarrow$ False
12             break
13     if $needNew$ then
14         Add $F[i]$ as a new bundle to $bundles$
```

Algorithm 6.5: Merge Exclusive Features [7]

```
Input: Number of data: $numData$, and One bundle of exclusive features: $F$
Output: $newBin$, and $binRanges$
1  $binRanges \leftarrow \{0\}$
2  $totalBin \leftarrow 0$
3  for $f$ in $F$ do
4      $totalBin+ = f.numBin$
5      $binRanges$.append($totalBin$)
6  $newBin \leftarrow$ new Bin($numData$)
7  for $i = 1$to $numData$ do
8      $newBin[i] \leftarrow 0$
9      for $j = 1$ to $len(F)$ do
10         if $F[j].bin[i] \neq 0$ then
11             $newBin[i] \leftarrow F[j].bin[i] + binRanges[j]$
```

6.12.3 Hyperparamaters of LightGBM

When writing code regarding a LightGBM model, the following are some of the most common **hyperparameters** that someone has to tune:

- **num_leaves**: This hyperparameter controls the maximum number of leaves (terminal nodes) in each tree. Increasing num_leaves can make the model more expressive, but may lead to overfitting.
- **learning_rate (or eta)**: This hyperparameter determines the step size at each iteration during the optimization process. Lower values make the optimization more robust but require more iterations.
- **max_depth**: Specifies the maximum depth of the individual decision trees in the ensemble. It is an alternative to controlling tree depth compared to the num_leaves parameter.
- **min_data_in_leaf**: This sets the minimum number of data points that should be present in a leaf node. It helps control overfitting and is another way to control tree complexity.
- **bagging_fraction (or subsample)**: It controls the fraction of data randomly sampled for training each tree. Values less than 1.0 introduce randomness and can prevent overfitting.
- **feature_fraction (or colsample_bytree)**: Similar to XGBoost, this parameter controls the fraction of features to be randomly selected for each tree, adding further randomness.
- **lambda_l2 (or reg_lambda)**: LightGBM supports L2 regularization on the leaf weights to prevent overfitting. This parameter controls the strength of L2 regularization.
- **lambda_l1 (or reg_alpha)**: This parameter controls the strength of L1 regularization on the leaf weights.
- **min_child_samples**: Specifies the minimum number of data points required in a leaf. It can be used as an alternative to min_data_in_leaf to control overfitting.
- **boosting_type**: LightGBM supports different boosting types, including "gbdt" (Gradient Boosting Decision Tree), "dart" (Dropouts meet Multiple Additive Regression Trees), and "goss" (Gradient-based One-Side Sampling).
- **objective**: It defines the learning task and corresponding objective function, such as "regression", "binary", or "multiclass".
- **metric**: Specifies the evaluation metric to be used during training and testing. Common options include "rmse" for regression and "binary_logloss" for binary classification.
- **early_stopping_rounds**: Like XGBoost, this parameter allows you to stop training if the model's performance on a validation set does not improve for a specified number of consecutive rounds.

- **scale_pos_weight**: Used in imbalanced classification tasks, it balances the positive and negative weights to handle class imbalance.
- **max_bin**: Controls the maximum number of bins to bucket feature values into during histogram-based tree building. Increasing this may lead to more accurate trees, but can also increase training time.

7 Deep Learning Methods

The content of subsections 7.3, 7.4, 7.5, 7.6, and 7.7 is obtained from [66], which has used a very concise and consistent notation describing basic concepts of neural networks and details on recurrent neural networks.

7.1 The concept of deep learning

Deep learning is a branch of machine learning which is based on artificial neural networks (ANNs), or deep neural networks (DNNs). It is capable of learning complex patterns and relationships within data. It has become increasingly popular in recent years due to the advances in processing power and the availability of large datasets. Some key features are the following [67]:

1. Deep learning is a subfield of machine learning that involves the use of neural networks to model and solve complex problems. Neural networks are modeled after the structure and function of the human brain and consist of layers of interconnected nodes that process and transform data.
2. The key characteristic of deep learning is the use of deep neural networks, which have multiple layers of interconnected nodes. These networks can learn complex representations of data by discovering hierarchical patterns and features in the data. Deep learning algorithms can automatically learn and improve from data without the need for manual feature engineering, and they have achieved significant success in various fields, including image recognition, natural language processing, speech recognition, and recommendation systems.
3. Deep learning architectures include Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Deep Belief Networks (DBNs).
4. Training DNNs typically requires a large amount of data and computational resources. However, the availability of cloud computing and the development of specialized hardware, such as Graphics Processing Units (GPUs), have made it easier to train DNNs.

7.2 The components of a deep learning network

In the view of a high level representation, a DNN contains the following components [68]:

1. **Input layer**:
   An ANN has several nodes that input data into it. These nodes make up the input layer of the system, and, in general, they represent the input "features".
2. **Hidden layer**:
   The input layer processes and passes the data to layers further in the neural network.

These hidden layers process information at different levels, adapting their behavior as they receive new information. Deep learning networks have hundreds of hidden layers that they can use to analyze a problem from several different angles. For example, if we were given an image of an unknown animal that we had to classify, we would compare it with animals you already know. For example, we would look at the shape of its eyes and ears, its size, the number of legs, and its fur pattern, and we would try to identify patterns. The hidden layers in DNNs work in the same way. If a deep learning algorithm is trying to classify an animal image, each of its hidden layers processes a different feature of the animal and tries to accurately categorize it.

3. **Output layer**:
   The output layer consists of the nodes that output the data. Deep learning models that output "yes" or "no" answers have only two nodes in the output layer. On the other hand, those that output a wider range of answers have more nodes.

In Figure 7.1, a high level illustration of a simple deep neural network is shown.

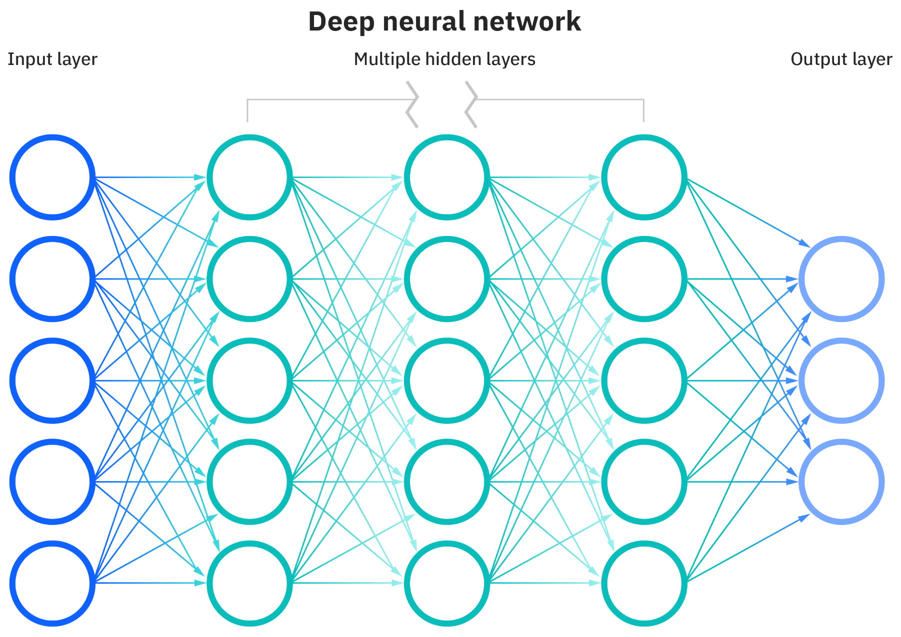


Figure 7.1: High level illustration of a simple deep neural network.

7.3 The Perceptron

The most basic type of artificial neuron is called a **perceptron**. Perceptrons consist of a number of external input links, a threshold, and a single external output link. Additionally, perceptrons have an internal input, b, called bias. The perceptron takes a vector of real-valued input values, all of which are weighted by a multiplier. In a previous perceptron training phase, the perceptron learns these weights on the basis of training data. It sums all weighted input values and "fires" if the resultant value is above a predefined threshold. The output of the perceptron is always boolean, and it is considered to have fired if the output is '1'. The deactivated value of the perceptron is '-1', and the threshold value is, in most cases, '0'.

As we only have one unit for the perceptron, we omit the sub-indexes that refer to the unit. Given the input vector $\mathbf{x} = (x_1, \ldots, x_n)^T$ and trained weights $\mathbf{w} = (w_1, \ldots, w_n)^T$, the perceptron outputs y, which is computed by the formula

$$y = \begin{cases} 1 & , \quad \text{if} \quad \sum_{i=1}^{n} w_i x_i + b > 0 \\ -1 & , \quad \text{otherwise} \end{cases} \tag{7.1}$$

We refer to $z = \sum_{i=1}^{n} w_i x_i$ as the weighted input, and to $s = z + b$ as the state of the perceptron. For the perceptron to fire, its state must exceed the value of the threshold. The general structure of a perceptron is shown in Figure 7.2.

Single perceptron units can already represent a number of useful functions. Examples are the boolean functions AND, OR, NAND, and NOR. Other functions are only representable using networks of neurons, as they are not linearly separable, e.g. boolean function XOR. Single perceptrons are limited to learning only functions that are **linearly separable**. In general, a problem is linear and the classes are linearly separable in a n-dimensional space if the decision surface is a $(n-1)$-dimensional hyperplane.

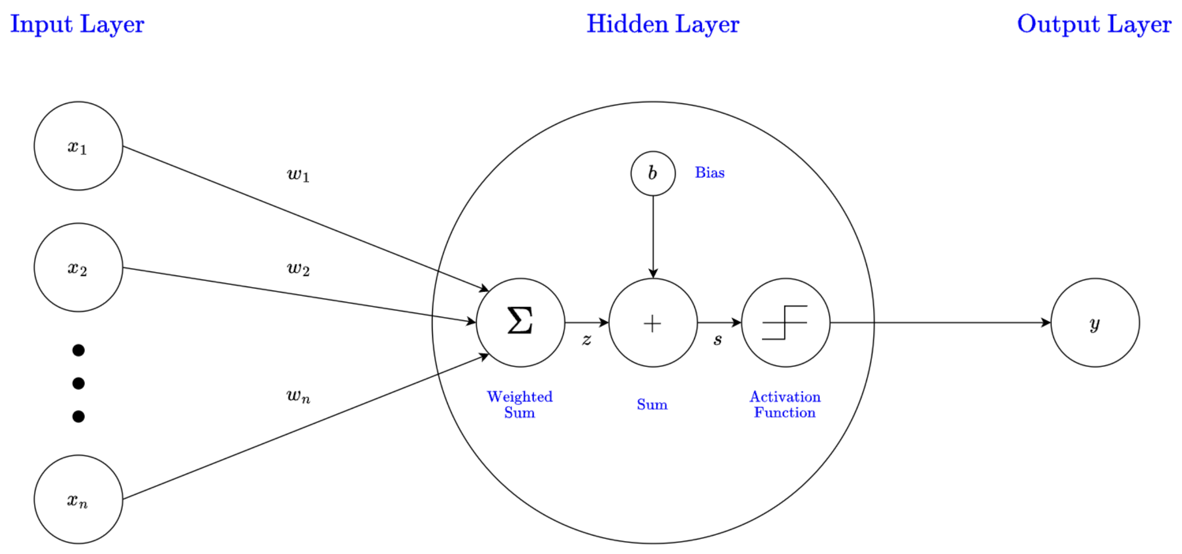


Figure 7.2: The general structure of the most basic type of artificial neuron, called a perceptron.

7.4 The Delta Learning Rule

Perceptron training is learning by imitation, which is called "supervised learning". During the training phase, the perceptron produces an output and compares it with the true output value provided by the training data. In cases of misclassification, it then modifies the weights accordingly [69]. In a finite time, the perceptron will converge to reproduce the correct behavior, provided that the training examples are linearly separable. Convergence is not assured if the training data is not linearly separable. A variety of training algorithms for perceptrons exist, of which the most common are the **perceptron learning rule** and the **delta learning rule**. Both start with random weights, and both guarantee convergence to an acceptable hypothesis.

Using the **perceptron learning rule algorithm**, the perceptron can learn from a set of samples. A sample is a pair $(\mathbf{x}, d)$ where $\mathbf{x}$ is the input and d is its label. For the sample $(\mathbf{x}, d)$, given the input $\mathbf{x} = (x_1, \ldots, x_n)^T$, the old weight vector $\mathbf{w} = (w_1, \ldots, w_n)^T$ is updated to the new vector $\mathbf{w}'$ using the rule

$$w_i' = w_i + \Delta w_i \tag{7.2}$$

with

$$\Delta w_i = \eta(d - y)x_i \tag{7.3}$$

where y is the output calculated using the input $\mathbf{x}$ and the weights $\mathbf{w}$ and η is the learning rate. The **learning rate** is a constant that controls the degree to which the weights are changed. As stated before, the initial weight vector $\mathbf{w}^0$ has random values. The algorithm will only converge towards an optimum if the training data is linearly separable, and the learning rate is sufficiently small. The perceptron rule fails if the training examples are not linearly separable.

The **delta learning rule** was specifically designed to handle linearly separable and linearly non-separable training examples. It also calculates the errors between calculated output and output data from training samples, and modifies the weights accordingly. The modification of weights is achieved by using the **gradient descent** optimization algorithm, which alters them in the direction that produces the steepest descent along the error surface towards the global minimum error. The delta learning rule is the basis of the error backpropagation algorithm, which we will discuss later in this section.

7.5 The Sigmoid Threshold Unit

The sigmoid threshold unit is a different kind of artificial neuron, very similar to the perceptron, but uses a sigmoid function to calculate the output. The output y is computed by the formula

$$y = \frac{1}{1 + e^{-l \cdot s}} \tag{7.4}$$

with

$$s = z + b, \quad \text{where} \quad z = \sum_{i=1}^{n} w_i x_i \tag{7.5}$$

where b is the bias and l is a positive constant that determines the steepness (slope) of the sigmoid function. The major effect on the perceptron is that the output of the sigmoid threshold unit now has more than two possible values. Now, the output is “squashed” by a continuous function that ranges between 0 and 1.

Accordingly, the sigmoid function is called the “squashing” function, because it maps a very large input domain onto a small range of outputs. For a low total input value, the output of the sigmoid function is close to zero, whereas it is close to one for a high total input value. The advantage of neural networks using sigmoid units is that they are capable of representing non-linear functions. Cascaded linear units, like the perceptron, are limited to representing linear functions. A sigmoid threshold unit is sketched in Figure 7.3.

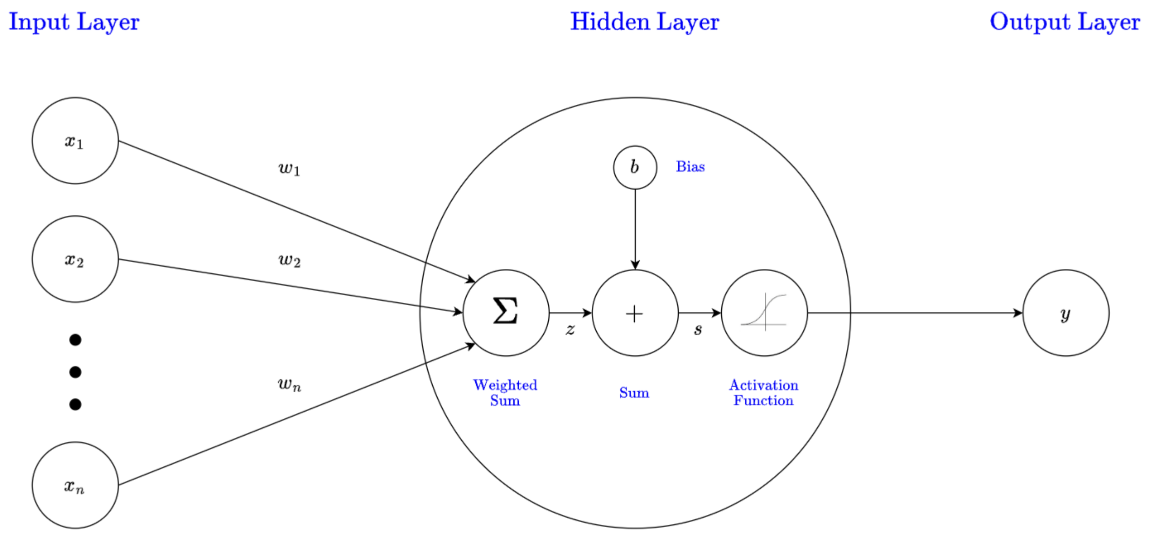


Figure 7.3: The general structure of a sigmoid threshold unit.

7.6 Feed-Forward Neural Networks and Backpropagation

In **feed-forward neural networks (FFNNs)**, sets of neurons are organized in layers, where each neuron computes a weighted sum of its inputs. **Input neurons** take signals from the environment, and **output neurons** present signals to the environment. Neurons that are not directly connected to the environment, but which are connected to other neurons, are called **hidden neurons**. FFNNs are **loop-free** and **fully connected**. This means that each neuron provides an input to each neuron in the following layer, and that none of the weights give an input to a neuron in a previous layer.

The simplest type of FFNNs are **single-layer perceptron networks**. Single-layer neural networks consist of a set of input neurons, defined as the input layer, and a set of output neurons, defined as the output layer. The outputs of the input-layer neurons are directly connected to the neurons of the output layer. The weights are applied to the connections between the input and output layer. In the single-layer perceptron network, every single perceptron calculates the sum of the products of the weights and the inputs. The perceptron fires ‘1’ if the value is above the threshold value. Otherwise, the perceptron takes the deactivated value, which is usually ‘-1’. The threshold value is typically ‘0’.

Sets of neurons organized in several layers can form multilayer forward-connected networks. The input and output layers are connected via at least one hidden layer, built from set(s) of hidden neurons. The multilayer FFNN sketched in Figure 7.1, with one input layer, three hidden layers and one output layer, is classified as a 4-layer FFNN. For most problems, FFNNs with large number of hidden layers offer no advantage.

Multilayer FFNNs using sigmoid threshold functions are able to express **non-linear decision surfaces**. Any function can be closely approximated by these networks, given enough hidden units.

The most common neural network learning technique is the **error backpropagation** algorithm. It uses **gradient descent** to learn the weights in multilayer networks. It works in small iterative steps, starting backwards from the output layer towards the input layer. A requirement is that the activation function of the neuron be differentiable. Usually, the weights of a feed-forward neural network are initialized to small, normalized random numbers using bias values. Then, error backpropagation applies all training samples to the neural network and computes the input and output of each unit for all (hidden and) output layers.

The set of units of the network is $N \triangleq I \sqcup H \sqcup O$, where $\sqcup$ is disjoint union, and I, H, O are the sets of input, hidden, and output units, respectively. We denote input units by i, hidden units by h, and output units by o. For convenience, we define the set of non-input units as $U \triangleq H \sqcup O$. For a non-input unit $u \in U$, the input to u is denoted by $\mathbf{x}_u$, its state by s_u, its bias by b_u, and its output by y_u. Given units $u, v \in U$, the weight that connects u with v is denoted by w_{uv}.

To model the external input that the neural network receives, we use the external input vector $\mathbf{x} = (x_1, \ldots, x_n)^T$. For each component of the external input vector, we find a corresponding input unit that models it, so the output of the i-th input unit should be the equivalent i-th component of the input to the network (i.e., x_i), and consequently, $|I| = n$.

For the non-input unit $u \in U$, the output of u, written y_u, is defined using the sigmoid activation function by

$$y_u = \frac{1}{1 + e^{-s_u}} \tag{7.6}$$

where s_u is the state of u, and it is defined by

$$s_u = z_u + b_u \tag{7.7}$$

where b_u is the bias of u, and z_u is the weighted input of u, defined in turn by

$$\begin{aligned} z_u &= \sum_v w_{vu} x_{v,u}, \quad \text{with } v \in \text{Pre}(u) \\ &= \sum_v w_{vu} y_v \end{aligned} \tag{7.8}$$

where x_{vu} is the information that v passes as input to u, and Pre(u) is the set of units v that precede u; that is, input units, and hidden units that feed their outputs y_u (see Equation (7.6)) multiplied by the corresponding weight w_{vu} to the unit u.

Starting from the input layer, the inputs are propagated forward through the network until the output units are reached at the output layer. Then, the output units produce an observable output (the network output) y. More precisely, for $o \in O$, its output y_o corresponds to the o-th component of $\mathbf{y}$.

Next, the backpropagation learning algorithm propagates the error backwards, and the weights and biases are updated such that we reduce the error with respect to the present training sample. Starting from the output layer, the algorithm compares the network output y_o with the corresponding desired target output d_o. It calculates the error e_o for each output neuron using some error function to be minimized. The error e_o is computed as

$$e_o = (d_o - y_o) \tag{7.9}$$

and we have the following notion of overall error of the network

$$E = \frac{1}{2}\sum_{o \in O} e_o^2 \tag{7.10}$$

To update the weight w_{uv}, we will use the formula

$$\Delta w_{uv} = -\eta \frac{\partial E}{\partial w_{uv}} \tag{7.11}$$

where η is the learning rate. We now make use of the **chain rule** to calculate the weight update by deriving the error with respect to the activation, and the activation in terms of the state, and in turn the derivative of the state with respect to the weight:

$$\Delta w_{uv} = -\eta \frac{\partial E}{\partial y_u}\frac{\partial y_u}{\partial s_u}\frac{\partial s_u}{\partial w_{uv}} \tag{7.12}$$

The derivative of the error with respect to the activation for output units is

$$\frac{\partial E}{\partial y_o} = -(d_o - y_o) \tag{7.13}$$

now, the derivative of the activation with respect to the state for output units is

$$\frac{\partial y_o}{\partial s_o} = y_o(1 - y_o) \tag{7.14}$$

and the derivative of the state with respect to a weight that connects the hidden unit h to the output unit o is

$$\frac{\partial s_u}{\partial w_{uv}} = y_h \tag{7.15}$$

Let's define, for the output unit o, the error signal by

$$\vartheta_o = -\frac{\partial E}{\partial y_o}\frac{\partial y_o}{\partial s_o} \tag{7.16}$$

for output units we have that

$$\vartheta_o = (d_o - y_o)y_o(1 - y_o) \tag{7.17}$$

and we see that we can update the weight between the hidden unit h and the output unit o by

$$\Delta w_{ho} = \eta \vartheta_o y_h \tag{7.18}$$

Now, for a hidden unit h, if we consider that its notion of error is related to how much it contributed to the production of a faulty output, then we can backpropagate the error from the output units that h sends signals to; more precisely, for an input unit i, we need to expand the equation

$$\Delta w_{ih} = -\eta \frac{\partial E}{\partial w_{ih}} \tag{7.19}$$

to

$$\Delta w_{ih} = -\eta \sum_o \frac{\partial E}{\partial y_o} \frac{\partial y_o}{\partial s_o} \frac{\partial s_o}{\partial y_h} \frac{\partial y_h}{\partial s_h} \frac{\partial s_h}{\partial w_{ih}} \quad , \quad o \in \mathrm{Suc}(h) \tag{7.20}$$

where Suc(h) is the set of units that succeed h, that is, the units that are fed with the output of h as part of their input. By solving the partial derivatives, we obtain

$$\begin{aligned} \Delta w_{ih} &= -\eta \sum_o (\vartheta_o w_{ho}) \frac{\partial y_h}{\partial s_h} \frac{\partial s_h}{\partial w_{ih}} \\ &= \eta \sum_o (\vartheta_o w_{ho}) y_h (1 - y_h) y_i \end{aligned} \tag{7.21}$$

If we define the error signal of the hidden unit h by

$$\vartheta_h = \sum_o (\vartheta_o w_{ho}) y_h (1 - y_h) \quad , \quad o \in \mathrm{Suc}(h) \tag{7.22}$$

then we have a uniform expression for weight change, that is,

$$\Delta w_{vu} = -\eta \vartheta_u y_v \tag{7.23}$$

We calculate Δw_{vu} again and again until all network outputs are within an acceptable range, or some other terminating condition is reached.

7.7 Recurrent Neural Networks

Recurrent Neural Networks (RNN) differ from feed-forward neural networks by the presence of feedback connections, where the flow of information occurs between neurons of the same layer or from higher layer neurons to lower layer neurons [70, 71]. The presence of feedback connections makes RNNs able to perform tasks that require memory. This is because the network keeps information about its previous status. More specifically, the network at the time t transmits to itself the information to be used at the moment $t + 1$ (together with the external input received at $t + 1$). Therefore, the behavior of the network is influenced by the input it receives at a given instant, and by what happened to the network at the previous instant (in turn influenced by the previous instants).

7.7.1 Basic architecture

Fully Recurrent Neural Networks (FRNN) connect the outputs of all neurons to the inputs of all neurons. This is the most general neural network topology because all other topologies can be represented by setting some connection weights to zero to simulate the lack of connections between those neurons. Figure 7.4 depicts a FRNN.

The **Elman network** [72] is similar to a 3-layer neural network, but additionally, the outputs of the hidden layer are saved in so-called “context cells”. The output of a context cell is circularly fed back to the hidden neuron along with the originating signal. Every hidden neuron has its own context cell and receives input both from the input layer and the context cells. Elman networks can be trained with standard error backpropagation, the output from the context cells being simply regarded as an additional input. Figure 7.5 depicts an Elman RNN. **Jordan networks** [73] have a similar structure to Elman networks, but the context cells are instead fed by the output layer. Figure 7.6 depicts a Jordan RNN.

RNNs need to be trained differently from FFNNs. This is because, for RNNs, we need to propagate information through the recurrent connections in-between steps. The most common and well-documented learning algorithms for training RNNs in temporal, supervised learning tasks are **backpropagation through time (BPTT)** and **real-time recurrent learning (RTRL)**. In BPTT, the network is **unfolded in time** to construct a FFNN. Then, the generalized delta rule is applied to update the weights. This is an **offline** learning algorithm in the sense that we first collect the data and then build the model from the system. In RTRL, the gradient information is propagated forward. Here, the data is collected online from the system, and the model is learned during collection. Therefore, RTRL is an **online** learning algorithm.

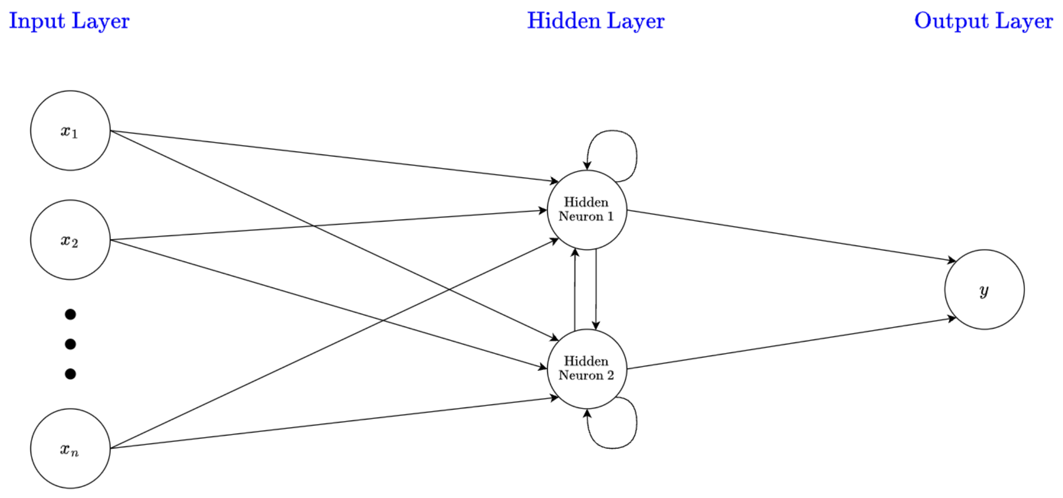


Figure 7.4: The general structure of a FRNN.

7.7.2 Training Recurrent Neural Networks

The most common methods to train recurrent neural networks are Backpropagation Through Time (BPTT) [74, 70, 71] and Real-Time Recurrent Learning (RTRL) [71, 75], whereas BPTT is the most common method. The main difference between BPTT and RTRL is the way the weight changes are calculated. The Long Short-Term Memory (LSTM) recurrent neural networks (in which we will focus next) uses the BPTT method.

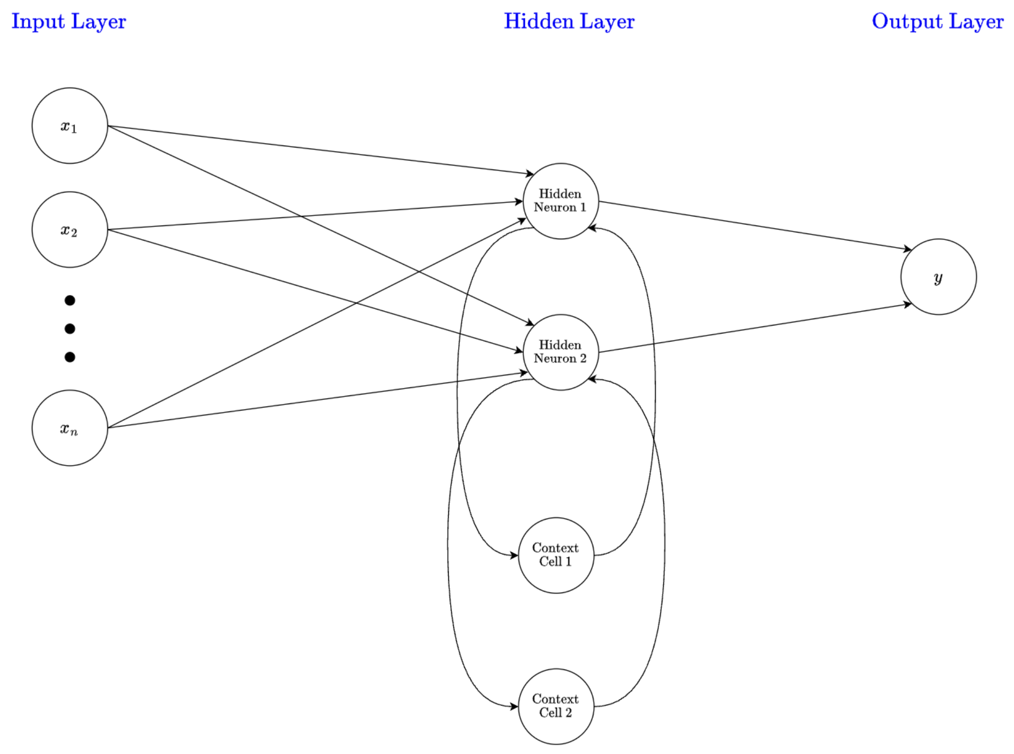


Figure 7.5: The general structure of an Elman RNN.

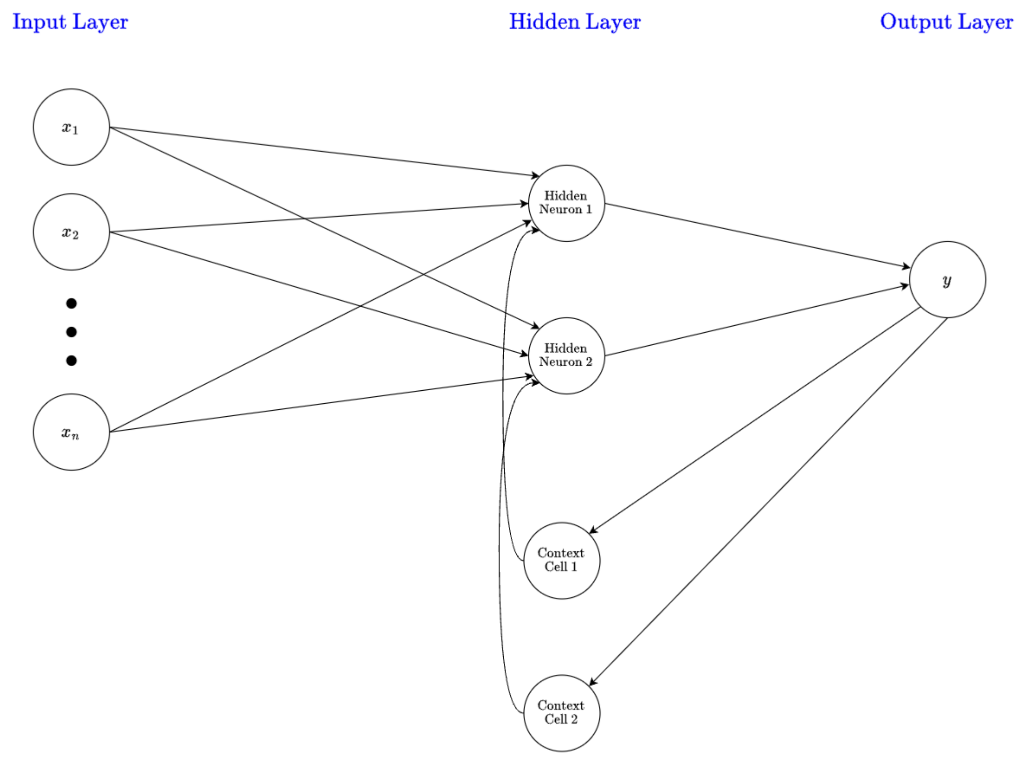


Figure 7.6: The general structure of a Jordan RNN.

7.7.3 Backpropagation Through Time

The BPTT algorithm makes use of the fact that, for a finite period of time, there is an FFNN with identical behavior for every RNN. To obtain this FFNN, we need to unfold the RNN in time. Figure 7.7 shows a simple, fully recurrent neural network with a single, two-neuron layer. The corresponding feed-forward neural network, shown in Figure 7.7, requires a separate layer for each time step with the same weights for all layers. If weights are identical to the RNN, both networks show the same behavior.

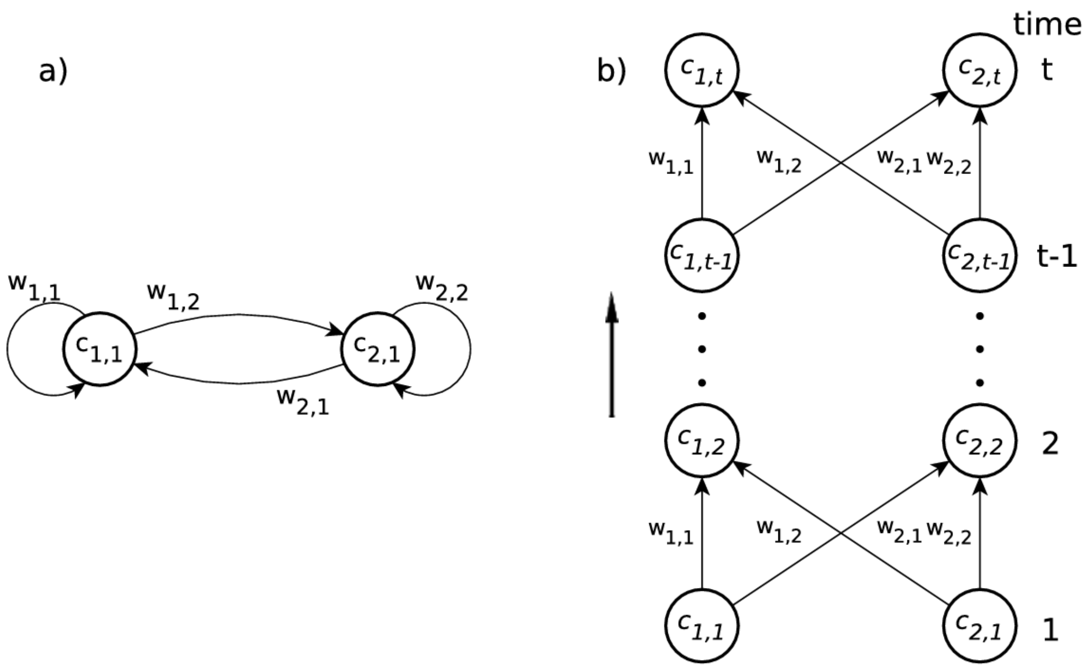


Figure 7.7: (a) It shows a simple fully recurrent neural network with a two neuron layer. (b) The same network unfolded over time, with a separate layer for each time step. This is the form of a feed-forward neural network.

The unfolded network can be trained using the backpropagation algorithm described before. At the end of a training sequence, the network is unfolded in time. The error is calculated for the output units with existing target values using some chosen error measure. Then, the error is injected backwards into the network, and the weight updates for all time steps are calculated. The weights in the recurrent version of the network are updated with the sum of their deltas over all time steps.

We calculate the error signal for a unit for all time steps in a single pass using the following iterative backpropagation algorithm. We consider discrete time steps $1, 2, 3, \ldots$, indexed by the variable τ. The network starts at a point in time t' and runs until a final time t. This time frame between t' and t is called an epoch. Let U be the set of non-input units, and let f_u be the differentiable, non-linear squashing function of the unit $u \in U$; the output $y_u(\tau)$ of u at time τ is given by

$$y_u(\tau) = f_u(z_u(\tau)) \tag{7.24}$$

with the weighted input

$$z_u(\tau+1) = \sum_l W_{[u,l]} X_{[u,l]}(\tau+1) \quad , \quad l \in \mathrm{Pre}(u) \tag{7.25}$$
$$= \sum_v W_{[u,v]} y_v(\tau) + \sum_i W_{[u,i]} y_i(\tau+1)$$

where $v \in U \cap \mathrm{Pre}(u)$ and $i \in I$, is the set of input units. Note that the inputs to u at time $\tau+1$ are of two types: the environmental input that arrives at time $\tau+1$ via the input units, and the recurrent output from all non-input units in the network produced at time τ. If the network is fully connected, then $U \cap \mathrm{Pre}(u)$ is equal to the set U of non-input units. Let $T(\tau)$ be the set of non-input units for which, at time τ, the output value $y_u(\tau)$ of the unit $u \in T(\tau)$ should match some target value $d_u(\tau)$. The cost function is the summed error $E_{total}(t', t)$ for the epoch $t', t'+1, \ldots, t$, which we want to minimize using a learning algorithm. Such total error is defined by

$$E_{total}(t', t) = \sum_{\tau=t'}^{t} E(\tau) \tag{7.26}$$

with the error $E(\tau)$ at time τ defined using the squared error as an objective function by

$$E(\tau) = \frac{1}{2} \sum_{u \in U} (e_u(\tau))^2 \tag{7.27}$$

and with the error $e_u(\tau)$ of the non-input unit u at time τ defined by

$$e_u(\tau) = \begin{cases} d_u(\tau) - y_u(\tau) & , \quad \text{if} \quad u \in T(\tau) \\ 0 & , \quad \text{otherwise} \end{cases} \tag{7.28}$$

To adjust the weights, we use the error signal $\vartheta_u(\tau)$ of a non-input unit u at a time τ, which is defined by

$$\vartheta_u(\tau) = \frac{\partial E(\tau)}{\partial z_u(\tau)} \tag{7.29}$$

When we unroll ϑ_u over time, we obtain the equality

$$\vartheta_u(\tau) = \begin{cases} f'_u(z_u(\tau)) e_u(\tau) & , \quad \text{if} \quad \tau = t \\ f'_u(z_u(\tau)) \left(\sum_{k \in U} w_{ku} \vartheta_k(\tau+1) \right) & , \quad \text{if} \quad t' \le \tau < t \end{cases} \tag{7.30}$$

After the backpropagation computation is performed down to time t', we calculate the weight update Δw_{uv} in the recurrent version of the network. This is done by summing the corresponding weight updates for all time steps:

$$\Delta w_{uv} = -\eta \frac{\partial E_{total}(t', t)}{\partial w_{uv}} \tag{7.31}$$

with

$$\frac{\partial E_{total}(t', t)}{\partial w_{uv}} = \sum_{\tau=t'}^{t} \vartheta_u(\tau) \frac{\partial z_u(\tau)}{\partial w_{uv}} \tag{7.32}$$
$$= \sum_{\tau=t'}^{t} \vartheta_u(\tau) x_{uv}(\tau)$$

7.7.4 Solving the vanishing gradient problem

Standard RNN can not bridge more than 5–10 time steps [76]. This is due to that back-propagated error signals tend to either grow or shrink with every time step. Over many time steps, the error therefore typically blows-up or vanishes [77, 78]. Blown-up error signals lead straight to oscillating weights, whereas with a vanishing error, learning takes an unacceptable amount of time, or does not work at all.

The explanation of how gradients are computed by the standard backpropagation algorithm and the basic vanishing error analysis is as follows: we update weights after the network has trained from time t to time t' using the formulas (7.31), (7.32) and (7.30). Consequently, given a fully recurrent neural network with a set of non-input units U, the error signal that occurs at any chosen output-layer neuron $o \in O$, at time-step τ, is propagated back through time for $t - t'$ time-steps, with $t' < t$ to an arbitrary neuron v. This causes the error to be scaled by the following factor:

$$\frac{\partial \vartheta_v(t')}{\partial \vartheta_o(t)} = \begin{cases} f'_v(z_v(t'))w_{ov}(\tau) & , \quad \text{if} \quad t-t'=1 \\ f'_v(z_v(t'))\left(\sum_{u\in U} w_{uv}\frac{\partial \vartheta_u(t'+1)}{\partial \vartheta_o(t)}\right) & , \quad \text{if} \quad t-t'>1 \end{cases} \tag{7.33}$$

To solve the above equation, we unroll it over time. For $t' \leq \tau \leq t$, let u_τ be a non-input-layer neuron in one of the replicas in the unrolled network at time τ. Now, by setting $u_t = v$ and $u_{t'} = o$, we obtain the equation

$$\frac{\partial \vartheta_v(t')}{\partial \vartheta_o(t)} = \sum_{u_{t'}\in U} \cdots \sum_{u_{t-1}\in U} \left(\prod_{\tau=t'+1}^{t} f'_{u_\tau}(z_{u_\tau}(t-\tau+t'))w_{u_\tau u_{\tau-1}} \right) \tag{7.34}$$

Observing Equation (7.34), it follows that if

$$|f'_{u_\tau}(z_{u_\tau}(t-\tau+t'))w_{u_\tau u_{\tau-1}}| > 1 \tag{7.35}$$

for all τ, then the product will grow exponentially, causing the error to **blow-up**; moreover, conflicting error signals arriving at neuron v can lead to oscillating weights and unstable learning. If now

$$|f'_{u_\tau}(z_{u_\tau}(t-\tau+t'))w_{u_\tau u_{\tau-1}}| < 1 \tag{7.36}$$

for all τ, then the product decreases exponentially, causing the error to **vanish**, preventing the network from learning within an acceptable time period. Finally, the equation

$$\sum_{o\in O} \frac{\partial \vartheta_v(t')}{\partial \vartheta_o(t)} \tag{7.37}$$

shows that if the local error vanishes, then the global error also vanishes. A more detailed theoretical analysis of the problem with long-term dependencies is presented in [79]. The paper also briefly outlines several proposals on how to address this problem.

7.8 Long Short-Term Memory

One solution that addresses the vanishing error problem is a gradient-based method called **Long Short-Term Memory (LSTM)** published by [80, 78, 76, 81]. LSTM can learn how to bridge minimal time lags of more than 1,000 discrete time steps. The solution uses **constant error carousels (CECs)**, which enforce a constant error flow within special cells. Access to the cells is handled by multiplicative gate units, which learn when to grant access.

7.8.1 Constant Error Carousel

Suppose that we have only one unit u with a single connection to itself. The local error back flow of u at a single time-step τ follows from Equation (7.30) and is given by

$$\vartheta_u(\tau) = f'_u(z_u(\tau))w_{uu}\vartheta_u(\tau+1) \tag{7.38}$$

From Equations (7.36) and (7.37) we see that, in order to ensure a constant error flow through u, we need to have

$$f'_u(z_u(\tau))w_{uu} = 1 \tag{7.39}$$

and by integration we have

$$f_u(z_u(\tau)) = \frac{z_u(\tau)}{w_{uu}} \tag{7.40}$$

From this, we learn that f_u must be linear, and that u's activation must remain constant over time; i.e.,

$$y_u(\tau+1) = f_u(z_u(\tau+1)) = f_u(y_u(\tau)w_{uu}) = y_u(\tau) \tag{7.41}$$

This is ensured by using the identity function as f_u, and by setting $w_{uu} = 1$. This preservation of error is called the constant error carousel (CEC), and it is the central feature of LSTM, where short-term memory storage is achieved for extended periods of time. Clearly, we still need to handle the connections from other units to the unit u, and this is where the different components of LSTM networks come into the picture.

7.8.2 Memory blocks

In the absence of new inputs to the cell, we now know that the CEC's backflow remains constant. However, as part of a neural network, the CEC is not only connected to itself but also to other units in the neural network. We need to take these additional weighted inputs and outputs into account. Incoming connections to neuron u can have conflicting weight update signals because the same weight is used for storing and ignoring inputs. For weighted output connections from neuron u, the same weights can be used to both retrieve u's contents and prevent u's output flow to other neurons in the network. To address the problem of conflicting weight updates, LSTM extends the CEC with input and output gates connected to the network input layer and to other memory cells. This results in a more complex LSTM unit, called a memory block; its standard architecture is shown in Figure 7.8. The input gates, which are simple sigmoid threshold units with an activation function range of $[0, 1]$, control the signals from the network to the memory cell by scaling them appropriately; when the gate is closed, activation is close to zero.

Additionally, these can learn to protect the contents stored in u from disturbance by irrelevant signals. The activation of a CEC by the input gate is defined as the cell state. The output gates can learn how to control access to the memory cell contents, which protects other memory cells from disturbances originating from u. So we can see that the basic function of multiplicative gate units is to either allow or deny access to constant error flow through the CEC.

7.8.3 Architecture

Long Short-Term Memory (LSTM), as already referred, is a recurrent neural network architecture designed by Sepp Hochreiter and Jürgen Schmidhuber in 1997. The LSTM architecture consists of one unit, the **memory unit**, also known as the **LSTM unit**. The LSTM unit is made up of **four feed-forward neural networks**. Each of these neural networks consists of an input layer and an output layer. In each of these neural networks, input neurons are connected to all output neurons. As a result, the LSTM unit has four fully connected layers. Three of the four feed-forward neural networks are responsible for selecting information and performing the typical memory management operations. The fourth neural network is used to create new candidate information. More specific, these neural networks are the following:

1. **Forget gate**: responsible for the deletion of information from memory.
2. **Input gate**: responsible for the insertion of new information in memory.
3. **Output gate**: responsible for the use of information present in memory.
4. **Candidate memory**: responsible for creating new candidate information to be inserted into the memory.

Figure 7.8 illustrates the architecture of a typical LSTM unit. Next, we will delve into the details of its components.

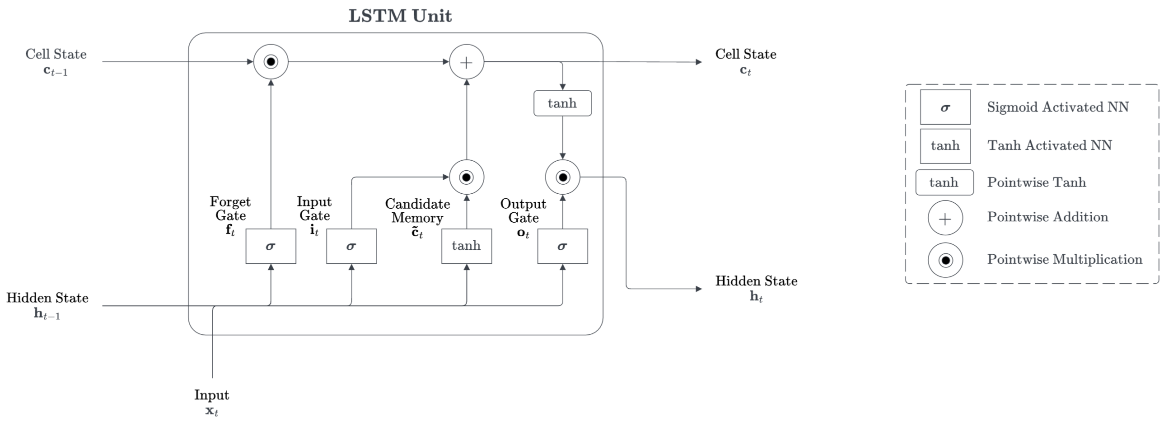

LSTM Unit
Cell State $\mathbf{c}_{t-1}$
Cell State $\mathbf{c}_t$
tanh
Forget Gate $\mathbf{f}_t$
Input Gate $\mathbf{i}_t$
Candidate Memory $\tilde{\mathbf{c}}_t$
Output Gate $\mathbf{o}_t$
σ
tanh
Hidden State $\mathbf{h}_{t-1}$
Hidden State $\mathbf{h}_t$
Input $\mathbf{x}_t$
Sigmoid Activated NN
Tanh Activated NN
Pointwise Tanh
Pointwise Addition
Pointwise Multiplication

Figure 7.8: Architecture of a typical vanilla LSTM unit.

7.8.4 Input and output

An LSTM unit receives three vectors as input. Two vectors come from the LSTM itself and were generated by the LSTM at the previous instant, the instant $t-1$. These are the cell state, $\mathbf{c} \in \mathbb{R}^Q$, and the hidden state, $\mathbf{h} \in \mathbb{R}^Q$. The third vector comes from outside. This is the vector $\mathbf{x} \in \mathbb{R}^F$, called input vector, submitted to the LSTM at instant t. We call Q the "output dimension" or "hidden size", and F the "input dimension" which is essentially the number of features in the input vector.

Given the three input vectors $\mathbf{c}$, $\mathbf{h}$, and $\mathbf{x}$, the LSTM regulates, through the gates, the internal flow of information and transforms the values of the cell state and hidden state vectors. Vectors that will be part of the LSTM input set in the next instant, the instant $t+1$. Information flow control is done so that the **cell state acts as a long-term memory**, while the **hidden state acts as a short-term memory**.

In practice, the LSTM unit uses recent past information (the short-term memory, $\mathbf{h}$) and new information coming from the outside (the input vector, $\mathbf{x}$) to update the long-term memory (cell state, $\mathbf{c}$). Finally, it uses the long-term memory (the cell state, $\mathbf{c}$) to update the short-term memory (the hidden state, $\mathbf{h}$). The **hidden state** determined in instant t is also the **output** of the LSTM unit in instant t. It is what the LSTM provides to the outside for the performance of a specific task. In other words, it is the behavior on which the performance of the LSTM is assessed [82].

7.8.5 Gates

The three gates (forget gate, input gate, and output gate) are **information selectors** [82]. Their task is to create selector vectors. A selector vector is a vector with values between zero and one and near these two extremes. A selector vector is created to be multiplied, element by element, by another vector of the same size. This means that a position where the selector vector has a value equal to zero completely eliminates the information included in the same position in the other vector, when doing element-wise (or point-wise) multiplication. A position where the selector vector has a value equal to one leaves unchanged the information included in the same position in the other vector. All three gates are neural networks that use the sigmoid function as the activation function in the output layer. The sigmoid function is used to produce, as an output, a vector composed of values between zero and one and near these two extremes. All three gates use as input the vector $\mathbf{x}_t$ and the hidden state vector coming from the previous instant $\mathbf{h}_{t-1}$.

7.8.6 Forget gate

At any time t, an LSTM receives an input vector $\mathbf{x}_t$ as an input. It also receives the hidden state $\mathbf{h}_{t-1}$ and cell state $\mathbf{c}_{t-1}$ vectors determined in the previous instant at $t-1$. The first activity of the LSTM unit is executed by the forget gate. The **forget gate** decides, based on $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$, what information to remove from the cell state vector coming from time $t-1$ [82]. The outcome of this decision is a selector vector. Formally. it is

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{R}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \tag{7.42}$$

where $\mathbf{W}_f \in \mathbb{R}^{Q \times F}$, and $\mathbf{R}_f \in \mathbb{R}^{Q \times Q}$ are the weights associated with $\mathbf{x}_t$, and $\mathbf{h}_{t-1}$, respectively, and $\mathbf{b}_f \in \mathbb{R}^Q$ represents the bias vector associated with the forget gate.

7.8.7 Input gate and candidate memory

After removing some of the information from the cell state received in input ($\mathbf{c}_{t-1}$), we can insert a new one. This activity is carried out by two neural networks: the **candidate memory** and the **input gate**. The two neural networks are independent of each other, and their inputs are the vectors $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$.

The candidate memory is responsible for the generation of a candidate vector: a vector of information that is candidate to be added to the cell state [82]. Candidate memory output neurons use hyperbolic tangent function. The properties of this function ensure that all values of the candidate vector are between -1 and 1. This is used to normalize the information that will be added to the cell state. The input gate is responsible for the generation of a selector vector, which will be multiplied element-wise with the candidate vector [82]. Formally, it is

$$\tilde{\mathbf{c}}_t = \tanh\left(\mathbf{W}_{\tilde{c}}\mathbf{x}_t + \mathbf{R}_{\tilde{c}}\mathbf{h}_{t-1} + \mathbf{b}_{\tilde{c}}\right) \tag{7.43}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{R}_i\mathbf{h}_{t-1} + \mathbf{b}_i) \tag{7.44}$$

where $\mathbf{W}_{\tilde{c}} \in \mathbb{R}^{Q \times F}$, and $\mathbf{R}_{\tilde{c}} \in \mathbb{R}^{Q \times Q}$ are the weights associated with $\mathbf{x}_t$, and $\mathbf{h}_{t-1}$, respectively, and $\mathbf{b}_{\tilde{c}} \in \mathbb{R}^Q$ represents the bias vector associated with the candidate memory, while $\mathbf{W}_i \in \mathbb{R}^{Q \times F}$, and $\mathbf{R}_i \in \mathbb{R}^{Q \times Q}$ are the weights associated with $\mathbf{x}_t$, and $\mathbf{h}_{t-1}$, respectively, and $\mathbf{b}_i \in \mathbb{R}^Q$ represents the bias vector associated with the input gate

7.8.8 Output Gate

The **output gate** essentially determines the value of the hidden state outputted by the LSTM instant at t and received by the LSTM in the next instant at $t+1$. A selector vector is generated from the output gate based on the values of $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ it receives as input [82]. The output gate uses the sigmoid function as the activation function of the output neurons. Formally, it is

$$\mathbf{o}_t = \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{R}_o\mathbf{h}_{t-1} + \mathbf{b}_o) \tag{7.45}$$

where $\mathbf{W}_o \in \mathbb{R}^{Q \times F}$, and $\mathbf{R}_o \in \mathbb{R}^{Q \times Q}$ are the weights associated with $\mathbf{x}_t$, and $\mathbf{h}_{t-1}$, respectively, and $\mathbf{b}_o \in \mathbb{R}^Q$ represents the bias vector associated with the output gate.

7.8.9 Cell state and hidden state

The selector vector $\mathbf{f}_t$ (calculated by the forget gate) is multiplied element-wise with the vector of the cell state $\mathbf{c}_{t-1}$ received as input by the LSTM unit [82]. This means that a position where the selector vector has a value equal to zero completely eliminates the information included in the same position in the cell state. A position where the selector vector has a value equal to one leaves unchanged the information included in the same position in the cell state.

Furthermore, the selector vector $\mathbf{i}_t$ (calculated by the input gate) is multiplied element-wise with the vector of the candidate memory $\tilde{\mathbf{c}}_t$ [82]. This means that a position where the selector vector has a value equal to zero completely eliminates the information included in the same position in the candidate vector. A position where the selector vector has a value equal to one leaves unchanged the information included in the same position in the candidate vector. The result of the multiplication between the candidate vector and the selector vector is added to the cell state vector. This adds new information to the cell state.

Now, we **update the cell state** by combining the above operations. The cell is used by the output gate and passed into the input set used by the LSTM unit in the next instant at $t+1$. Formally, it is

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \tag{7.46}$$

where $\odot$ is the element-wise multiplication, $\mathbf{f}_t \in \mathbb{R}^Q$ is the selector vector produced by the forget gate, $\mathbf{i}_t \in \mathbb{R}^Q$ is the selector vector produced by the input gate, $\tilde{\mathbf{c}}_t \in \mathbb{R}^Q$ is the selector vector produced by the candidate memory, and $\mathbf{c}_{t-1} \in \mathbb{R}^Q$ is the cell state that LSTM received as input from the previous instant $t-1$.

Output generation also works with a multiplication between a selector vector and a candidate vector [82]. In this case, however, the candidate vector is **not** generated by a neural network, but it is obtained simply by using the hyperbolic tangent function on the cell state vector. This step makes the vector values of the cell state normalized within a range of -1 to 1. In this way, after multiplying with the selector vector (whose values are between zero and one), we get a hidden state with values between -1 and 1. This makes it possible to control the stability of the network over time. The selector vector and the candidate vector are multiplied with each other element-wise [82]. This means that a position where the selector vector has a value equal to zero completely eliminates the information included in the same position in the candidate vector. A position where the selector vector has a value equal to one leaves unchanged the information included in the same position in the candidate vector. Formally, it is

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right) \tag{7.47}$$

where $\odot$ is the element-wise multiplication, $\mathbf{o}_t \in \mathbb{R}^Q$ is the selector vector produced by the output gate, and $\mathbf{c}_t \in \mathbb{R}^Q$ is the update cell state at instant t. Finally, the output of the whole LSTM unit is $\mathbf{h}_t \in \mathbb{R}^Q$.

7.8.10 Dimensionality, weights, and biases

From the above analysis, note that the weight matrices $\mathbf{W}$ and $\mathbf{R}$, as well as the biases $\mathbf{b}$, of each component, do not change with time unroll. This means that once the LSTM is trained, the weight matrices are fixed during inference and not time-dependent. In other words, the **same** weight matrices and biases are used in every time step.

The total weight-bias parameters of the LSTM are

$$\text{num\_parameters} = 4 \cdot (Q \cdot F) + 4 \cdot (Q \cdot Q) + 4 \cdot F \tag{7.48}$$

and the matrix containing all the trained weight-bias parameters will be of the following form

$$\begin{bmatrix} \mathbf{W}_f & \mathbf{R}_f & \mathbf{b}_f \\ \mathbf{W}_i & \mathbf{R}_i & \mathbf{b}_i \\ \mathbf{W}_o & \mathbf{R}_o & \mathbf{b}_o \\ \mathbf{W}_c & \mathbf{R}_c & \mathbf{b}_c \end{bmatrix} \tag{7.49}$$

7.9 Stacked Long Short-Term Memory

Stacked Long Short-Term Memory (Stacked LSTMs) networks represent a powerful and sophisticated architecture within the realm of recurrent neural networks (RNNs). In the pursuit of modeling complex temporal dependencies in sequential data, Stacked LSTMs have emerged as a valuable tool. They extend the capabilities of their single-layer counterparts, allowing for the hierarchical extraction of patterns and information from sequential data. Stacked LSTMs or Deep LSTMs were introduced by Graves, *et al.* [83, 84]. Given that LSTMs operate on sequence data, the addition of layers adds levels of abstraction to input observations over time. In effect, chunking observations over time or representing the problem at different time scales.

Stacked LSTMs are a stable technique for challenging sequence prediction problems. A Stacked LSTM architecture can be defined as an LSTM model comprised of multiple LSTM layers. An LSTM layer above provides a sequence output rather than a single value output to the LSTM layer below. Specifically, it is one output per input time step, rather than one output time step for all input time steps. Figure 7.9 illustrates the architecture of a stacked LSTM neural network. In this type of neural network, each layer consists of a different LSTM unit with its own weights and biases. As the output of the whole structure, we define the hidden state of the topmost layer. In order to increase the flexibility and ability of the model to capture patterns, it is often to choose different hidden size Q from the output size R, and use an intermediate dense layer that performs the dimensionality transformation from one to the other. The dense layer is assumed to be a part of the whole structure, and its weights and biases are trained with backpropagation during the training phase.

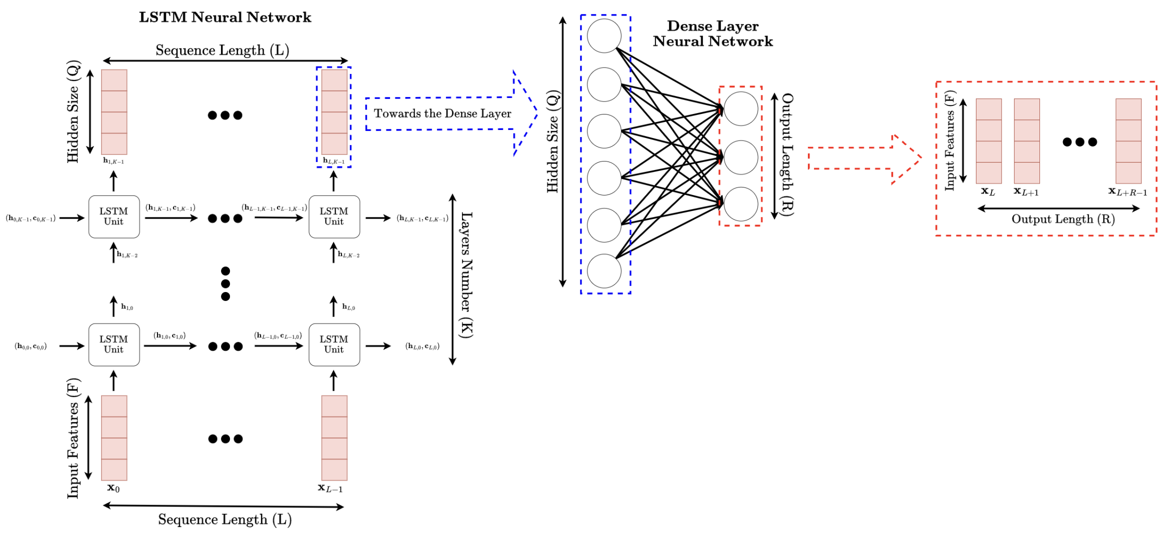


Figure 7.9: Architecture of a stacked LSTM neural network. This image depicts the network unrolled in time (i.e. the horizontal axis). A dense layer receives the output of the stacked LSTM in order to perform a linear transformation and get the data to the desired dimensionality.

8 A Concise Overview of Heliophysics

8.1 Sun

The content of this subsection is based on information about the solar system that is publicly available from NASA's website [85] (except 8.1.4 whose bibliographic references are explicitly written in the text).

8.1.1 General characteristics

The Sun, situated at the heart of our solar system, is a yellow dwarf star that has been shining for approximately 4.5 billion years. It emits a radiant glow generated by its hydrogen and helium composition. Positioned roughly 150 million kilometers away from Earth, it serves as the sole star in our solar system. The Sun's vitality is crucial for the sustenance of life on our planet, as it supplies the energy necessary for existence. Although from our terrestrial standpoint, the Sun may seem like a constant source of light and warmth, it is, in fact, a dynamic celestial body that undergoes continuous transformations while emitting energy into space. The field of study that focuses on comprehending the Sun and its impact on the solar system is referred to as Heliophysics. While the Sun holds a central role in our solar system and is indispensable for our survival, it possesses an average magnitude compared to other stars, some of which can be up to 100 times larger. Moreover, numerous solar systems consist of multiple stars. Through the examination of our Sun, scientists can enhance their understanding of the mechanisms at work in distant stars.

The Sun, our medium-sized star, boasts a radius of approximately $700,000$ kilometers. While there are stars of larger dimensions in the universe, the Sun surpasses our home planet in terms of mass significantly. To put it into perspective, it would require over $330,000$ Earths to equal the Sun's mass and a staggering 1.3 million Earths to fill the Sun's volume. The Sun's closest stellar companion is the Alpha Centauri triple star system.

The core of the Sun holds the title for being the hottest region, with temperatures soaring above 15 million degrees Celsius. In contrast, the portion we refer to as the surface or photosphere is comparatively cooler, measuring around $5,500$ degrees Celsius. One of the most perplexing enigmas surrounding the Sun lies in its outer atmosphere known as the corona. Astonishingly, the corona becomes increasingly hotter as it extends farther away from the surface. It reaches scorching temperatures of up to 2 million degrees Celsius, surpassing the photosphere by a significant margin. Due to its extreme temperatures and radiation, the Sun cannot sustain life as we know it. Nevertheless, life on Earth thrives solely because of the Sun's light and energy.

Within the vast expanse of the Milky Way galaxy, the Sun finds its abode in a spiral arm known as the Orion Spur, which extends outward from the Sagittarius arm. Orbiting around the center of the Milky Way, the Sun carries along with it a retinue of planets, asteroids, comets, and various celestial objects that compose our solar system. In our cosmic journey, our solar system maintains an average velocity of $720,000$ kilometers per hour. However, even at this rapid speed, it takes approximately 230 million years for the Sun to complete a single revolution around the Milky Way. As the Sun traverses its path around the galaxy, it also rotates on its own axis. With a tilt of 7.25 degrees in relation to the plane of the planets' orbits, the Sun's non-solid structure causes different parts to rotate at varying rates. At the equator, the Sun completes a full rotation approximately every 25 Earth days, while at its poles, the Sun rotates once on its axis every 36 Earth days.

Approximately 4.6 billion years ago, the Sun took shape within a colossal, swirling mass of gas and dust known as the solar nebula. As the nebula contracted due to its gravitational forces, it underwent increased rotation and transformed into a flattened disk. The majority of the nebula's material migrated towards the center, culminating in the formation of our Sun, which accounts for a staggering 99.8% of the total mass in our solar system. The remaining substances coalesced to give rise to the planets and other celestial bodies that presently orbit the Sun. The surplus gas and dust that remained were expelled by the youthful Sun's early solar wind. Like all stars, the Sun will eventually deplete its energy reserves. As it nears the end of its life cycle, the Sun will expand into a red giant, growing so immense that it may engulf Mercury, Venus, and potentially Earth as well. Scientists estimate that the Sun is currently at approximately the halfway point of its lifespan and has approximately another 5 billion years left before it transitions into a white dwarf.

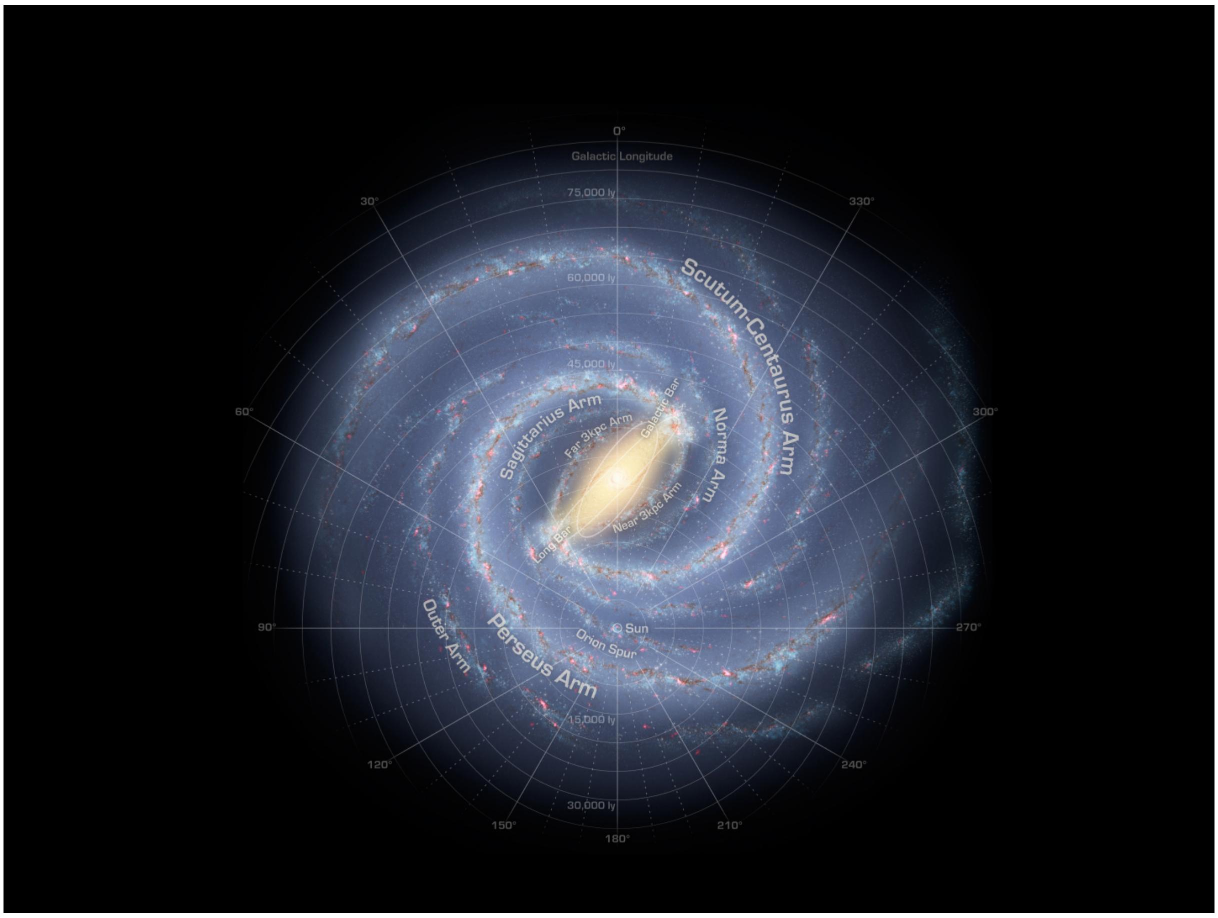


Figure 8.1: Graphic view of our Milky Way Galaxy. The Milky Way Galaxy is organized into spiral arms of giant stars that illuminate interstellar gas and dust. The Sun is on a finger called the Orion Spur. Overlaid is a graphic of galactic longitude in relation to our Sun. Credit: NASA/Adler/U. Chicago/Wesleyan/JPL-Caltech

8.1.2 Structure

The Sun, an immense celestial body composed predominantly of hydrogen and helium, remains intact through the force of its own gravitational pull. The remaining is composed of trace amounts of heavier elements, including oxygen, carbon, neon, and iron. The Sun can be divided into various distinct regions. Starting from the interior, we encounter the **core**, the **radiative zone**, and the **convection zone**. Progressing outward, we reach the visible surface referred to as the **photosphere**, followed by the **chromosphere**, the transition zone, and finally, the **corona** — an expansive outer atmosphere enveloping the Sun. Among these regions, the core stands out as the hottest segment. It is within the core that nuclear reactions occur, fusing hydrogen nuclei to form helium, and generating the Sun's abundant heat and luminosity.

Unlike Earth and other solid planets and moons, the Sun lacks a solid surface. However, the portion commonly referred to as the "surface" of the Sun is known as the **photosphere**. The term photosphere, meaning "light sphere", is fitting as it emits the majority of visible light, making it observable from Earth with our eyes. Although commonly termed the surface, the photosphere is actually the first layer of the solar atmosphere. It possesses a thickness of approximately 402 kilometers and experiences temperatures of around $5,500$ degrees Celsius. While significantly cooler than the scorching core, it remains hot enough to cause carbon, including diamonds and graphite, to not only melt but also vaporize. The photosphere serves as the primary source of the Sun's radiation, with most of it escaping outward into space.

Above the photosphere, we find the **chromosphere**, the **transition zone**, and the **corona**. While the chromosphere and corona are commonly recognized as distinct regions, not all scientists categorize the transition zone as an individual layer. Instead, the transition zone denotes the narrow region where the chromosphere undergoes rapid heating, eventually merging into the corona. Together, the photosphere, chromosphere, and corona comprise the Sun's **atmosphere**. It is within this atmospheric realm that notable features become visible, including **sunspots**, **coronal holes**, and **solar flares**. These phenomena contribute to the dynamic nature of the Sun and offer intriguing areas of study.

Typically, the visible light emitted from the upper regions of the Sun is overshadowed by the brightness of the photosphere, making it difficult to observe. However, during total solar eclipses, when the Moon aligns to cover the photosphere, the chromosphere becomes visible as a delicate red rim encircling the Sun. Simultaneously, the corona manifests as a stunning white crown, exhibiting plasma streamers that extend outward and resemble the shapes of flower petals.

One of the most significant mysteries surrounding the Sun lies in the corona's significantly higher temperature compared to the layers situated directly beneath it. This disparity in temperature poses a major unsolved puzzle in solar research and is known as the **coronal heating problem**. Scientists are still actively studying and seeking explanations for the mechanisms responsible for the intense heating of the corona.

8.1.3 Solar activity

The Sun exhibits variable behavior and undergoes distinct phases of high and low activity, collectively known as the **solar cycle**. Roughly every 11 years, the geographic

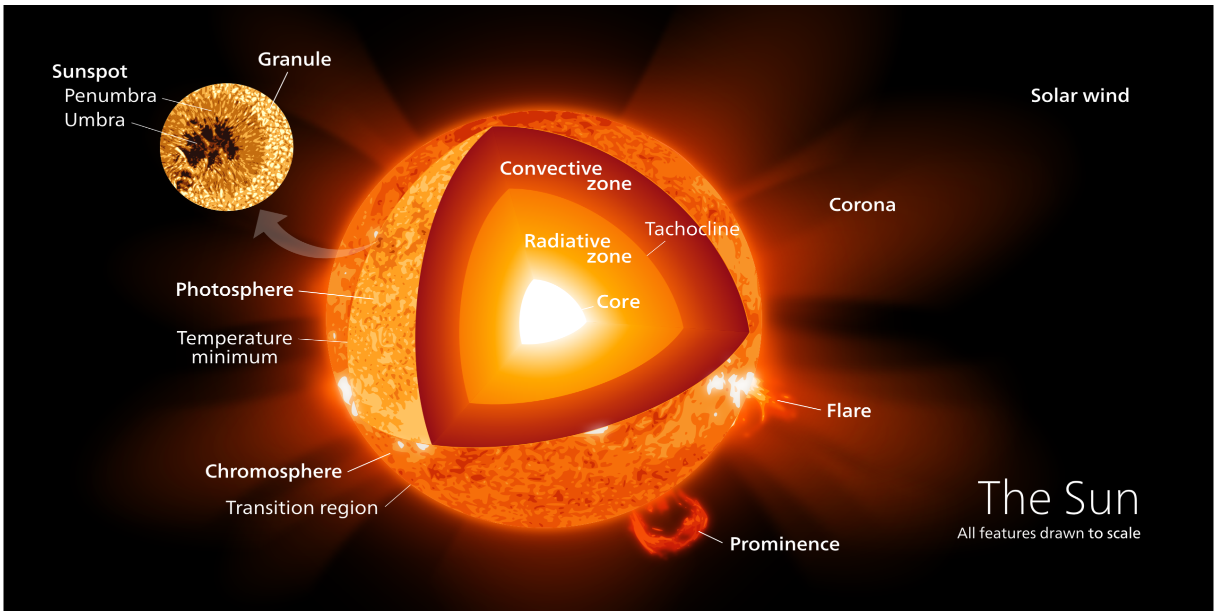


Figure 8.2: Illustration of the Sun's structure, in false color for contrast. Credit: Wikipedia, "Sun"

poles of the Sun experience a reversal of their magnetic polarity, resulting in the swapping of the north and south magnetic poles. Throughout this cycle, the photosphere, chromosphere, and corona of the Sun transition from a state of tranquility to one of vigorous activity. The peak of this activity cycle, referred to as solar maximum, corresponds to a period of significantly increased solar storm activity. During solar maximum, phenomena such as sunspots, intense eruptions known as solar flares, and coronal mass ejections become more frequent occurrences.

Solar activity possesses the capability to release immense quantities of energy and particles, a portion of which can have an impact on Earth. Similar to the dynamic nature of weather on our planet, conditions in space, known as **space weather**, are in a constant state of flux due to the Sun's activity. Space weather phenomena can pose challenges and disruptions to various technological systems and infrastructure on Earth. For instance, it can interfere with satellite operations, GPS navigation, and radio communications. Moreover, space weather events have the potential to cripple power grids and cause corrosion in pipelines that transport oil and gas. Thus, understanding and monitoring space weather is crucial for mitigating and managing the potential risks and impacts it can have on our technological systems and infrastructure.

8.1.4 Magnetic field

The Sun possesses a large and complex magnetic field. Magnetic fields are generated by magnetic objects or moving charged particles, and they describe the force exerted by these objects in their surrounding space. The Sun's magnetic field is created by the movement of charged particles within its plasma, consisting of positively charged ions and negatively charged electrons [86].

The Sun's high temperatures cause the plasma to move vigorously, creating intricate and twisted magnetic fields. Additionally, the solar wind, consisting of extremely hot plasma blown off the Sun's surface, contributes to the magnetic field. The plasma near the poles rotates at a slower rate than at the equator, resulting in the twisting and stretching of the magnetic fields. The interaction and influence of these plasma motions give rise to the Sun's complex magnetic field.

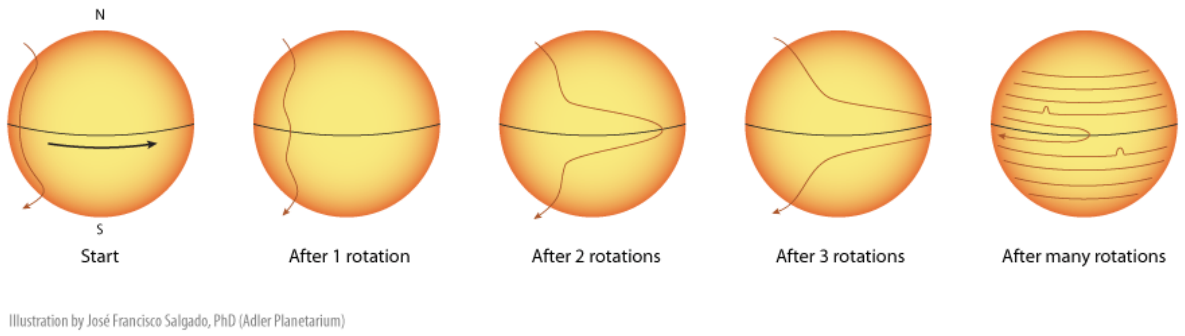


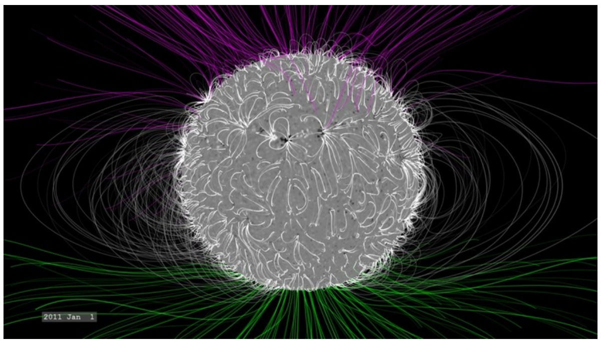
Figure 8.3: The Sun's magnetic field is shown in a series of illustrated images with the poles and equator indicated. The magnetic field lines change as the Sun rotates. After 1, 2, and 3 rotations the magnetic field line gets progressively wrapped around the Sun, becoming stretched as it nears the equator. After many rotations the magnetic field is complex and wrapped tightly around the Sun in many loops. Credit: José Francisco Salgado, PhD (Adler Planetarium)

Close to the Sun's surface, the magnetic fields are intricate and twisted, while further away, certain trends emerge. The magnetic field strength is stronger near the poles and weaker at the equator, although it remains approximately 100 times stronger than Earth's magnetic field even at the equator. The Sun's magnetic field extends well beyond the orbits of the known planets, reaching distances of around 75-100 times the distance between the Earth and the Sun [86].

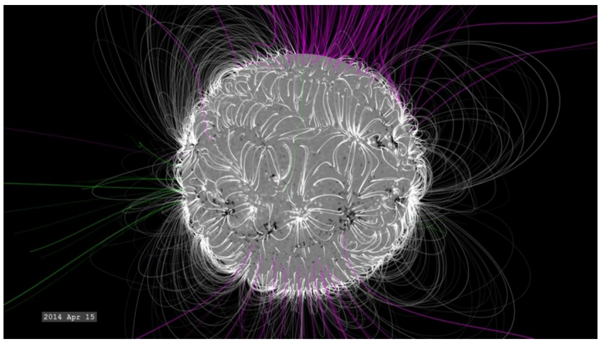
During the approximately 11-year sunspot cycle, the Sun's magnetic field undergoes reversals. Magnetograms from instruments like SOHO/MDI and SDO/HMI are used to analyze the configurations of the magnetic field above the Sun's surface. These configurations play a crucial role in understanding the potential conditions of severe space weather [87]. The Sun's magnetic field can be observed through the behavior of solar material. The Sun's plasma, consisting of charged particles, creates magnetic fields as the particles move. These invisible magnetic fields guide the motion of the plasma and can be visualized through loops and towers of material in the corona, which emit light in wavelengths invisible to the human eye. Instruments like magnetographs are used to measure the strength and direction of magnetic fields at the footpoints of these loops on the Sun's surface [88].

To gain a deeper understanding of the Sun's magnetic field, scientists utilize models and simulations. The Potential Field Source Surface (PFSS) model, for example, illustrates the undulating nature of the magnetic fields around the Sun. These models combine observational measurements with knowledge of solar material movement and magnetism to provide insights into the structure and behavior of the Sun's magnetic field, both in the corona and on the far side of the Sun.

While a complete understanding of the Sun's magnetic field is still evolving, scientists have made significant progress. The Sun's magnetic field is responsible for driving its approximately 11-year activity cycle, with periods of maximum and minimum solar ac-



(a) In January 2011, three years after solar minimum, the magnetic field of the sun is still relatively simple, with open field lines concentrated near the poles. Credit: NASA's Goddard Space Flight Center/Bridgman.



(b) In July 2014, at solar maximum, the structure is much more complex, with closed and open sun's magnetic field lines poking out all over – ideal conditions for solar explosions. Credit: NASA's Goddard Space Flight Center/Bridgman.

Figure 8.4: This comparison shows the relative complexity of the solar magnetic field between (a) January 2011 and (b) July 2014. Credit: NASA's Goddard Space Flight Center/Bridgman.

tivity. During solar maximum, the magnetic field becomes highly complex, characterized by numerous active regions. In contrast, during solar minimum, the field is weaker and concentrated at the poles, without the formation of sunspots. By studying the changes in the magnetic fields over time, scientists can observe the evolution from concentrated and smooth structures near solar minimum to more tangled and disorderly fields during solar maximum. These changing magnetic conditions contribute to solar events such as flares and coronal mass ejections, highlighting the dynamic nature of the Sun's magnetic field and its impact on space weather phenomena [88].

8.2 Solar wind

The solar wind is a continuous stream of charged particles, mainly protons and electrons, flowing outward from the Sun's corona. It is formed by the expansion of plasma from the outermost atmosphere of the Sun, known as the corona. The plasma is heated to such high temperatures that the Sun's gravity cannot hold it down, and it is propelled outward. This plasma follows the Sun's magnetic field lines, which extend radially outward and create a spiral pattern due to the Sun's rotation. The solar wind is generated by these expanding plasma streamers, which originate from large bright patches called "coronal holes" in the Sun's corona [89].

As the solar wind travels away from the Sun, it becomes thin and encounters the interstellar medium, which is the space between stars. It passes through a shock wave called the "termination shock", slows down, and enters a subsonic flow region known as the heliosheath. The heliosheath extends to the heliopause, the boundary where the heliosphere (the region influenced by the Sun's magnetic field) meets the interstellar medium. Similar to Earth's magnetic shield, the solar wind interacts with Earth's magnetosphere when it reaches our planet. Earth's magnetic shield, known as the magnetosphere, deflects the solar wind, causing most of its energetic particles to flow around and beyond Earth.

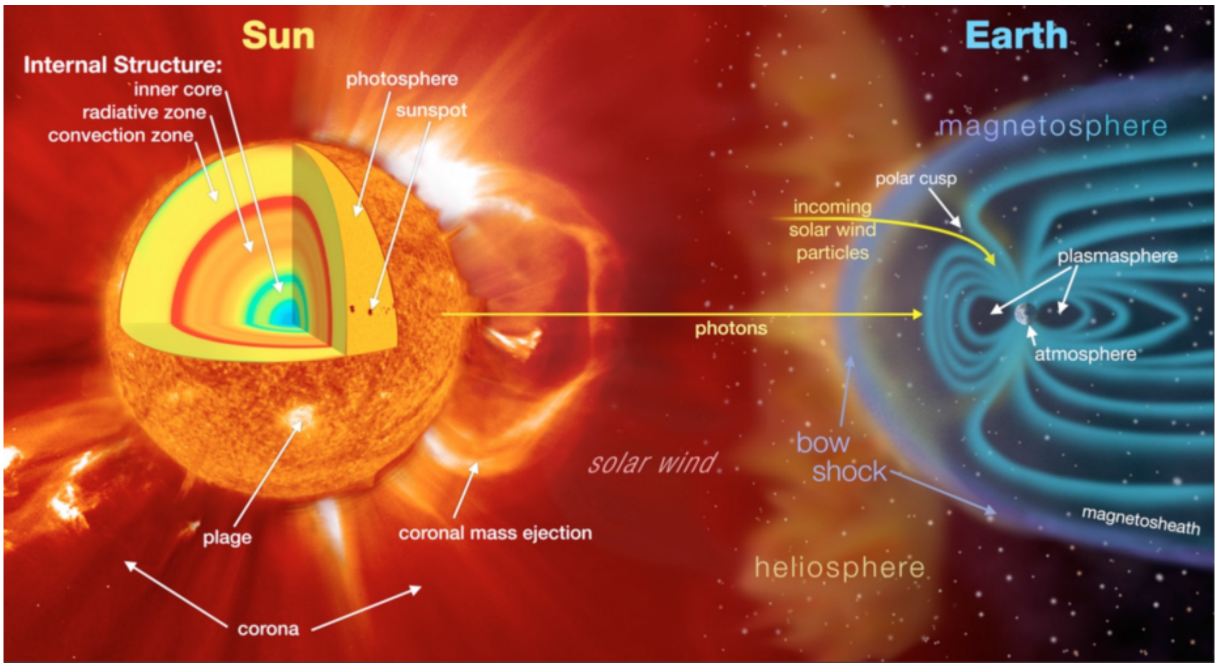


Figure 8.5: Solar wind consisting of charged particles and the sun's magnetic field bombard Earth's magnetosphere. Credit: NASA Goddard Space Flight Center

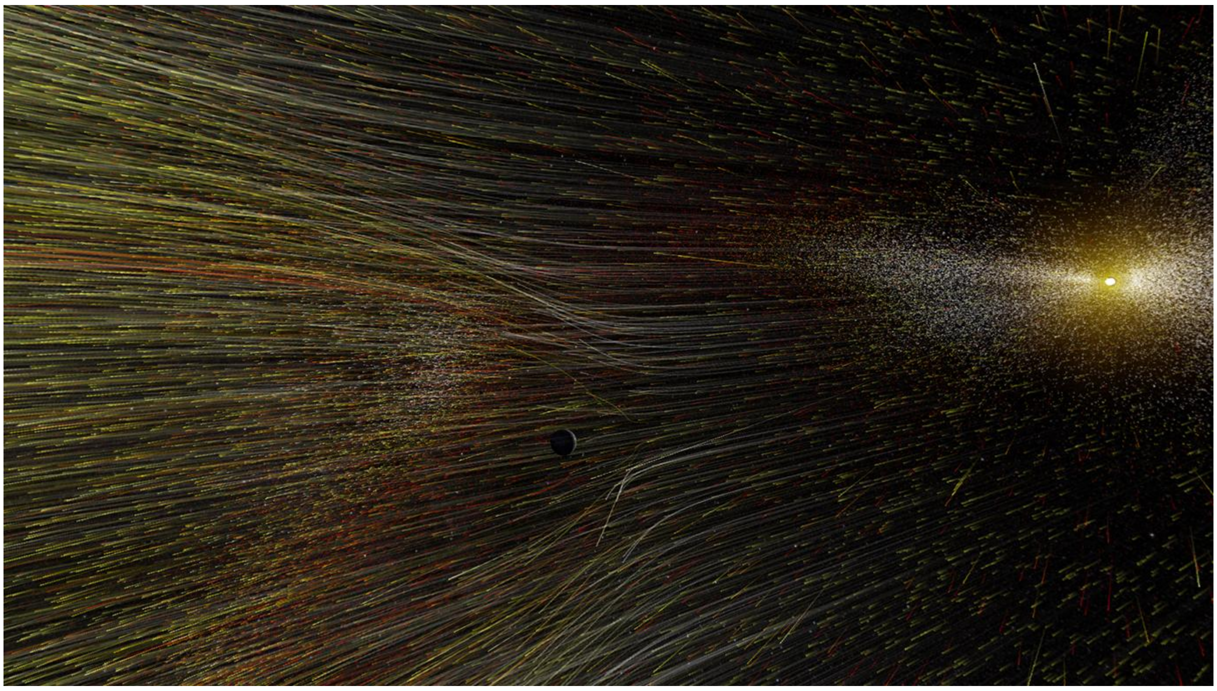
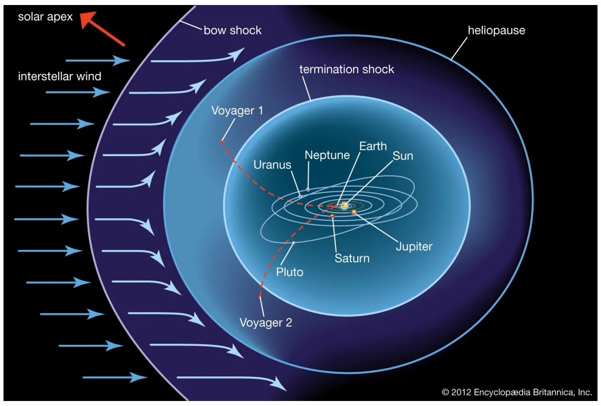
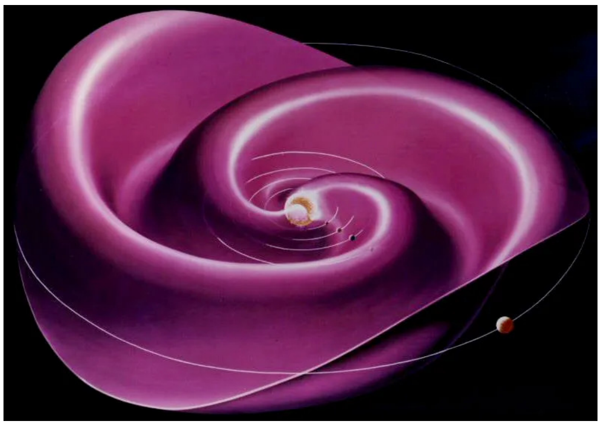


Figure 8.6: Solar wind is continually released from the sun's outermost atmosphere. This artist's illustration shows solar wind streaming out from the sun. It, also, depicts that solar wind particles coming towards Earth. Credit: NASA

The solar wind is composed of protons and electrons and carries the Sun's magnetic field with it. Different regions on the Sun produce solar wind of varying speeds and densities. Coronal holes, which are large, persistent patches in the Sun's corona, produce high-speed solar wind. The solar wind travels at average speeds of 1.4 million kilometers



(a) Illustration of the heliosphere. Credit: Encyclopedia Britannica, Inc.



(b) The heliospheric current sheet. Credit: Werner Heil/NASA.

Figure 8.7: (a) Illustration of the heliosphere. The solar wind first encounters the interstellar medium at the bow shock. At the heliopause the outward pressure of the solar wind balances the pressure of the incoming interstellar medium. Credit: Encyclopædia Britannica, Inc. (b) The heliospheric current sheet. Its shape results from the influence of the Sun's rotating magnetic field on the plasma in the interplanetary medium. Credit: Werner Heil/NASA.

per hour. The Sun's north and south poles have prominent coronal holes, resulting in fast solar wind at high latitudes. In the equatorial plane where the planets orbit, the solar wind tends to be slower, forming the equatorial current sheet.

The effects of our windy star, the Sun, are felt throughout the solar system. As Nicky Fox, the division director for heliophysics at NASA Headquarters, stated, "if the sun sneezes, Earth catches a cold" due to the impact of the solar wind. The solar wind plays a crucial role in space weather, with high-speed solar winds causing geomagnetic storms and slow-speed solar winds resulting in calmer space weather conditions [90].

One notable effect of the solar wind on Earth is the stunning aurora displays seen around the polar regions. Known as the northern lights (aurora borealis) in the Northern Hemisphere and the southern lights (aurora australis) in the Southern Hemisphere, these displays can be expanded closer to the equator during geomagnetic storms triggered by high-speed solar wind. However, geomagnetic storms can also have damaging consequences. They can disrupt satellite operations and electricity networks, posing a threat to astronauts in space. During these storms, astronauts on the International Space Station seek shelter, spacewalks are paused, and sensitive satellites are powered down until the radiation storm subsides. SpaceX experienced firsthand the impact of space weather when a geomagnetic storm destroyed up to 40 Starlink satellites, worth over $50 million, in February 2022 [90]. Geomagnetic storms also lead to changes in the Earth's atmosphere. The storms energize the atmosphere, causing it to heat up and expand upward, resulting in a denser thermosphere. This denser thermosphere generates more drag, which can affect satellite operations. For instance, the increased drag from a geomagnetic storm caused the Starlink satellites to fall back to Earth and burn up in the atmosphere.

Given the costly consequences of solar weather, enhancing our understanding, monitoring, and prediction of such events is crucial. Scientists study the solar wind to gain insights into the space weather environment and improve space weather forecasts. Heliophysics missions, such as the Parker Solar Probe, the Solar and Heliospheric Observatory (SOHO), the Solar Terrestrial Relations Observatory (STEREO), and ESA's Solar Orbiter, are dedicated to studying the Sun and its influence on the solar system. Together, these missions form the Heliophysics System Observatory (HSO), aiming to comprehend various aspects, from the formation of planetary atmospheres to the impact of space weather on astronauts and technology near Earth.

The solar wind also releases charged particles in events called coronal mass ejections (CMEs), which can trigger geomagnetic storms and are associated with aurora displays. Moreover, the solar wind extends far beyond the orbit of Pluto, forming a large protective region known as the heliosphere, which shields the solar system from harmful cosmic rays. Variations in the properties of the solar wind occur throughout the Sun's 11-year cycle of activity, which directly affects space weather conditions on Earth.

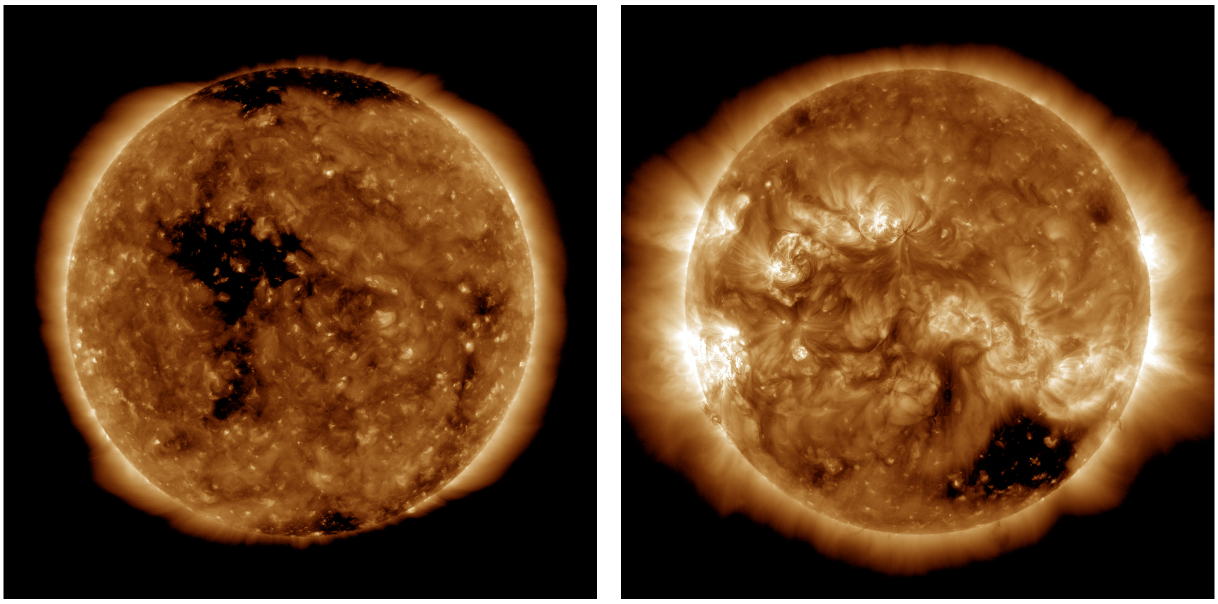
8.3 Coronal holes

Coronal holes appear as dark regions in the solar corona, characterized by cooler and less dense plasma compared to their surroundings. These open regions are associated with unipolar magnetic fields and allow the solar wind to escape more easily into space, resulting in streams of relatively fast solar wind [91]. Coronal holes can develop at any time and location on the Sun, but they are more common and persistent during the years around the solar minimum. These holes can last through several solar rotations and are most prevalent at the solar poles. However, they can also extend to lower solar latitudes or develop as isolated structures separate from the polar holes [92].

The interaction of the high-speed solar wind from coronal holes with the slower ambient solar wind leads to the formation of a compression region called a co-rotating interaction region (CIR). The CIR can result in particle density enhancement and increases in interplanetary magnetic field (IMF) strength before the arrival of the coronal hole's high-speed stream (CH HSS). As the CH HSS reaches Earth, solar wind speed and temperature increase, while particle density decreases. The IMF strength generally weakens as the CH HSS flow progresses [91]. Coronal holes located near the solar equator are most likely to result in CIR passages and higher solar wind speeds at Earth. Strong CIRs and faster CH HSS can cause geomagnetic storming, leading to periods of heightened geomagnetic activity and enhanced auroral displays. The size and location of the coronal hole on the solar disk determine the level of auroral activity, with larger holes typically associated with faster solar wind. However, coronal holes usually have minimal effects on aurora watchers at middle latitudes, occasionally causing geomagnetic storm conditions [91]. These regions of open magnetic field lines, known as coronal holes, play a significant role in the solar wind's formation. They can persist for weeks or months, changing in shape and size over time. Coronal holes can develop independently or as extensions of the polar coronal holes, which are more stable during solar minimum. The extensions towards lower latitudes can disconnect and become isolated structures themselves.

Coronal holes are observed in extreme ultraviolet and X-ray imagery, appearing as large dark regions in the solar atmosphere. They are characterized by unipolar magnetic fields that extend far into the solar system. These regions have lower plasma density, facilitating the escape of charged plasma and contributing to the fast component of the solar wind, reaching speeds of approximately 700 kilometers per second [91].

Understanding coronal holes is crucial for space weather forecasting. Their influence



(a) Coronal holes at solar minimum. Credit: NASA's Scientific Visualization Studio/SDO.

(b) Coronal holes at solar maximum. Credit: NASA's Scientific Visualization Studio/SDO.

Figure 8.8: (a) A sample of solar coronal holes around the time of the minimum of sunspot activity (October 2019). Note the coronal holes in the solar polar regions (near the top and bottom of the solar disk) and the large coronal hole across the Sun's equator. Credit: NASA's Scientific Visualization Studio/SDO. (b) A sample of solar coronal holes around the time of the maximum of sunspot activity (April 2014). Note the polar regions are devoid of coronal holes, but a large hole appears in the Southern Hemisphere. Credit: NASA's Scientific Visualization Studio/SDO.

on the solar wind and its interaction with Earth's magnetosphere can lead to geomagnetic storms and associated aurorae. By closely analyzing and monitoring coronal holes, forecasters can provide valuable information about the expected levels of geomagnetic response and overall space weather conditions.

In summary, coronal holes are temporary regions of cooler, less dense plasma in the solar corona, where the Sun's magnetic field extends as open field lines into interplanetary space. These regions allow solar wind to escape at an accelerated rate, resulting in decreased plasma temperature and density, as well as increased solar wind speed. Streams of high-speed solar wind from coronal holes can cause significant displays of aurorae and geomagnetic storms, particularly during periods of solar minimum when coronal mass ejections are less frequent [91].

8.4 Coronal Mass Ejections

Coronal mass ejections (CMEs) are massive bubbles of coronal plasma ejected from the Sun, characterized by intense magnetic field lines. They often resemble twisted ropes known as "flux ropes". While CMEs can occur simultaneously with solar flares, they can also happen spontaneously. The frequency of CMEs varies throughout the 11-year solar cycle, with approximately one per week during solar minimum and an average of two to three per day near solar maximum [93].

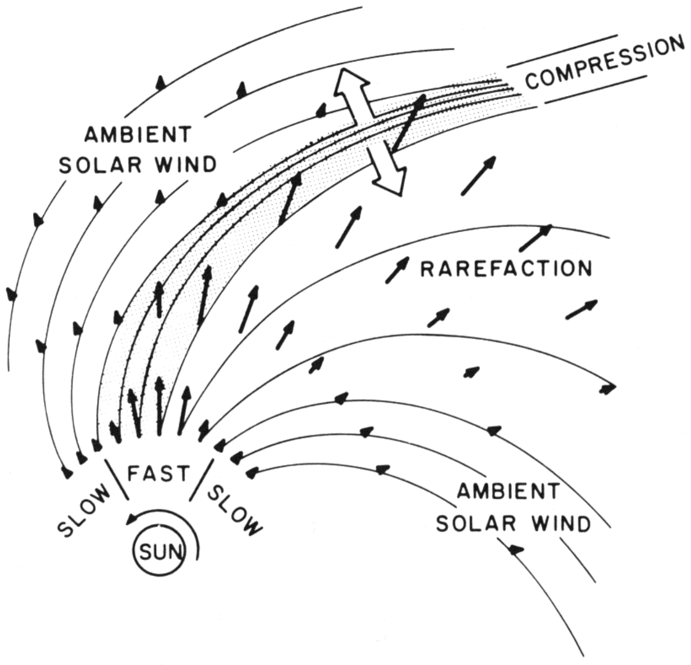
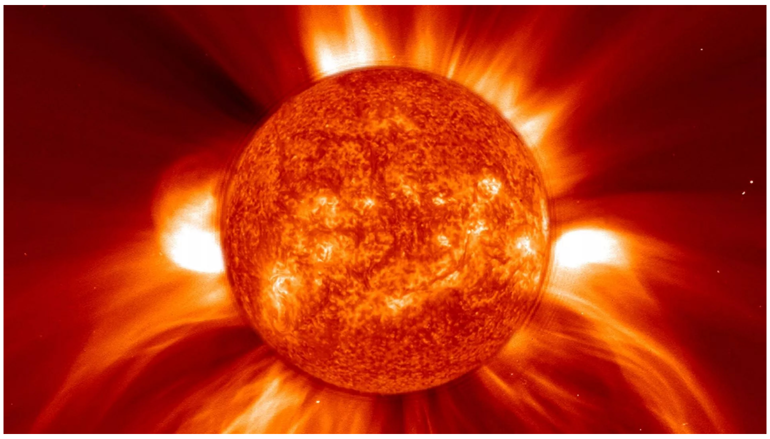


Figure 8.9: Co-rotating interaction region (CIR). Geometry of the interaction between fast solar wind and ambient solar wind. Credit: SpaceWeatherLive

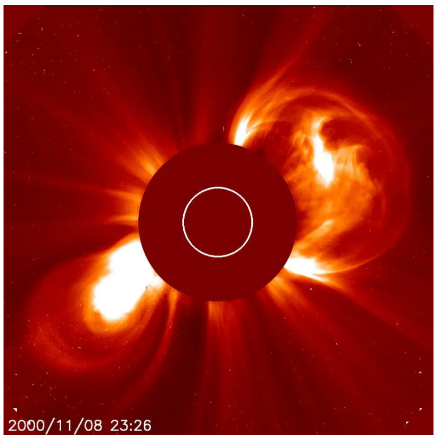
CMEs disrupt the solar wind flow and can cause disturbances that impact systems on Earth and in near-Earth space. The magnetic fields of CMEs merge with the interplanetary magnetic field (IMF) and geomagnetic field lines, resulting in increased energy transfer from the solar wind to the magnetosphere. As a consequence, CMEs play a crucial role in driving geomagnetic storms and substorms, leading to the mesmerizing auroral lights observed at high latitudes [93]. This large-scale plasma and magnetic field ejections expand in size as they propagate away from the Sun. Faster CMEs can travel at speeds ranging from slower than 250 kilometers per second (km/s) to nearly 3000 km/s, reaching Earth in as little as 15 to 18 hours. Slower CMEs may take several days to arrive, but they can still have significant impacts. The size of larger CMEs can encompass a quarter of the space between Earth and the Sun when they reach our planet [94].

CMEs result from the realignment of highly twisted magnetic field structures known as flux ropes in the lower corona of the Sun. This reconfiguration, termed magnetic reconnection, can trigger a sudden release of electromagnetic energy in the form of a solar flare. CMEs predominantly originate from active regions associated with sunspot groups but can also occur from locations where relatively cool and denser plasma is trapped by magnetic flux, such as filaments and prominences. Faster CMEs can generate shock waves, accelerating charged particles ahead of them and intensifying radiation storm potential [94].

Important parameters for CME analysis include size, speed, and direction, which are determined through coronagraph imagery obtained from orbital satellites. Instruments like the Large Angle and Spectrometric Coronagraph (LASCO) on the NASA Solar and Heliospheric Observatory (SOHO) and the coronagraph on the NASA STEREO-A spacecraft aid in analyzing and categorizing CMEs. The Deep Space Climate Observatory (DSCOVR) satellite, positioned at the L1 orbital area, provides advanced warning of



(a) A coronal mass ejection (CME) captured by NASA and ESA's Solar and Heliospheric Observatory (SOHO). Credit: NASA/GSFC/SOHO/ESA.



(b) The eruption of two CMEs on Jan. 8, 2000 recorded from NASA's LASCO coronagraph. Credit: NASA/SOHO/LASCO

Figure 8.10: Coronal Mass Ejections.

CME-associated interplanetary shocks, aiding in monitoring CME arrival and potential geomagnetic storm initiation [94].

When a CME collides with Earth's magnetosphere, the resulting disturbance can cause geomagnetic storms, aurorae, and, in rare cases, damage to electrical power grids. The occurrence of CMEs near solar maximum is more frequent, with approximately three CMEs daily, while near solar minimum, there is approximately one CME every five days [95]. CMEs, with their explosive plasma outbursts, carry a substantial mass of material, traveling at high speeds of hundreds of kilometers per second. They contain particle radiation and powerful magnetic fields stronger than the normal solar wind. These eruptions, originating from magnetically disturbed regions in the Sun's upper atmosphere, occur more frequently during the solar maximum phase of the sunspot cycle. While closely associated with solar flares, the exact relationship between CMEs and solar flares remains uncertain. As CMEs traverse the solar system, some are directed towards Earth, and if they interact with Earth's magnetosphere, they can generate radiation storms and awe-inspiring auroras [96].

In summary, coronal mass ejections are enormous plasma bubbles with intense magnetic field lines that erupt from the Sun, exhibiting a variety of characteristics and behaviors throughout the solar cycle. Their impacts on the solar wind, magnetosphere, and Earth's systems make them significant drivers of geomagnetic storms, substorms, and captivating auroras.

8.5 Solar flares

A solar flare is an intense, localized eruption of electromagnetic radiation in the Sun's atmosphere. Flares occur in active regions and are often, but not always, accompanied by coronal mass ejections, solar particle events, and other solar phenomena. The occurrence of solar flares varies with the 11-year solar cycle.

Solar flares are thought to occur when stored magnetic energy in the Sun's atmosphere accelerates charged particles in the surrounding plasma. This results in the emission of electromagnetic radiation across the electromagnetic spectrum.

Solar flares are large explosions of high-energy electromagnetic radiation from the Sun and are absorbed by the daylight side of Earth's upper atmosphere, particularly the ionosphere, and do not reach the surface. This absorption can temporarily increase the ionization of the ionosphere, potentially interfering with short-wave radio communication. These intense bursts of radiation can be visible as bright flashes and last from minutes to hours. They occur when energy stored in tangled magnetic fields is suddenly released in the Sun's atmosphere. Flares are often accompanied by CMEs, which are large releases of plasma and magnetic field.

Solar flares are classified based on their brightness in X-ray wavelengths, with X-class flares being the most powerful, followed by M-, C-, and B-class flares. A-class flares are the smallest and have minimal impact on Earth. The number of sunspots, dark and cooler regions on the Sun's surface with strong magnetic fields, can indicate the likelihood of a solar flare eruption.

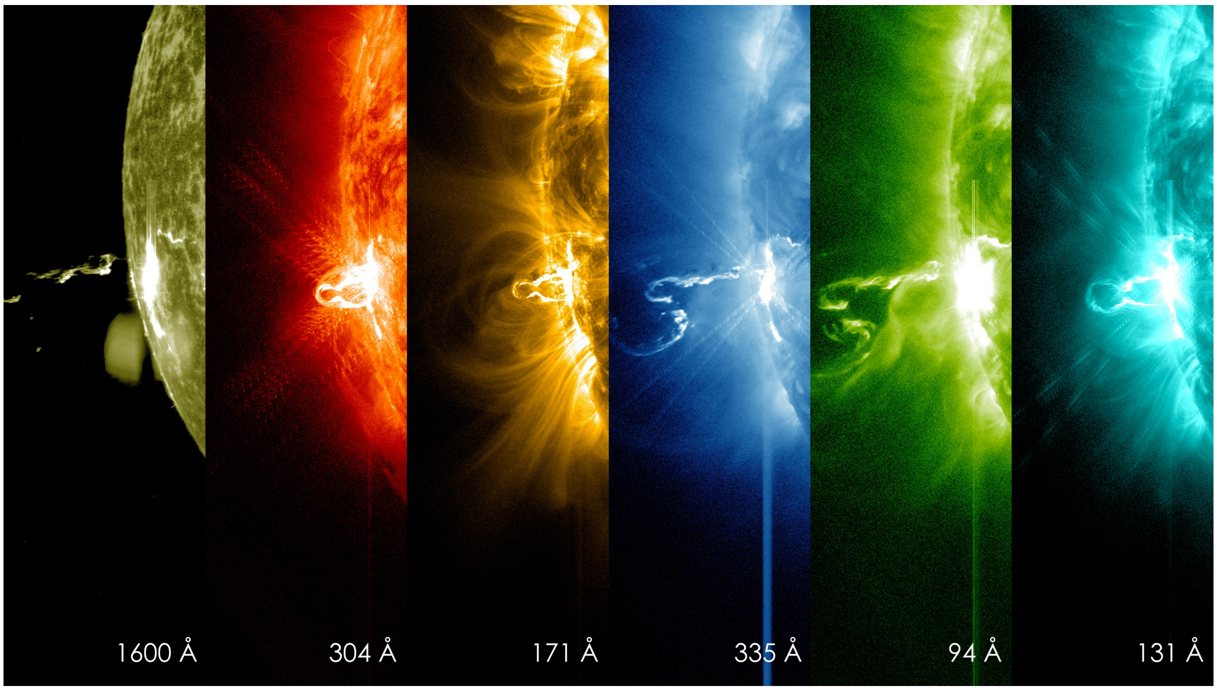


Figure 8.11: On Feb. 24, 2014, the sun emitted a significant solar flare, peaking at 7:49 p.m. EST. NASA's Solar Dynamics Observatory (SDO), which keeps a constant watch on the sun, captured images of the event. These SDO images from 7:25 p.m. EST on Feb. 24 show the first moments of this X-class flare in different wavelengths of light – seen as the bright spot that appears on the left limb of the sun. Hot solar material can be seen hovering above the active region in the sun's atmosphere, the corona. Credit: NASA/SDO

During strong solar flares, charged electrons in the upper atmosphere can temporarily disrupt radio waves, causing radio blackouts. The severity of these blackouts depends on the strength of the solar flare and is classified on the NOAA Solar Radiation Storm Scale. The effects of solar flares on Earth's technologies and communications can be mitigated by monitoring and issuing warnings from organizations like NASA, NOAA, and the U.S. Air Force Weather Agency. Solar flares can significantly affect spacecraft, satellites, and communication systems. However, they don't possess enough energy to cause lasting damage to Earth itself. It is important to note that solar flares do not pose a threat of destroying Earth. While they can disrupt the technological world, appropriate measures can be taken to protect ourselves.

Monitoring space weather and understanding the effects of solar activity require studying both solar flares and CMEs. By closely monitoring the Sun's activity, organizations can provide warnings to vulnerable technology sectors. Predicting space weather is an ongoing effort that has improved over the years. Solar flares and their associated phenomena are fascinating, but not something to worry about. They are natural events that occur as part of the Sun's dynamic behavior. By studying and understanding them, we can better protect our technology and prepare for their potential impacts.

8.6 Sunspots

The content of this subsection has been extracted from [97].

8.6.1 General characteristics

Sunspots are planet-size, dark regions on the surface of the Sun, boasting strong magnetic fields. These magnetic fields can give rise to eruptive disturbances like solar flares and coronal mass ejections (CMEs).

The reason sunspots appear darker is due to their relatively lower temperature compared to the surrounding areas. The central region of a sunspot, known as the umbra, has a temperature of approximately $3,500$ degrees Celsius, while the adjacent photosphere registers around $5,500$ degrees Celsius, as stated by the National Weather Service (NWS).

The frequency and intensity of visible sunspots are indicative of the solar activity level within the 11-year solar cycle, which is influenced by the Sun's magnetic field. Sunspots provide a valuable glimpse into the complex magnetic nature of the Sun's interior and have captivated solar observers for centuries.

8.6.2 Formation

According to the European Solar Telescope, sunspots are formed when concentrations of magnetic fields from the Sun's deep interior rise to the surface. These sunspots consist of a central dark region called the umbra and a surrounding region known as the penumbra. Although the exact process of sunspot formation is not fully understood, scientists generally accept the theory proposed by astronomer Horace Babcock in 1961, which suggests that sunspots are a result of the Sun's magnetic field. Sunspots are, on average, about the same size as Earth, though they can vary from hundreds to tens of thousands of miles across, according to Cool Cosmos.

To visualize this, imagine the Sun's magnetic field as loops of rubber bands, with one end attached to the north pole and the other to the South Pole. As the Sun rotates, a phenomenon known as "differential rotation" occurs, with the equator rotating faster than the poles, as explained by the Royal Museums Greenwich. This differential rotation

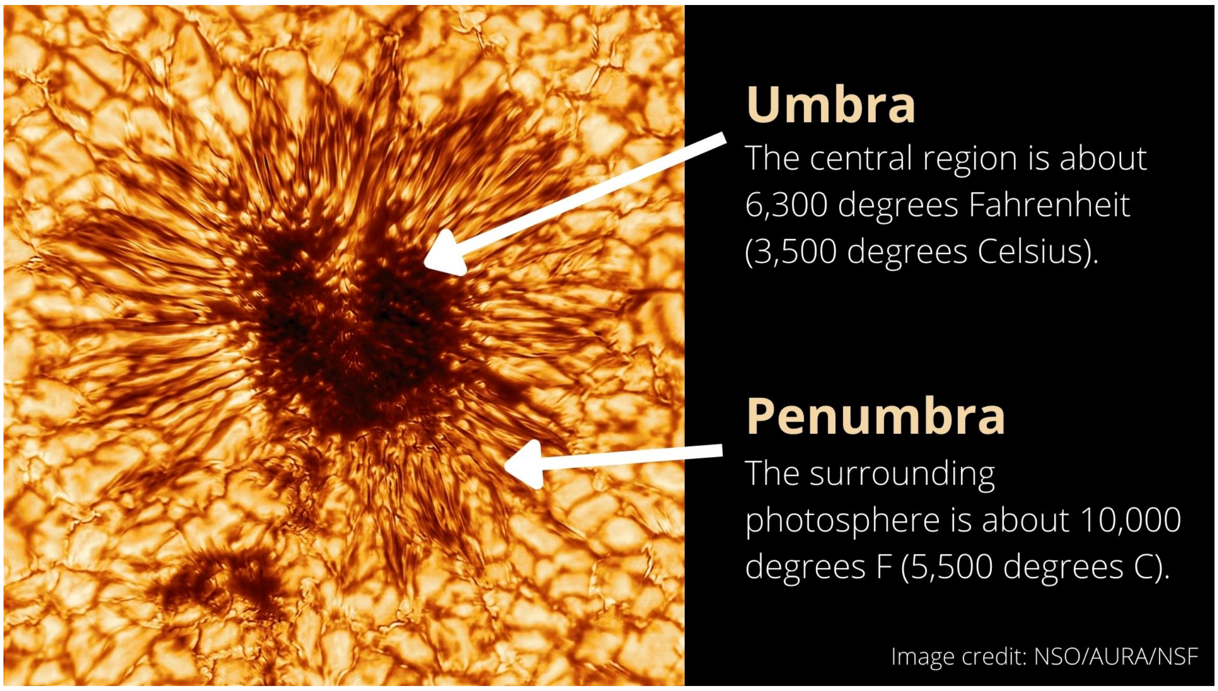

Umbra
The central region is about 6,300 degrees Fahrenheit (3,500 degrees Celsius).
Penumbra
The surrounding photosphere is about 10,000 degrees F (5,500 degrees C).
Image credit: NSO/AURA/NSF

Figure 8.12: Sunspots consist of a central darker region, known as the umbra, and a surrounding region, known as the penumbra. Credit: NSO/AURA/NSF. Infographic made by Space.com

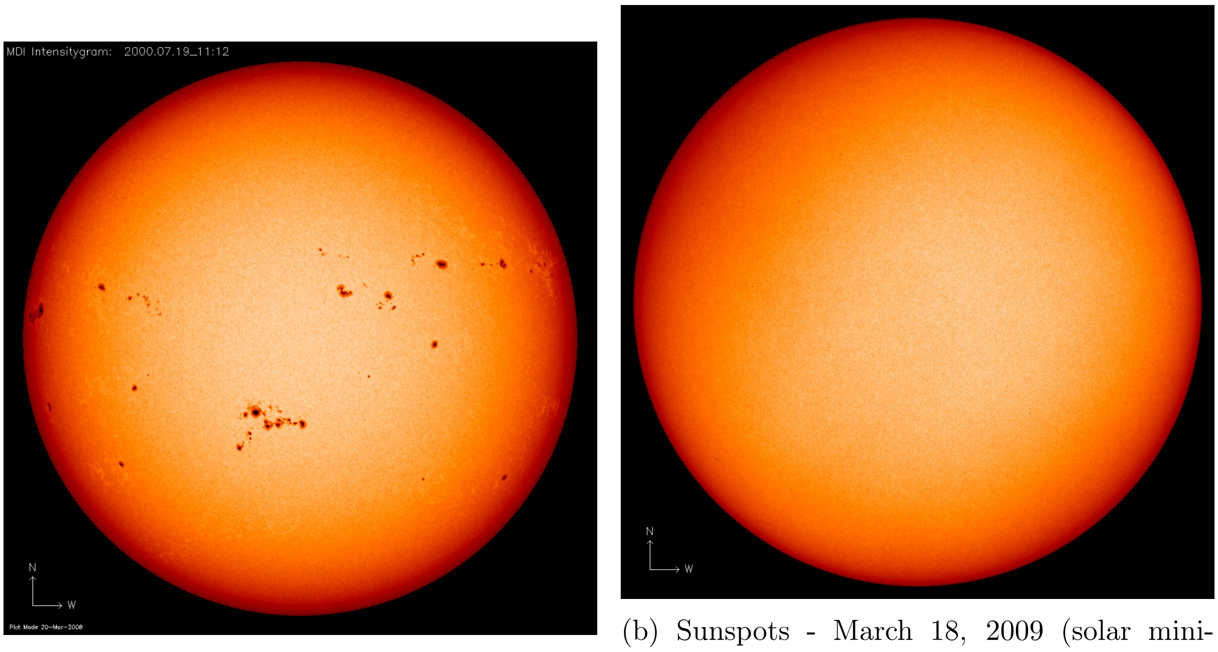
causes the magnetic loops to become increasingly twisted and complex. Eventually, these magnetic fields reach a point where they snap, rise, and break through the surface of the Sun. This disruption in the magnetic field forms small pores, which can merge and grow into larger structures called proto-spots, eventually developing into fully-fledged sunspots. A collection of sunspots is referred to as an active region.

The magnetic field in active sunspot regions can be up to $2,500$ times stronger than Earth's magnetic field, according to the National Weather Service (NWS). This strong magnetic field restricts the inflow of hot gas from the Sun's interior, causing sunspots to be cooler and appear darker in comparison to their surroundings. The University Corporation for Atmospheric Research (UCAR) notes that if you were to cut out a standard sunspot from the Sun and place it in the night sky, it would shine as brightly as a full moon.

8.6.3 Sunspots and the solar cycle

Sunspots have a lifespan that typically spans from a few days to several weeks, and occasionally they can persist for months before eventually dissipating. The total number of sunspots fluctuates throughout the 11-year solar cycle, commonly referred to as the sunspot cycle. According to the National Oceanic and Atmospheric Administration's (NOAA) Space Weather Prediction Center (SWPC), the peak of sunspot activity coincides with the solar maximum, whereas the solar minimum is characterized by a reduction in sunspot occurrence.

The positions of sunspots also change as the solar cycle progresses. During the solar maximum, a larger number of sunspots are observed along mid-latitudes, approximately 30 degrees north and south. Subsequently, as the cycle advances, sunspots gradually migrate



(a) Sunspots - July 19, 2000 (solar maximum). (b) Sunspots - March 18, 2009 (solar minimum).

Figure 8.13: Visible light images show the Sun at solar maximum in July 2000 and at solar minimum in March 2018. Sunspots freckle the Sun during solar maximum. The dark spots are associated with solar activity. Credit: NASA Earth Observatory/GSFC/SOHO.

towards the equator, resulting in fewer sunspots during the solar minimum. There are instances during the solar minimum when no sunspots are visible. It is worth noting that although the 11-year solar cycle is relatively consistent, an exceptional period known as the "Maunder minimum" occurred between 1645 and 1715. During this time, there was a substantial dearth of sunspot activity, with fewer than 50 sunspots recorded between 1672 and 1699, according to Physics World. In comparison, a typical solar minimum usually features 12 to over 100 sunspots per year. The Maunder minimum was named after British astronomer Edward Walter Maunder, who, along with his wife Annie, identified the prolonged period of minimal sunspot activity from historical records in 1890, as reported by The Times.

8.6.4 Discovery of the sunspots

There is some debate about who discovered sunspots. According to the Chandra X-ray Center, the earliest records of solar activity are from Chinese astronomers around 800 B.C. Chinese and Korean astronomers frequently observed sunspots, according to the Chandra X-ray Center. However, there are no known early illustrations of such observations.

The earliest known drawings of solar activity appeared many years later, in 1128, in John of Worcester's chronicle. “In the third year of Lothar, emperor of the Romans, in the twenty-eighth year of King Henry of the English ... on Saturday, 8 December, there appeared from the morning right up to the evening two black spheres against the sun”, Worcester wrote. Just five days after Worcester described a large sunspot group, Korean astronomers reported that they'd observed a red vapor that “soared and filled the sky”. This description suggests the presence of the aurora borealis, or northern lights, at

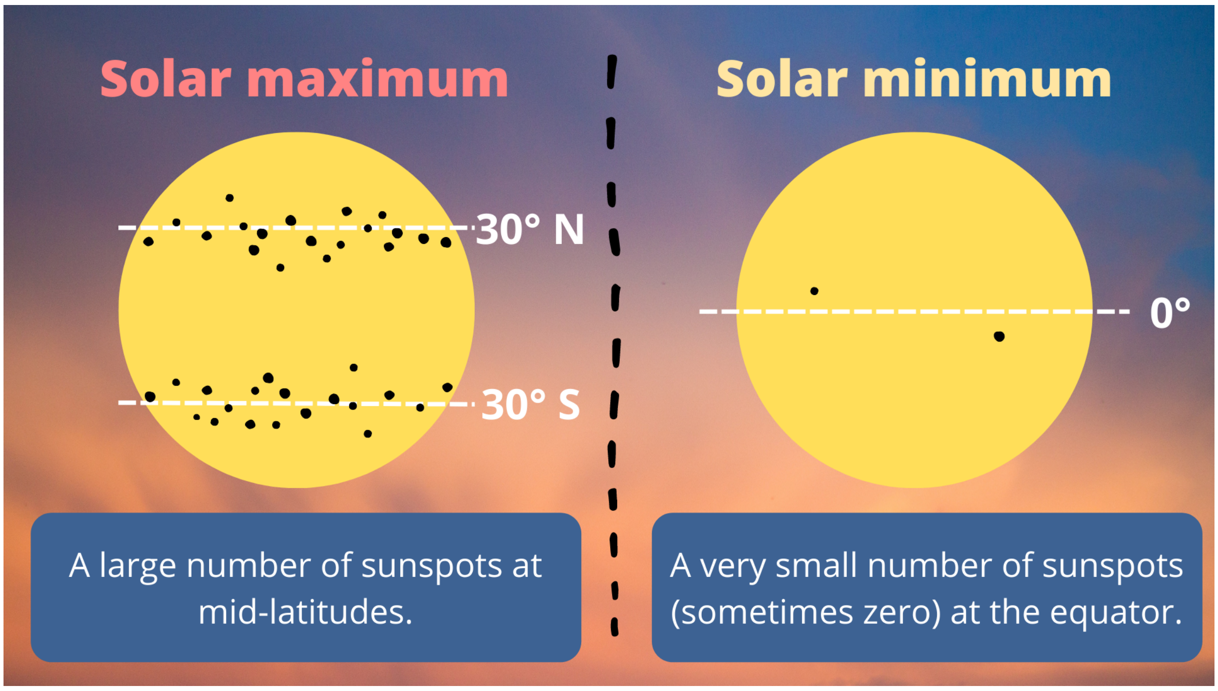

Solar maximum
30° N
30° S
A large number of sunspots at mid-latitudes.
Solar minimum
0°
A very small number of sunspots (sometimes zero) at the equator.

Figure 8.14: During solar maximum a large number of sunspots are visible at mid-latitudes and during solar minimum a very small number (sometimes zero) of sunspots are visible at the equator. Credit: Future.

relatively low latitudes.

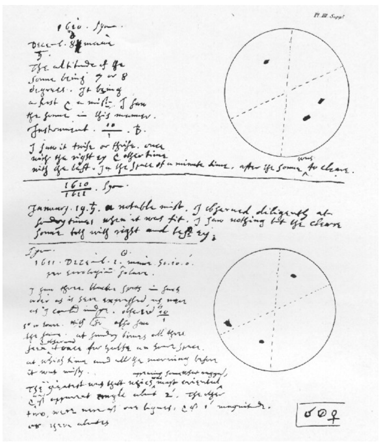
In 1610, aided by a telescope, English astronomer Thomas Harriot detailed his solar observations according to NASA with detailed notes and sketches. His drawings are the earliest known pictorial record of sunspots. A year later, David and Johannes Fabricius (father and son) independently discovered sunspots. A couple of months after that, Johannes Fabricius became the first person in the West to publish anything on the subject of sunspots, in a pamphlet titled "On the Spots Observed in the Sun and their Apparent Rotation with the Sun".

According to NASA, there were two other independent sunspot discoveries at the same time in 1611. Galileo Galilei and Jesuit Christoph Scheiner competed over who deserved the credit for discovering sunspots. Unbeknownst to the quarreling astronomers, sunspots had already been observed and recorded hundreds of years earlier, so their lifelong feud was futile.

8.6.5 Observing sunspots

Sunspots have been observed for hundreds of years and continue to be the main focus for scientists who want to learn more about the solar cycle and assess the risk of space weather, such as solar flares and CMEs.

Our current picture of solar activity would not be as clear without the work of Japanese astronomer Hisako Koyama. Between 1947 and 1996, Koyama sketched sunspots from the roof of the National Museum of Nature and Science in Tokyo, using a 20-centimeter refracting telescope. For over 40 years, Koyama made more than 10,000 sunspot observations that have shaped solar science and our understanding of space weather, according


Pl. III. Suppl.

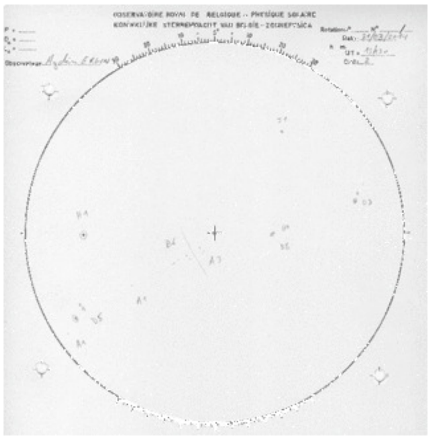
1610. Syon.
Decemb. 8 mane
♄.
The altitude of the
sonne being 7 or 8
degrees. It being
a frost & a mist. I saw
the sonne in this manner.
Instrument. $\frac{10}{1}$. B.
I saw it twise or thrise. once
with the right ey & other time
with the left. In the space of a minute time. after the sonne was to cleare.

$\frac{1610}{1611}$. Syon.
January. 19. ♄. a notable mist. I observed diligently at
sundry times when it was fit. I saw nothing but the cleare
sonne both with right and left ey.

Syon.
1611. Decemb. 1. mane [illegible]
per [illegible] solare.
I saw three blacke spots in such
order as is here expressed as neare
as I could iudge. observed $\frac{10}{1}$
[illegible] also saw
the same. at sundry times all three
I observed seen at once for halfe an howre space.
at which time and all the morning before
it was misty.
The greatest was that which was most oriental appearing somewhat ragged
& of apparent angle about 2'. the other
two, were neare of one bignes, & of 1' magnitude.
or there abouts

[illegible]

(a)



(b)

Figure 8.15: (a) A page from Thomas Harriot's notebook. Credit: Thomas Harriot. (b) Sunspots can be monitored with daily hand-drawn sketches. This image shows sunspot drawings from the World Data Center for the Sunspot Index and Long-term Solar Observations (SILSO) at the Royal Observatory of Belgium. Credit: SILSO/Royal Observatory of Belgium.

to a commentary about her work published in the journal Space Weather.

Nowadays, scientists at NOAA's Space Weather Prediction Center analyze sunspot regions daily to access their threats. They monitor and record changes in sunspot size, number and position to assess the likelihood of a solar flare and/or CME from an active region. The World Data Center for the Sunspot Index and Long-term Solar Observations at the Royal Observatory of Belgium also tracks sunspots and records the highs and lows of the solar cycle to evaluate solar activity and improve space weather forecasting. Scientists classify sunspot groups to assess which are more likely to incite a solar flare or CME. To do so, researchers at the Mount Wilson Observatory in California have come up with a set of classifications to assign to sunspot groups, according to SpaceWeatherLive.

Each day, sunspots are counted and receive both a magnetic classification and a spot classification. Another classification system is based on the Zürich/McIntosh system and is designed to classify sunspots to inform scientists about how long the sunspot will last, its complexity and size, SpaceWeatherLive says.

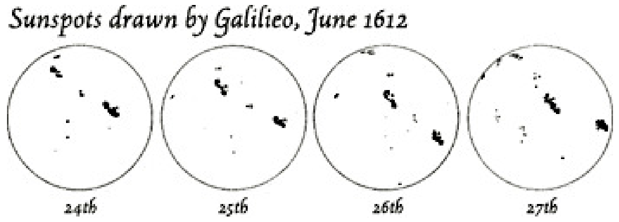


Figure 8.16: Some of Galileo's sketches of sunspots from the 17th century. Credit: NASA Earh Observatory.

8.6.6 Sunspots today

Solar cycle 25 is the current solar cycle, the 25th since 1755, when extensive recording of solar sunspot activity began. It began in December 2019 with a minimum smoothed sunspot number of 1.8. It is expected to continue until about 2030 [98].

The forecast that is shown in 8.17 comes from the Solar Cycle Prediction Panel representing NOAA, NASA and the International Space Environmental Services (ISES) which was convened in 2019. This amounts to the official forecast for the solar cycle 25. After an open solicitation, the Panel received nearly 50 distinct forecasts for Solar Cycle 25 from the scientific community. Prediction methods include a variety of physical models, precursor methods, statistical inference, machine learning, and other techniques. The prediction released by the panel is a synthesis of these community contributions [98].

The Prediction Panel predicted Cycle 25 to reach a maximum of 115 occurring in July, 2025. The error bars on this prediction mean the panel expects the cycle maximum could be between 105-125 with the peak occurring between November 2024 and March 2026 [98].

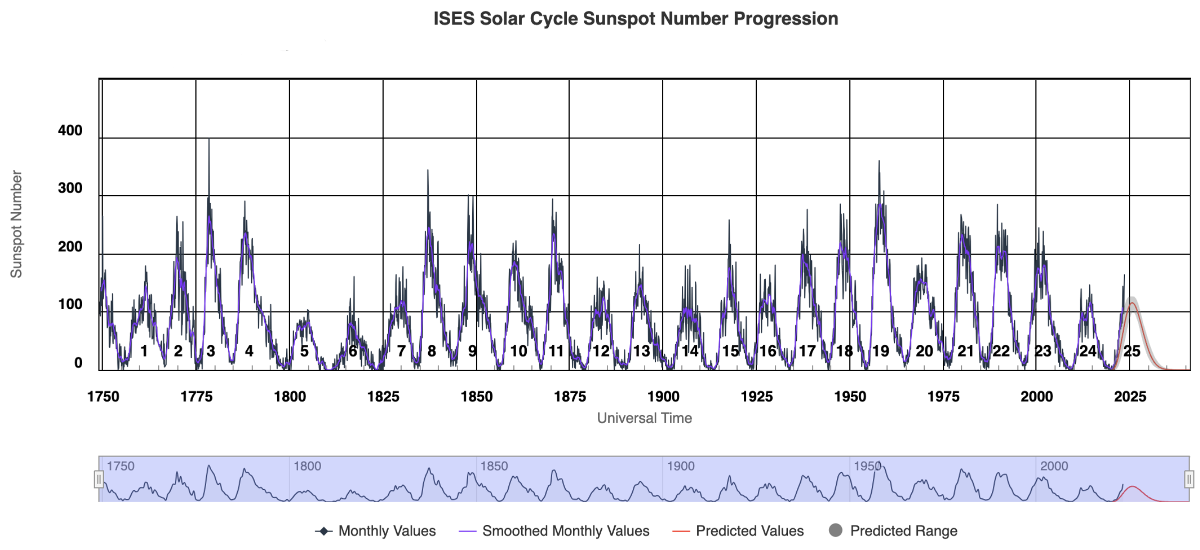


Figure 8.17: ISES Solar Cycle Sunspot Number Progression. Credit: NOAA/Space Weather Prediction Center (SWPC).

9 Exploratory Analysis of Sunspot Number Datasets

The sunspot number time-series exhibits a peculiar and rich character, while being one of the most important indicators of solar activity. Here, we will analyze the time-series of:

- the yearly mean total sunspot number, and
- the 13-month smoothed monthly total sunspot number,

in both, time and frequency domain. In that way, we will explore some of the key characteristics present in these time-series, whose knowledge is instrumental when someone wants to build forecasting models.

9.1 Sunspots time-series analysis in time domain

In this subsection, we analyze the sunspot time-series (yearly and monthly) in the time domain and inspect some statistical properties.

9.1.1 Autocorrelation and Partial Autocorrelation

The **Autocorrelation Function (ACF)** and the **Partial Autocorrelation Function (PACF)** are fundamental tools used in time-series analysis. These tools help us understand and quantify the temporal dependencies that exist in a time-series dataset. Below, we explain what these functions represent and make the corresponding plots for our sunspot time-series.

The ACF provides a measure of the correlation between observations of a time-series at two different points in time, as a function of the time difference (or lag) between these two points. It gives us information about the direct and indirect relationships between different lags of a time-series. By plotting the ACF, we can visually see the correlation of a time-series with its own lags. This helps us understand the degree to which a data point in a time series is influenced by its past data points and can provide insights into whether the time-series is stationary.

On the other hand, the PACF quantifies the correlation between observations at two points in time, while controlling for any influence from other time points. In other words, the PACF only measures the direct relationships between different lags of a time series, effectively removing the correlations already explained by intervening time points. It isolates the impact of each lag and provides a clearer picture of the relationship between a particular lag and the current observation.

Both ACF and PACF plots provide valuable insights that can inform the selection of an appropriate model for forecasting and generally help in time-series modeling by understanding the underlying patterns. In particular, the ACF and PACF are instrumental when determining the order of Autoregressive (AR) or Moving Average (MA) components in an ARIMA model for a time-series. The patterns observed in these plots can suggest the presence of trend, seasonality, and the suitable orders for AR, MA, or ARMA models.

In Figures 9.1 and 9.2, the **ACF** and **PACF** for the time-series of the **yearly mean total sunspot number** are shown. In the **ACF** plot, we observe a tail-off. This suggests that the data might be characterized by an Autoregressive (AR) process. However, after the lag of 38, the autocorrelation drops and stays within the 95% confidence interval of no autocorrelation (shown by the horizontal bounds around zero). This implies that lags beyond 38 are not significantly correlated with the present value. In **PACF** plot, there is also a tail-off in the correlations, indicative of a Moving Average (MA) process. However, note that lags after the 85 lag are not statistically significant at the 5% level. Thus, the partial autocorrelations beyond this point may not be meaningful. Furthermore, we could set lag 9 as the cut-off lag in the PACF, given that greater lags are slightly significant at the 5% level. Considering these plots, a MA model of order 9 may be a reasonable starting point for modeling this series, along with an AR model of order 38, or even a combined ARMA model.

In Figures 9.3 and 9.4, the **ACF** and **PACF** for the time-series of the **13-month smoothed monthly total sunspot number** are shown. We also see a progressive degradation in the **ACF** plot. The autocorrelation decreases and remains within the 95% confidence interval of no autocorrelation after the lag of 460. This suggests that lags above 460 have a negligible correlation to the present value. The correlations in the **PACF** plot likewise show a tail-off, and lags after lag 59 are not statistically significant at the 5% level. The partial autocorrelations that continue past this point might not be significant. Furthermore, considering that longer lags are marginally significant at the 5% level, we might choose lags 335 and 36 as the cut-off lags in the ACF and PACF, respectively. A MA model of order 36, an AR model of order 335, or even a mixed ARMA model may be a good place to start when modeling this series in light of these charts.

From ACF and PACF plots we can extract useful information about **seasonality** and **stationarity** in sunspot time-series. A clear sign of seasonality in such plots is a significant spike at the lag corresponding to the seasonal period, which is the length of one seasonal cycle, or periodic correlations with a period of that cycle. Here, in particular, we observe a periodicity of an 11 lag at the ACF plot of the yearly sunspots, while in the monthly sunspots it is a periodicity of a 130 lag. If a time-series is stationary, the ACF plot will show autocorrelation quickly decaying to zero. Non-stationary data, on the other hand, typically shows slow decay and have large and positive 1 lag, which is the case here. In the PACF plot, a stationary time-series will usually have one or two significant spikes, and the rest will mostly be within the insignificant range (inside the confidence band), which is not true for the sunspot number time-series. So, we can conclude that the sunspot data may not be stationary.

9.1.2 Trend analysis

Here, we conduct a trend analysis on the time-series of sunspot number. The analysis involves the following two main statistical methods:

- Linear regression (Ordinary Least Squares) (see Appendix A)
- Mann-Kendall trend test (and its variants) (see Appendix B)

The linear regression is performed to determine if a significant linear trend exists in the sunspot data, while the Mann-Kendall test checks for the existence of any increasing or decreasing trend. Additionally, we use the Theil-Sen estimator as a more robust check

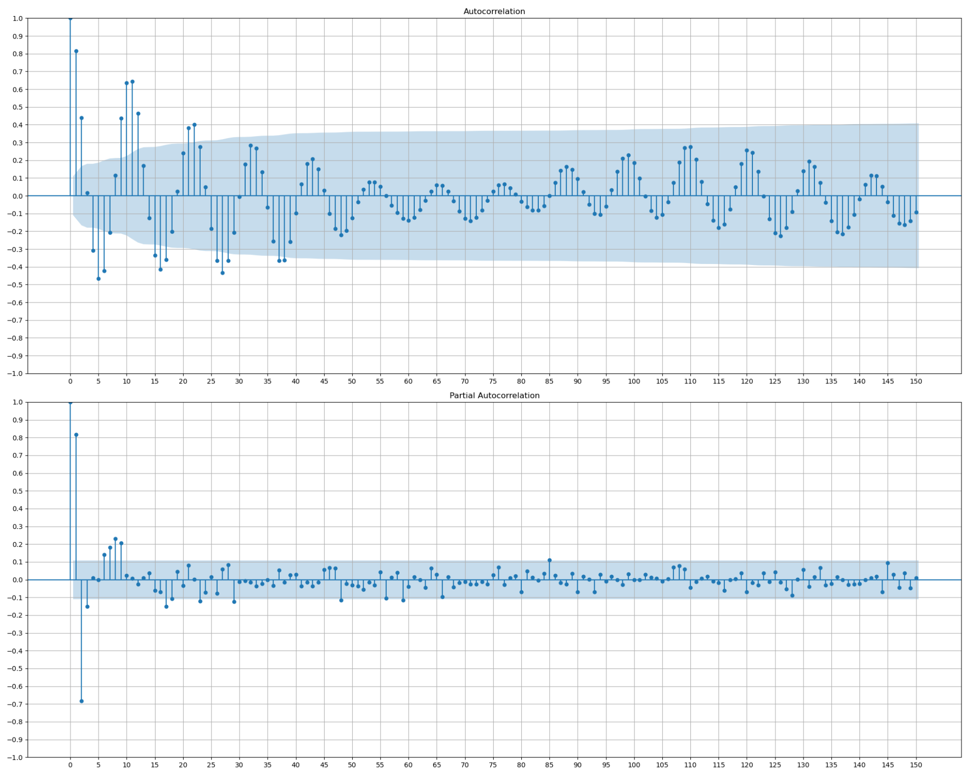


Figure 9.1: ACF and PACF for the time-series of the yearly mean total sunspot number.

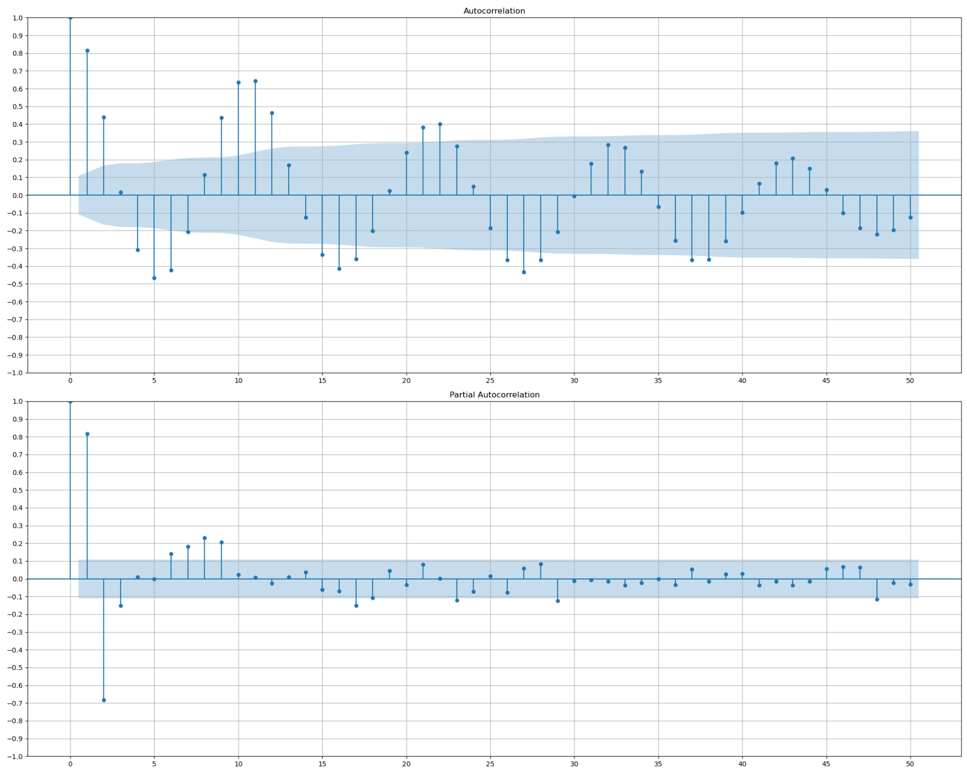


Figure 9.2: ACF and PACF for the time-series of the yearly mean total sunspot number (zoomed-in).

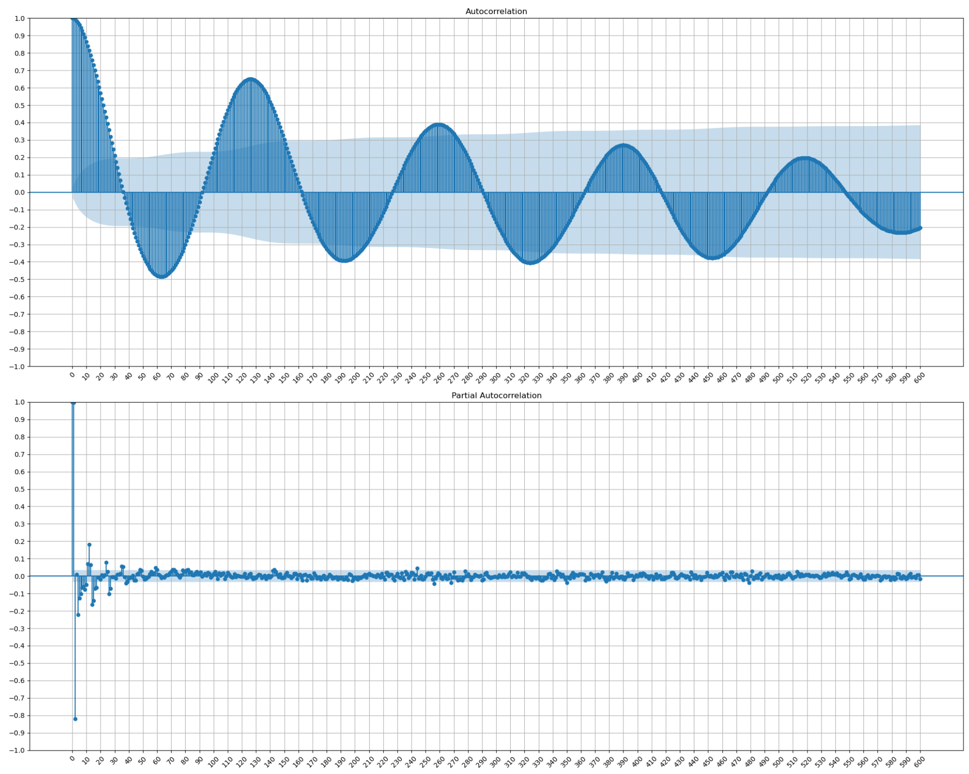


Figure 9.3: ACF and PACF for the time-series of the monthly sunspot number.

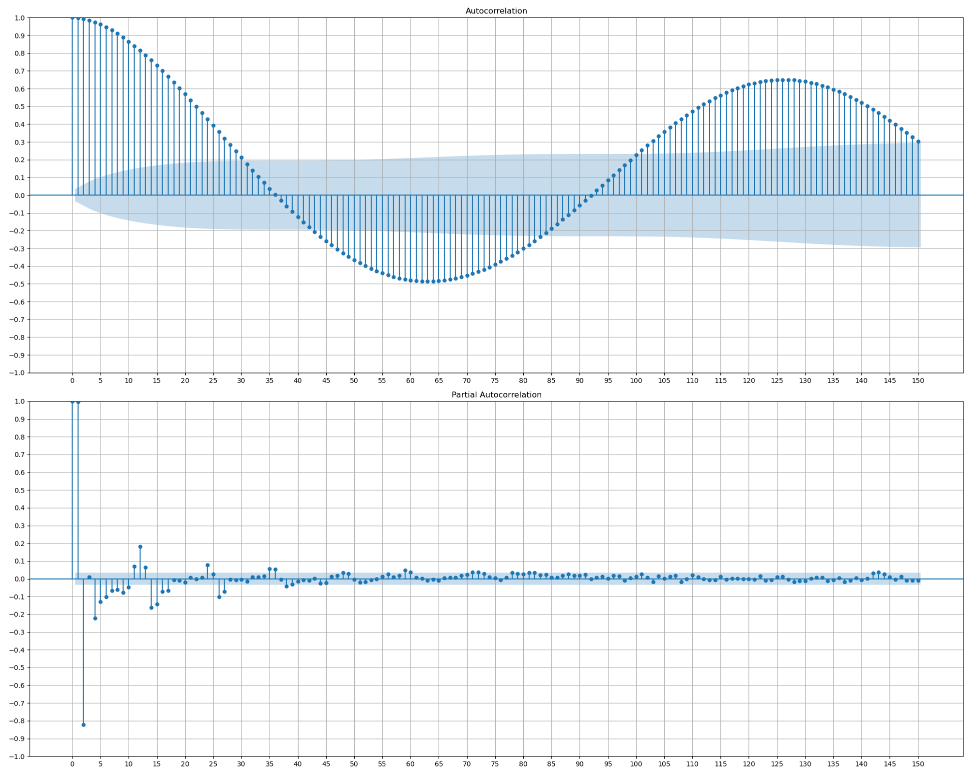


Figure 9.4: ACF and PACF for the time-series of the monthly sunspot number (zoomed-in).

for any trend line. The dependent variable in the model is the sunspot count, and the independent variable is time. We check for yearly and monthly sunspots as well.

In an attempt to break down the results of the **linear regression** in the time-series of the **yearly mean total sunspot number** (see Table 9.1), we comment the following:

- An **R-squared** of 0.011 indicates that only about 1.1% of the total variance in the data can be explained by the model. This is quite low, suggesting that this linear model does not explain much of the variability of the response data around its mean.
- A p-value of 0.0564 for the **F-statistic** is just above the common alpha level of 0.05, suggesting that the relationship between the predictor and outcome is not statistically significant at the 5% level.
- The **slope** indicates that for each unit increase in time, the number of sunspots increases by 0.0705, on average. However, the p-value for x1 is 0.056, which suggests that this effect is not statistically significant at the 5% level.
- The p-value associated with **Omnibus** is 0, lower than the chosen alpha level of 0.05, indicating that the residuals are not normally distributed.
- The **skewness** is 0.785, indicating a moderately positively skewed distribution.
- The **kurtosis** is 2.847, which is less than 3, indicating a distribution that is slightly platykurtic.
- The **Durbin-Watson** statistic is 0.367, indicating a strong positive auto-correlation.

In summary, the linear regression model suggests a slight increase in yearly sunspot counts over time. However, this trend is not statistically significant, and this model explains a very small portion of the variance in sunspot counts.

Now, we make use of the **Mann-Kendall test** (and its variants) in order to detect any trend, not necessarily linear, in the time-series of the **yearly mean total sunspot number** (see Tables 9.3, 9.4, and 9.5). In summary, while the Theil-Sen estimator suggests a slight upward trend in the yearly number of sunspots over time, all the variants of the Mann-Kendall test indicate that this trend is not statistically significant. Consequently, based on this analysis, we cannot conclude that there is a significant trend in the yearly number of sunspots.

All the aforementioned results about any existing trends for the yearly mean total sunspot number time-series are shown in Figure 9.5.

Next, we are trying to analyze the results of the **linear regression** in the time-series of the **monthly sunspot number** (see Table 9.2). We comment the following:

- An **R-squared** of 0.001 indicates that only about 0.1% of the total variance in our data can be explained by the model. This is incredibly low, indicating that just a small portion of the variability in the response data around its mean can be explained by the model.
- A p-value of 0.0377 for the **F-statistic** is less than the common alpha level of 0.05, suggesting that the relationship between your predictor and outcome is statistically significant at the 5% level.

- The **slope** indicates that for each unit increase in time, the number of sunspots increases by 0.0290. The p-value of 0.038 for x1 suggests that this effect is statistically significant at the 5% level.
- The p-value associated with **Omnibus** is 0, indicating that the residuals (the difference between the observed and predicted values) are not normally distributed.
- A **skewness** of 0.773 indicates a moderately positively skewed distribution.
- The **kurtosis** is 2.824, which is less than 3, indicating a distribution that is slightly platykurtic.
- The **Durbin-Watson** statistic is 0.004, indicating a strong positive autocorrelation in the residuals from the regression analysis.

In conclusion, the linear regression model predicts a modest rise in sunspot numbers over time. Despite the fact that the model only accounts for a relatively small percentage of the variance in sunspot counts, this trend seems to be statistically significant.

Now, we utilize the **Mann-Kendall** test (and its variants) to detect any trend, not necessarily linear, in the time-series of the **13-month smoothed monthly total sunspot number** (see Tables 9.6, 9.7, and 9.8). Theil-Sen Estimator implies a minor rising trend in the monthly number of sunspots over time, but according to all Mann-Kendall tests, this trend is not statistically significant. Therefore, we are unable to draw the conclusion that the monthly number of sunspots has a discernible trend.

All the aforementioned results about any existing trends for the 13-month smoothed monthly total sunspot number time-series are shown in Figure 9.6.

```
------------------------------
OLS Lineear Regression Results
------------------------------

Model:                           OLS    R-squared:                0.011
No. Observations:                323    Adj. R-squared:           0.008
Df Residuals:                    321    F-statistic:              3.666
AIC:                          3582.0    Prob (F-statistic):      0.0564
BIC:                          3589.0    Log-Likelihood:         -1788.8
=======================================================================
            coef    std err          t      P>|t|      [0.025     0.975]
-----------------------------------------------------------------------
const   -52.8527     68.629     -0.770      0.442    -187.872     82.167
x1        0.0705      0.037      1.915      0.056      -0.002      0.143
=======================================================================
Omnibus:                      27.857    Durbin-Watson:            0.367
Prob(Omnibus):                 0.000    Jarque-Bera (JB):        33.487
Skew:                          0.785    Prob(JB):              5.35e-08
Kurtosis:                      2.847    Cond. No.              3.72e+04
```

Table 9.1: Linear regression results in the time-series of the yearly mean total sunspot number.

```
----------------------
OLS Regression Results
----------------------

Model:                          OLS    R-squared:                0.001
No. Observations:              3282    Adj. R-squared:           0.001
Df Residuals:                  3280    F-statistic:              4.324
AIC:                      3.652e+04    Prob (F-statistic):      0.0377
BIC:                      3.653e+04    Log-Likelihood:        -18258.0
======================================================================
            coef    std err          t      P>|t|      [0.025    0.975]
----------------------------------------------------------------------
const    26.8593     26.330      1.020      0.308     -24.765    78.483
x1        0.0290      0.014      2.079      0.038       0.002     0.056
======================================================================
Omnibus:                    265.416    Durbin-Watson:            0.004
Prob(Omnibus):                0.000    Jarque-Bera (JB):       330.754
Skew:                         0.773    Prob(JB):              1.51e-72
Kurtosis:                     2.824    Cond. No.              4.51e+04
```

Table 9.2: Linear regression results in the time-series of the 13-month smoothed monthly total sunspot number.

```
------------------------------------------------
Hamed and Rao Modified Mann-Kendall Test Results
------------------------------------------------

Null Hypothesis:                  There is no trend in the series.
Alternative Hypothesis:           There is trend in the series.

Trend:                            No trend
Hypothesis Test Result (h):       False
p-value:                          0.1071
Test Statistic (z):               1.6116
Kendall's Tau:                    0.0622
Mann-Kendall Score (s):           3236.0
Variance (var_s):                 4029602.9833
Slope:                            0.0484
Intercept:                        57.5097
```

Table 9.3: Hamed and Rao modified Mann-Kendall test results in the time-series of the yearly mean total sunspot number.

```
-----------------------------------------------
Yue and Wang Modified Mann-Kendall Test Results
-----------------------------------------------

Null Hypothesis:                  There is no trend in the series.
Alternative Hypothesis:           There is trend in the series.

Trend:                            No trend
Hypothesis Test Result (h):       False
p-value:                          0.1966
Test Statistic (z):               1.2913
Kendall's Tau:                    0.0622
Mann-Kendall Score (s):           3236.0
Variance (var_s):                 6276410.9540
Slope:                            0.0484
Intercept:                        57.5097
```

Table 9.4: Yue and Wang modified Mann-Kendall test results in the time-series of the yearly mean total sunspot number.

```
----------------------------------
Seasonal Mann-Kendall Test Results
----------------------------------

Null Hypothesis:                  There is no trend in the series.
Alternative Hypothesis:           There is trend in the series.

Trend:                            No trend
Hypothesis Test Result (h):       False
p-value:                          0.0953
Test Statistic (z):               1.6680
Kendall's Tau:                    0.0622
Mann-Kendall Score (s):           3236.0
Variance (var_s):                 3761480.6667
Slope:                            0.0484
Intercept:                        57.5097
```

Table 9.5: Seasonal Mann-Kendall test results in the time-series of the yearly mean total sunspot number.

```
------------------------------------------------
Hamed and Rao Modified Mann-Kendall Test Results
------------------------------------------------

Null Hypothesis:                  There is no trend in the series.
Alternative Hypothesis:           There is trend in the series.

Trend:                            No trend
Hypothesis Test Result (h):       False
p-value:                          0.8633
Test Statistic (z):               0.1721
Kendall’s Tau:                    0.0122
Mann-Kendall Score (s):           65495.0
Variance (var_s):                 144769808459.2507
Slope:                            0.0010
Intercept:                        69.9727
```

Table 9.6: Hamed and Rao modified Mann-Kendall test results in the time-series of the 13-month smoothed monthly total sunspot number.

```
-----------------------------------------------
Yue and Wang Modified Mann-Kendall Test Results
-----------------------------------------------

Null Hypothesis:                  There is no trend in the series.
Alternative Hypothesis:           There is trend in the series.

Trend:                            No trend
Hypothesis Test Result (h):       False
p-value:                          0.8725
Test Statistic (z):               0.1605
Kendall’s Tau:                    0.0122
Mann-Kendall Score (s):           65495.0
Variance (var_s):                 166595238084.2805
Slope:                            0.0010
Intercept:                        69.9727
```

Table 9.7: Yue and Wang modified Mann-Kendall test results in the time-series of the 13-month smoothed monthly total sunspot number.

```
----------------------------------
Seasonal Mann-Kendall Test Results
----------------------------------

Null Hypothesis:                  There is no trend in the series.
Alternative Hypothesis:           There is trend in the series.

Trend:                            No trend
Hypothesis Test Result (h):       False
p-value:                          0.2493
Test Statistic (z):               1.1520
Kendall's Tau:                    0.0135
Mann-Kendall Score (s):           6034.0
Variance (var_s):                 27426600.0
Slope:                            0.0133
Intercept:                        69.7772
```

Table 9.8: Seasonal Mann-Kendall test results in the time-series of the 13-month smoothed monthly total sunspot number.

```
---------------------------------------------
Correlated Seasonal Mann-Kendall Test Results
---------------------------------------------

Null Hypothesis:                  There is no trend in the series.
Alternative Hypothesis:           There is trend in the series.

Trend:                            No trend
Hypothesis Test Result (h):       False
p-value:                          0.7549
Test Statistic (z):               0.3121
Kendall's Tau:                    0.0125
Mann-Kendall Score (s):           5547.0
Variance (var_s):                 315798852.3333
Slope:                            0.0133
Intercept:                        69.7772
```

Table 9.9: Correlated seasonal Mann-Kendall test results in the time-series of the 13-month smoothed monthly total sunspot number.

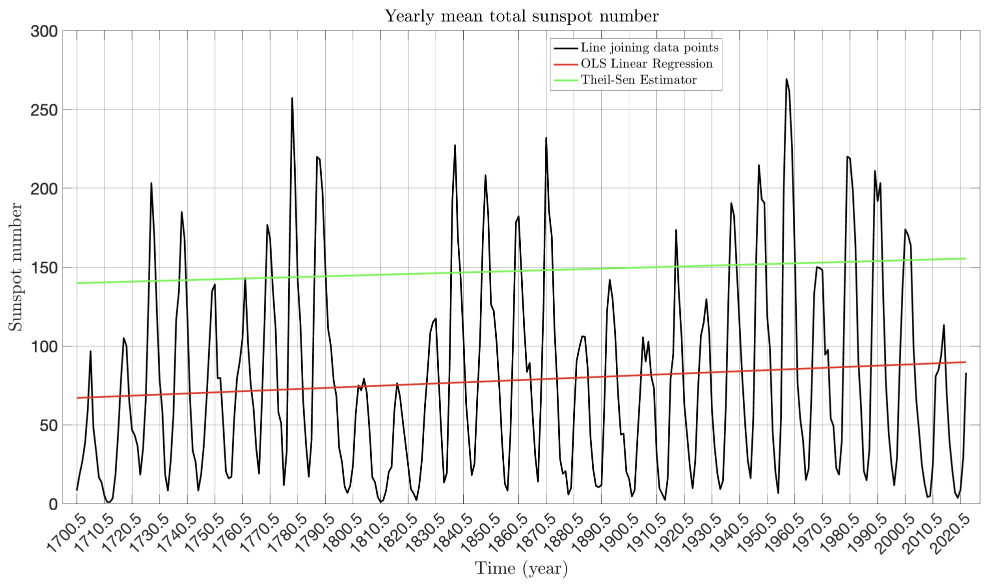


Figure 9.5: Yearly mean total sunspot number time-series, along with various linear trends estimations. The significance of these trends is being tested in the text above.

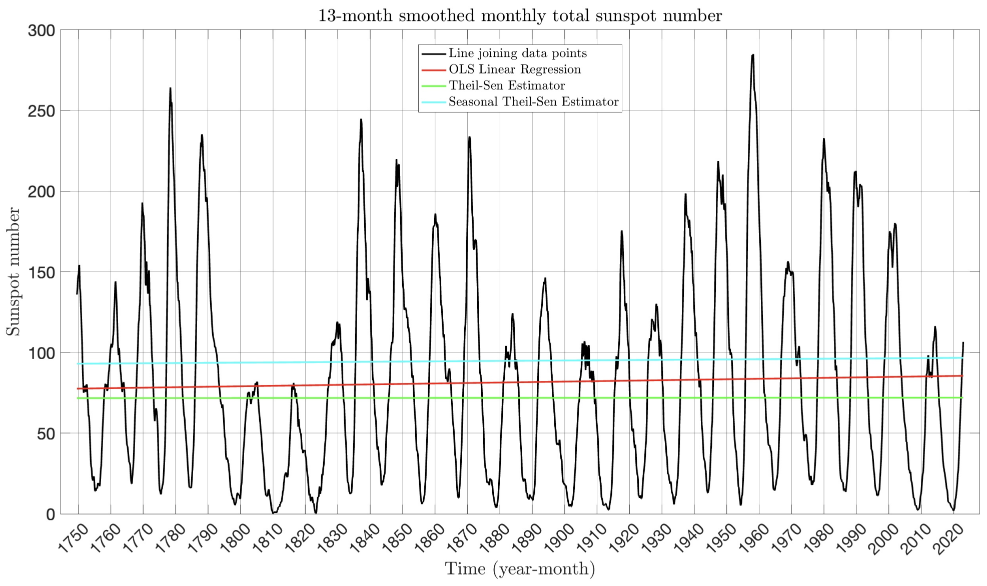


Figure 9.6: 13-month smoothed monthly total sunspot number time-series, along with various linear trends estimations. The significance of these trends is being tested in the text above.

9.1.3 Stationarity analysis

In time-series analysis, a fundamental assumption often made is that the data are stationary. A stationary time-series is one whose properties do not depend on the time at which the series is observed. In other words, it does not exhibit trends or seasonality and its variance is constant over time. This assumption is crucial because many statistical modeling techniques require the data to be stationary to make reliable forecasts. However, many real-world time-series data exhibit trends, seasonality, or other non-stationary behaviors. Therefore, before applying any statistical model, it is important to first check whether the sunspot data are stationary or not.

There are several statistical tests available to test the stationarity of a time-series. Among these, the Augmented Dickey-Fuller, the Dickey-Fuller Generalized Least Squares, the Phillips-Perron, and the Kwiatkowski–Phillips–Schmidt–Shin tests are widely used. The first three are unit root tests, while the last one is a trend-stationarity test. In Tables 9.10, 9.11, 9.12, and 9.13, we apply these tests on the time-series of the yearly mean total sunspot number, while in Tables 9.14, 9.15, 9.16, and 9.17, these tests are applied on the time-series of the 13-month smoothed monthly total sunspot number.

```
----------------------------
Augmented Dickey-Fuller Test
----------------------------
Null Hypothesis:        The series contains a unit root
Alternative Hypothesis: The series does not contain a unit root
-------------------------------------
Test Statistic                 -3.187
P-value                         0.021
Lags                                8
-------------------------------------
Trend: Constant
Critical Values: -3.45 (1%), -2.87 (5%), -2.57 (10%)
```

Table 9.10: ADF test results in the time-series of the yearly mean total sunspot number.

```
----------------------
Dickey-Fuller GLS Test
----------------------
Null Hypothesis:        The series contains a unit root
Alternative Hypothesis: The series does not contain a unit root
-------------------------------------
Test Statistic                 -1.187
P-value                         0.222
Lags                                8
-------------------------------------
Trend: Constant
Critical Values: -2.63 (1%), -2.01 (5%), -1.69 (10%)
```

Table 9.11: DF-GLS test results in the time-series of the yearly mean total sunspot number.

```
--------------------
Phillips-Perron Test
--------------------
Null Hypothesis:        The series contains a unit root
Alternative Hypothesis: The series does not contain a unit root
-------------------------------------
Test Statistic                 -4.469
P-value                         0.000
Lags                                8
-------------------------------------
Trend: Constant
Critical Values: -3.45 (1%), -2.87 (5%), -2.57 (10%)
```

Table 9.12: PP test results in the time-series of the yearly mean total sunspot number.

```
---------
KPSS Test
---------
Null Hypothesis:        The series is trend stationary
Alternative Hypothesis: The series is non-stationary
-------------------------------------
Test Statistic                  0.233
P-value                         0.213
Lags                                7
-------------------------------------
Trend: Constant
Critical Values: 0.74 (1%), 0.46 (5%), 0.35 (10%)
```

Table 9.13: KPSS test results in the time-series of the yearly mean total sunspot number.

```
----------------------------
Augmented Dickey-Fuller Test
----------------------------
Null Hypothesis:        The series contains a unit root
Alternative Hypothesis: The series does not contain a unit root
-------------------------------------
Test Statistic                -10.404
P-value                         0.000
Lags                               29
-------------------------------------
Trend: Constant
Critical Values: -3.43 (1%), -2.86 (5%), -2.57 (10%)
```

Table 9.14: ADF test results in the time-series of the 13-month-smoothed monthly total sunspot number.

```
----------------------
Dickey-Fuller GLS Test
----------------------
Null Hypothesis:        The series contains a unit root
Alternative Hypothesis: The series does not contain a unit root
-------------------------------------
Test Statistic                 -6.451
P-value                         0.000
Lags                               29
-------------------------------------
Trend: Constant
Critical Values: -2.57 (1%), -1.95 (5%), -1.63 (10%)
```

Table 9.15: DF-GLS test results in the time-series of the 13-month-smoothed monthly total sunspot number.

```
--------------------
Phillips-Perron Test
--------------------
Null Hypothesis:        The series contains a unit root
Alternative Hypothesis: The series does not contain a unit root
-------------------------------------
Test Statistic                 -6.593
P-value                         0.000
Lags                               29
-------------------------------------
Trend: Constant
Critical Values: -3.43 (1%), -2.86 (5%), -2.57 (10%)
```

Table 9.16: PP test results in the time-series of the 13-month-smoothed monthly total sunspot number.

```
---------
KPSS Test
---------
Null Hypothesis:        The series is trend stationary
Alternative Hypothesis: The series is non-stationary
-------------------------------------
Test Statistic                  0.104
P-value                         0.569
Lags                               36
-------------------------------------
Trend: Constant
Critical Values: 0.74 (1%), 0.46 (5%), 0.35 (10%)
```

Table 9.17: KPSS test results in the time-series of the 13-month-smoothed monthly total sunspot number.

For the time-series of the yearly mean total sunspot number, we see that all unit root tests give a p-value less than 0.05 (5% significance level), which indicates that the series does not contain a unit root, and consequently it is weakly stationary. The KPSS test also advocates for this result, as it indicates that the series is trend-stationary by producing a p-value greater than 0.05 (5% significance level).

For the time-series of the 13-month smoothed monthly total sunspot number, we see that ADF and Phillips-Perron say that the series does not contain a unit root, while the DF-GLS test indicates that there is a unit root (5% significance level). However, the KPSS test says that the series is trend-stationary. Some of the tests produce contradictory results, so we cannot draw a clear conclusion. Nonetheless, by majority voting, let's say that the series might be stationary.

9.1.4 Heteroscedasticity analysis

In time-series analysis, an assumption often made is that the data are homoscedastic. A homoscedastic time-series is one whose variance does not change over time. This assumption is crucial because many statistical modeling techniques require the data to be homoscedastic to make reliable forecasts. However, many real-world time-series data exhibit heteroscedasticity. Therefore, before applying any statistical model, it is important to first check whether the sunspot data are homoscedastic or not.

There are several statistical tests available to test the heteroscedasticity of a time-series. Among these, the White, and the Breusch-Pagan tests are widely used. In Tables 9.18, and 9.19, we apply these tests on the time-series of the yearly mean total sunspot number, while in Tables 9.20, and 9.21, these tests are applied on the time-series of the 13-month smoothed monthly total sunspot number.

For the time-series of the yearly mean total sunspot number, we see that White test give a p-value greater than 0.05 (5% significance level), which indicates that the series is homoscedastic, while Breusch-Pagan test give a p-value slightly less than 0.05 indicating heteroscedasticity. For the time-series of the 13-month smoothed monthly total sunspot number, we see that both tests give a p-value less than 0.05 (5% significance level), which indicates that the series is heteroscedastic. In general, we can conclude that both time-series are more likely to be heteroscedastic.

```
----------
White Test
----------

Null Hypothesis:        The error variance does not depend on x
Alternative Hypothesis: The error variance depends on x

-------------------------------------
Test Statistic                  1.932
P-value                         0.147
-------------------------------------
```

Table 9.18: White test results in the time-series of the yearly mean total sunspot number.

```
------------------
Breusch-Pagan Test
------------------

Null Hypothesis:        The residual variance does not depend on x
Alternative Hypothesis: The residual variance depends on x

-------------------------------------
Test Statistic                  3.843
P-value                         0.049
-------------------------------------
```

Table 9.19: Breusch-Pagan test results in the time-series of the yearly mean total sunspot number.

```
----------
White Test
----------

Null Hypothesis:        The error variance does not depend on x
Alternative Hypothesis: The error variance depends on x

-------------------------------------
Test Statistic                  7.565
P-value                     5.273e-04
-------------------------------------
```

Table 9.20: White test results in the time-series of the 13-month-smoothed monthly total sunspot number.

```
------------------
Breusch-Pagan Test
------------------

Null Hypothesis:        The residual variance does not depend on x
Alternative Hypothesis: The residual variance depends on x

-------------------------------------
Test Statistic                 14.641
P-value                     1.325e-04
-------------------------------------
```

Table 9.21: Breusch-Pagan test results in the time-series of the 13-month-smoothed monthly total sunspot number.

9.2 Sunspot time-series analysis in frequency domain

In this subsection, we analyze the sunspot time-series (yearly and monthly) in the frequency domain in order to perform a seasonality analysis.

9.2.1 Frequency spectrum and seasonality analysis

Sunspot numbers have been recorded for centuries, providing a time-series that offers insights into the solar activity cycle. These sunspot numbers vary over time, with periods of high and low activity. To better understand the periodicities embedded within this data, a frequency domain analysis is conducted using the Fast Fourier Transform (FFT).

The FFT is an efficient algorithm used to compute the Discrete Fourier Transform (DFT) and its inverse. By converting our time-series data into the frequency domain using FFT, we can identify the dominant frequencies, which represent the main periodicities in the sunspot activity. The use of FFT in analyzing the sunspot number time-series is instrumental in deciphering the underlying periodicities of solar activity. By isolating these dominant frequencies, we gain a better understanding of the sun's behavior. In Figures 9.7, and 9.8, the FFT coefficient in the complex plane and the frequency spectrum of the yearly mean total sunspot number time-series are shown. Additionally, in Figures 9.9, and 9.10, the FFT coefficient in the complex plane and the frequency spectrum of the 13-month smoothed monthly total sunspot number time-series are shown.

Upon visualizing the frequency spectrum of both time-series, specific dominant peaks can be identified that correspond to known cycles of sunspot activity. The most notable peak in the frequency spectrum of the yearly sunspots is at frequency 0.09 year^{-1}, representing the nearly 11-year solar cycle, a well-documented periodicity in sunspot numbers. In the frequency spectrum, we see a dominant peak at frequency 0.0076 month^{-1} (or 132 months), which is the periodicity corresponding to the 11-year solar cycle. Additionally, other minor peaks may provide insights into less prominent cycles or harmonics in the sunspot data. Such harmonics can be seen at frequencies 0.008 year^{-1}, 0.0181 year^{-1}, and 0.1181 year^{-1} in the yearly sunspots time-series, or at frequencies 0.0009 month^{-1}, 0.007 month^{-1}, and 0.0098 month^{-1} in the monthly sunspots time-series.

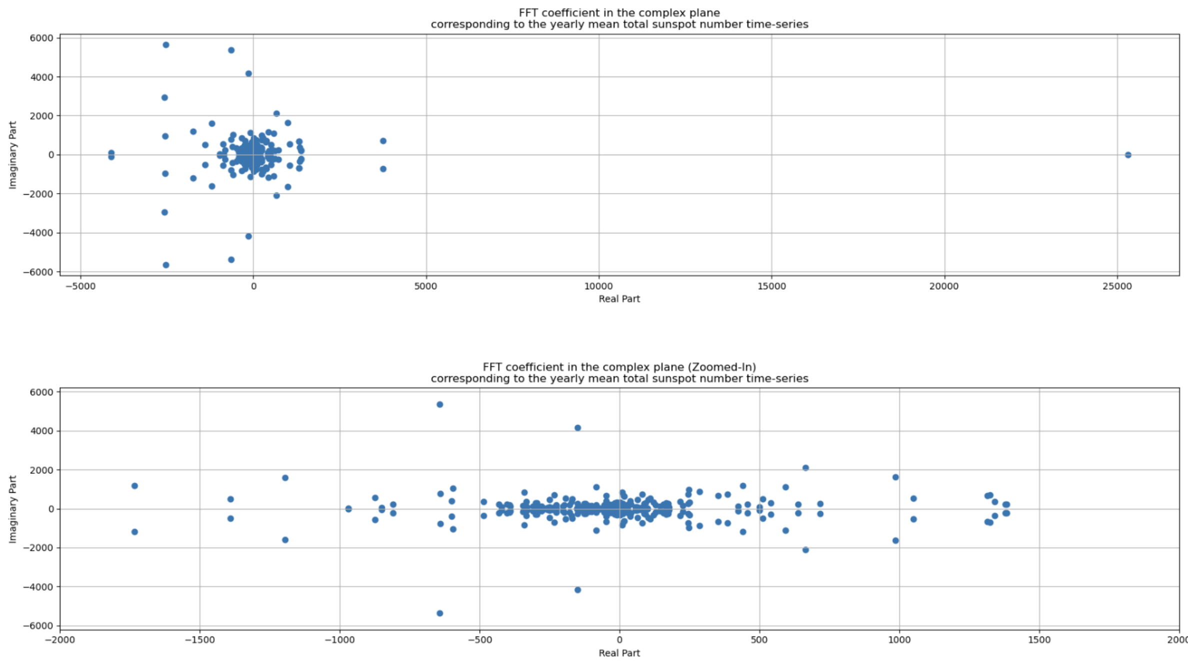


Figure 9.7: Fast Fourier Transform (FFT) coefficients in the complex plane, corresponding to the yearly mean total sunspot number time-series.

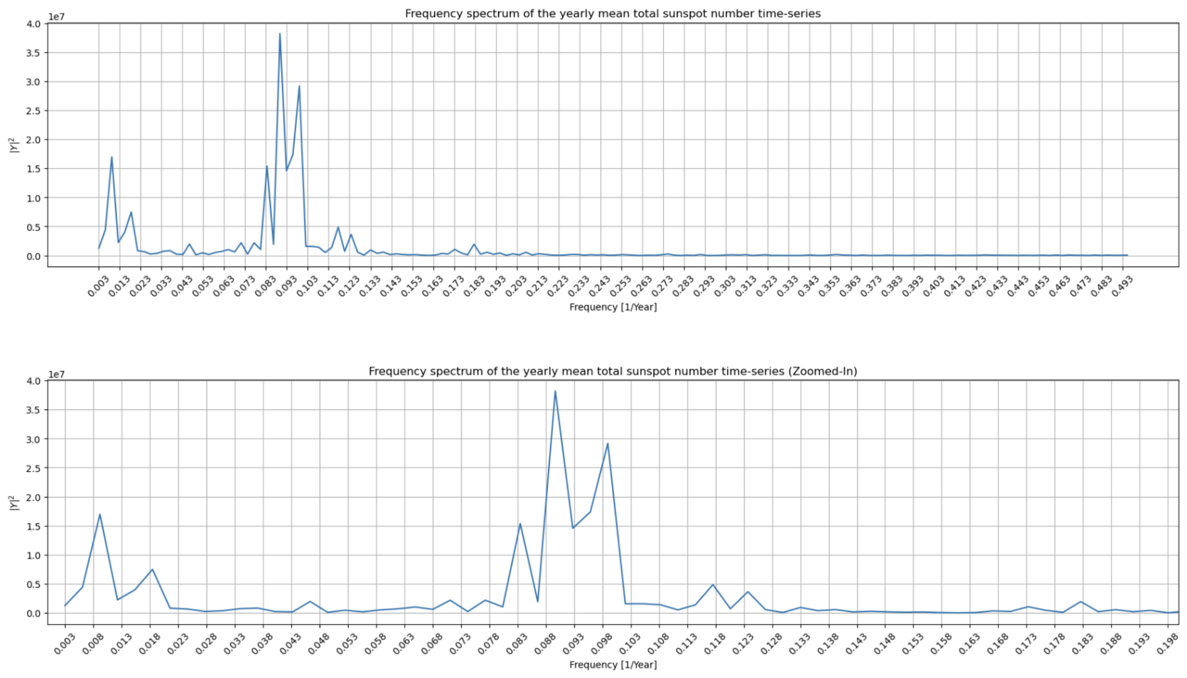
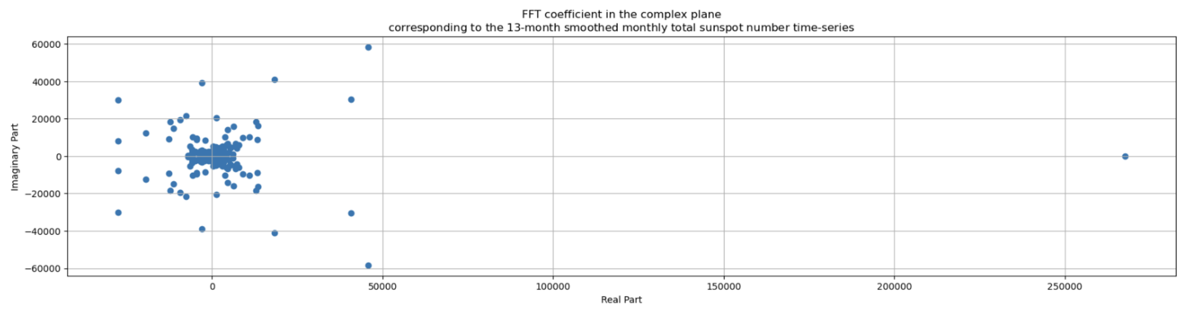


Figure 9.8: Frequency spectrum of the yearly mean total sunspot number time-series. It is calculated by Fast Fourier Transform (FFT).



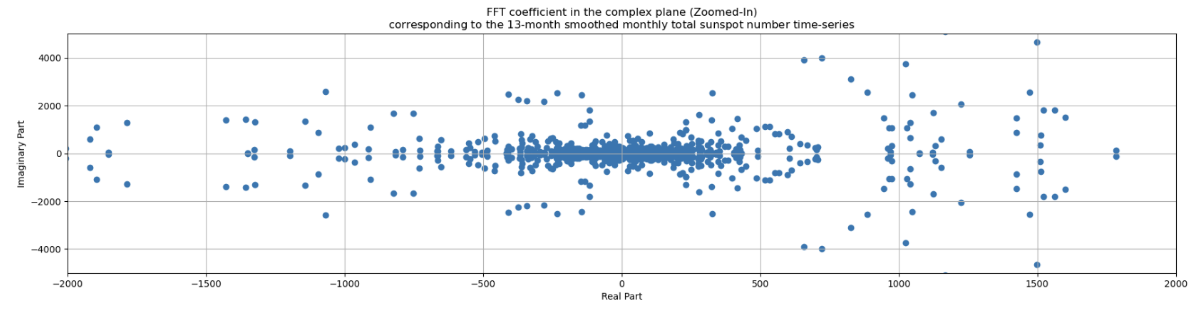
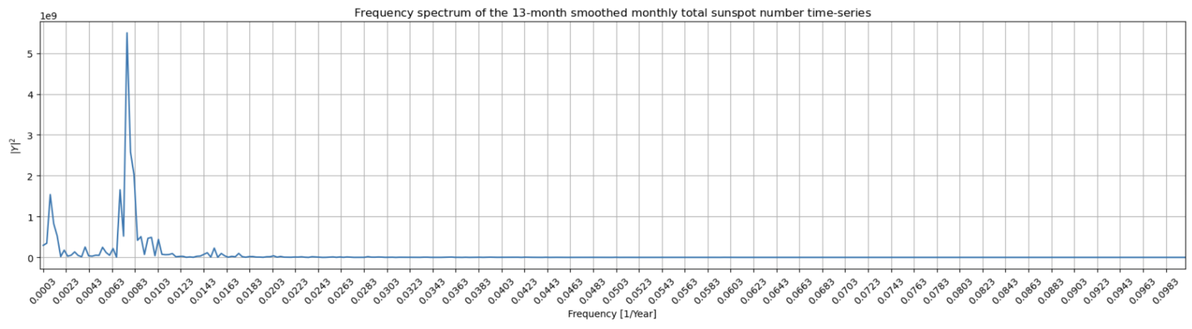


Figure 9.9: Fast Fourier Transform (FFT) coefficients in the complex plane, corresponding to the 13-month smoothed monthly total sunspot number time-series.



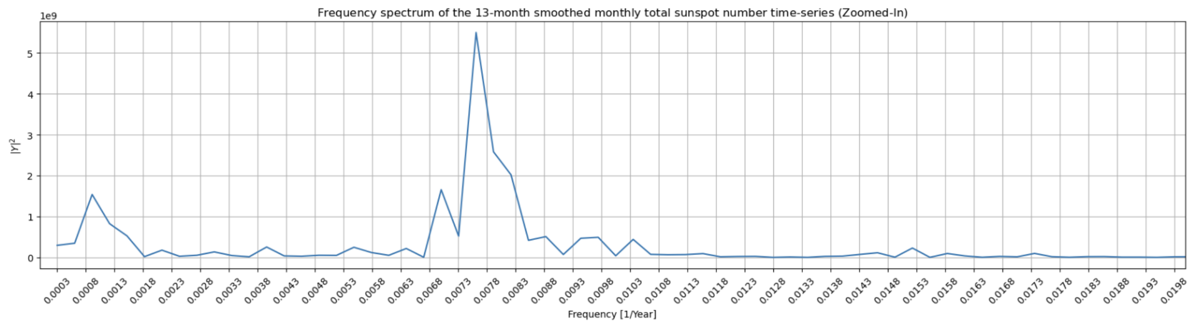


Figure 9.10: Frequency spectrum of the 13-month smoothed monthly total sunspot number time-series. It is calculated by Fast Fourier Transform (FFT).

10 Forecasting the Yearly Number of Sunspots

In previous sections, we introduced three different machine learning models that can be used for time-series forecasting. Now, we will use **Warped Gaussian Process Regression (Warped GPR)**, **Light Gradient Boosting Machine (LightGBM)**, and **Long Short-Term Memory (LSTM)** in order to predict the **yearly mean total sunspot number** time-series. In this section, we delve into the results obtained from employing these models in forecasting sunspot activity, the pros and cons of each method, and how they are compared to other methods and results in the bibliography.

10.1 Warped Gaussian Process Regression (Warped GPR)

10.1.1 Data pre-processing

As analyzed in section 5, Gaussian Process Regression (GPR) framework often assumes that the likelihood of the observed data points, given the latent function values, follows a Gaussian distribution with some noise, as well as that the joint distribution of the observed data at different input points follows a multivariate Gaussian distribution. However, we saw in section 9 that the histogram of the yearly mean sunspot number is more likely to follow an exponential distribution with a right-skewness. To boost GPR performance, it is a good practice to apply first a non-linear transformation to the data and get them closer to "normality", and then work on the transformed data. Then, when we have the predictions made by the GPR model, we apply the inverse transformation to get the predicted value in the observation space domain.

In our case, we apply the **κ-logarithmic transformation**, to account for the skewness and the non-negativity of the original sunspot data. For the training phase, explained below, the train-test split was $90\% - 10\%$, respectively. In Figure 10.2, we see the original time-series of the yearly mean sunspot number, the histogram of all data, the histogram of the train data, and the histogram of the test data. The transformed time-series of the yearly mean sunspot number, the histogram of the transformed train data, and the histogram of the transformed test data are shown in Figure 10.3.

To assess the normality of the transformed time-series data, we employ the **Kolmogorov-Smirnov** test (see Appendix C), a widely recognized statistical non-parametric test used to determine whether a given dataset follows a normal distribution. This test plays a pivotal role in validating the assumption of normality, which can be crucial under the GPR framework. The Kolmogorov-Smirnov test quantifies the maximum vertical discrepancy between the empirical cumulative distribution function (ECDF) of the transformed data and the cumulative distribution function (CDF) of the standard normal distribution. In other words, it assesses how closely our data aligns with a theoretical standard normal distribution. The null hypothesis of this test is that the data follows a standard normal distribution. Upon performing the Kolmogorov-Smirnov test, we obtain a p-value of 0.27, which is above 0.05 (assume 5% significance level). Consequently, we cannot reject the null hypothesis on a 5% significance level and the data can be assumed normal. In Figure 10.1, we can see the CDF and the ECDF that correspond to our data from the transformed yearly mean total sunspot number after performing standardization (the Kolmogorov-Smirnov test is applied on the standardized data).

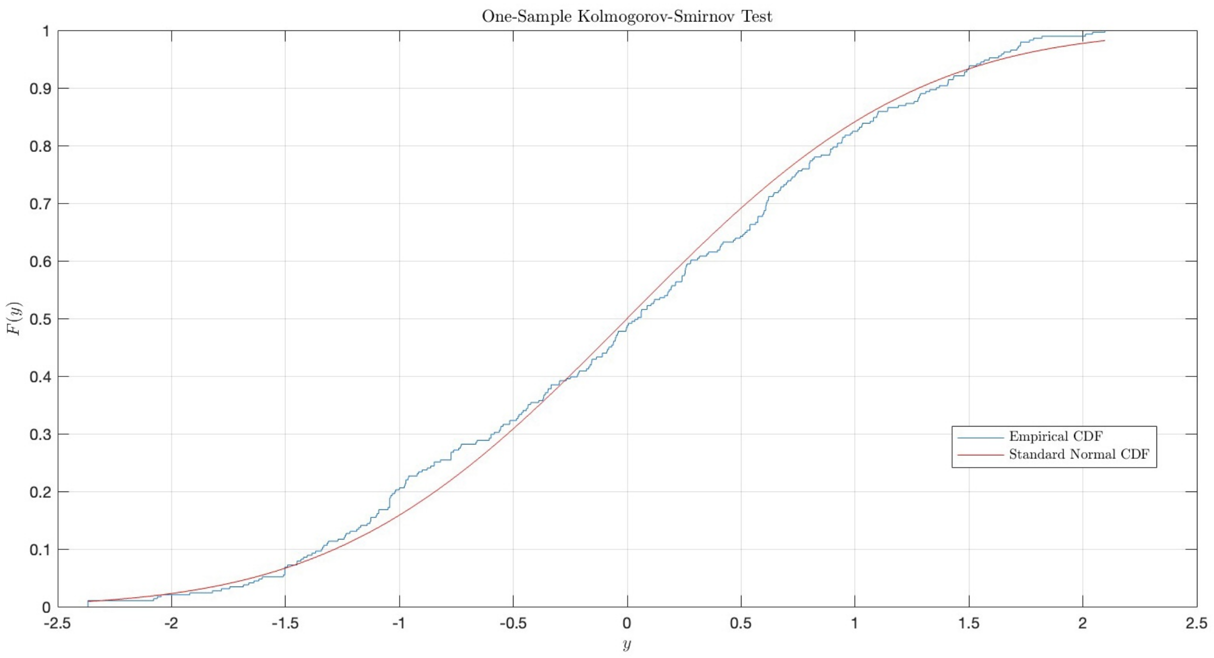
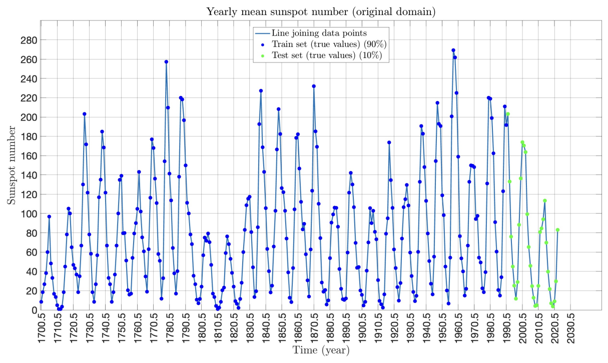


Figure 10.1: One-sample Kolmogorov-Smirnov test to check the normality of the transformed sunspot data. This figure depicts the CDF of the standard normal distribution, and the empirical CDF obtained by the transformed data. We can see that they are very close.

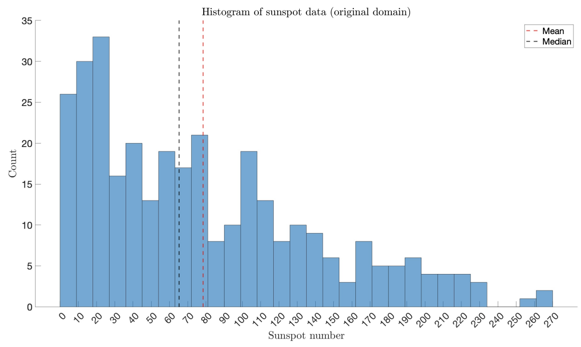
In Table 10.1, we can see the sample mean and the sample standard deviation for the time-series of the yearly mean sunspot number, in the original as well as in the transformed domain. Additionally, the mean and the standard deviation are computed for their train and test subsets.

	Original data	Original train data	Original test data	Transformed data	Transformed train data	Transformed test data
$\hat{\mu}$	78.39	79.54	67.90	6.54	6.60	6.04
$\hat{\sigma}$	61.95	62.44	57.11	2.79	2.79	2.82

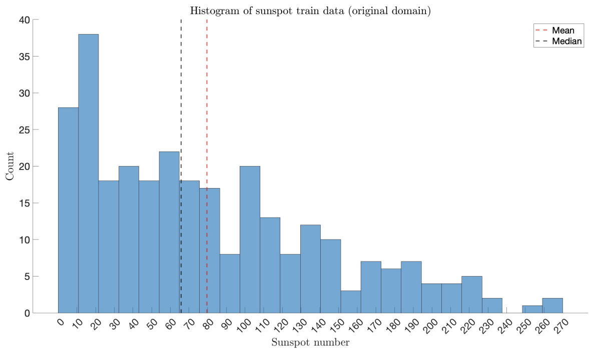
Table 10.1: Sample mean ($\hat{\mu}$) and sample standard variation ($\hat{\sigma}$) for the original data, the transformed data, and their train and test splits.



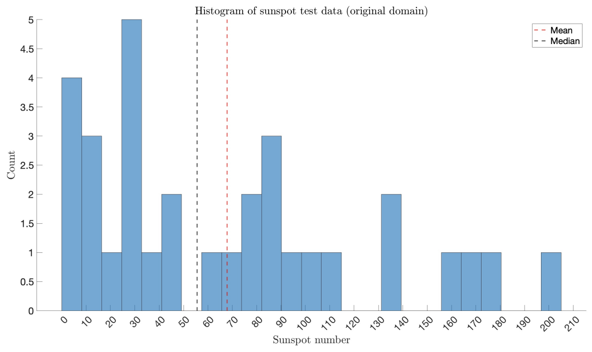
(a) Plot of the original sunspot time-series.



(b) Histogram of the original sunspot data.

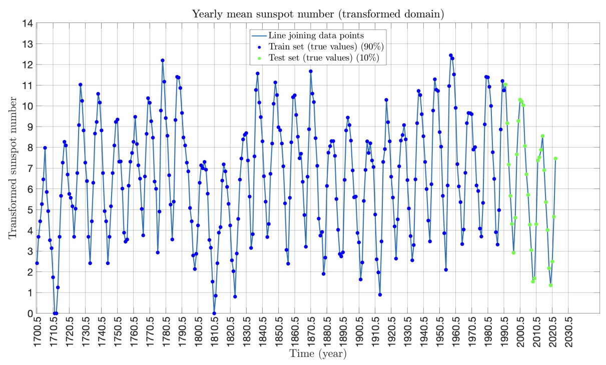


(c) Histogram of the original train sunspot data.

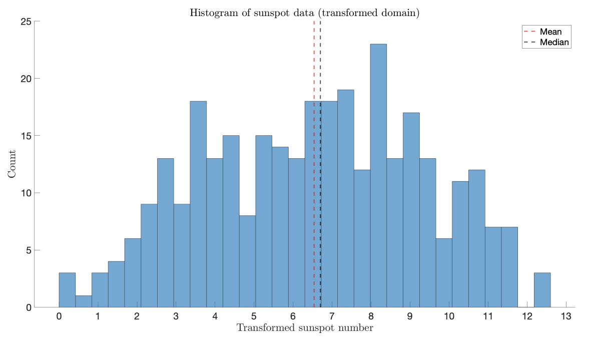


(d) Histogram of the original test sunspot data.

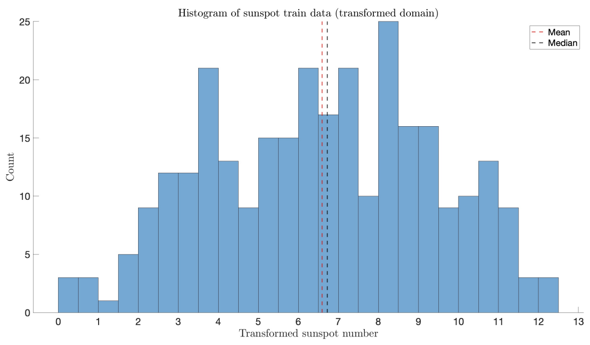
Figure 10.2: (a) Time-series of the yearly mean sunspot number in the original domain. (b) Histogram of the time-series data. (c) Histogram of the train data (90%). Training set contains observations from 1700.5 to 1990.5 (291 data points). (d) Histogram of the test data (10%). Test set contains observations from 1991.5 to 2022.5 (32 data points).



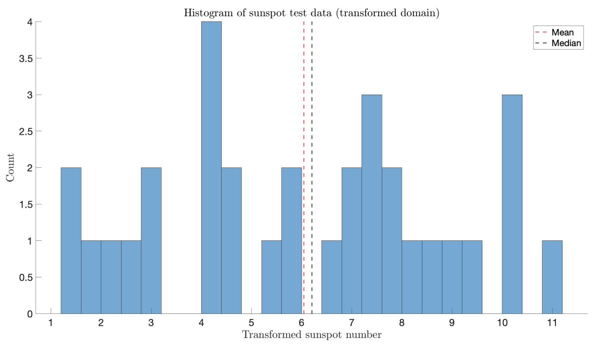
(a) Plot of the transformed sunspot time-series.



(b) Histogram of the transformed sunspot data.



(c) Histogram of the transformed train sunspot data.



(d) Histogram of the transformed test sunspot data.

Figure 10.3: (a) Time-series of the yearly mean sunspot number in the transformed domain. (b) Histogram of the time-series data. Training set contains observations from 1700.5 to 1990.5 (291 data points). (c) Histogram of the train data (90%). (d) Histogram of the test data (10%). Test set contains observations from 1991.5 to 2022.5 (32 data points).

As already mentioned, in time-series analysis an assumption often made is that the data are homoscedastic. A homoscedastic time-series is one whose variance does not change over time. This assumption is crucial because many statistical modeling techniques require the data to be homoscedastic to make reliable forecasts. We saw in a previous subsection that the time-series of the yearly mean total sunspot number is heteroscedastic.

Now, we will ensure that the yearly sunspot data become homoscedastic after applying the κ-logarithmic transformation. There are several statistical tests available to test the heteroscedasticity of a time-series. In Tables 10.2, and 10.3, we apply the White and the Breusch-Pagan test on the time-series of the transformed yearly mean total sunspot number, respectively. We see that both tests give a p-value greater than 0.05 (5% significance level), which indicates that the series is homoscedastic.

```
----------
White Test
----------

Null Hypothesis:        The error variance does not depend on x
Alternative Hypothesis: The error variance depends on x

-------------------------------------
Test Statistic                  0.520
P-value                         0.593
-------------------------------------
```

Table 10.2: White test results in the transformed time-series of the yearly mean total sunspot number.

```
------------------
Breusch-Pagan Test
------------------

Null Hypothesis:        The residual variance does not depend on x
Alternative Hypothesis: The residual variance depends on x

-------------------------------------
Test Statistic                  0.790
P-value                         0.373
-------------------------------------
```

Table 10.3: Breusch-Pagan test results in the transformed time-series of the yearly mean total sunspot number.

10.1.2 Model selection

In GPR framework, it is essential to define first a GP prior with a mean function $m(\mathbf{x})$ and kernel function $k(\mathbf{x}, \mathbf{x}')$. In our case, for modeling the yearly mean sunspot number, the input variable is one-dimensional, and it corresponds to the time. So instead of $\mathbf{x}$, the notation x will be used. For the design matrix $\mathbf{X}$, it is just the row vector $(x_1, x_2, \ldots, x_N)$ for N observations. We select a **constant mean function** and a covariance function that is the product of an **exponential kernel** and a **periodic kernel**. More specific, it is:

$$m_{\boldsymbol{\theta}}(x) = c$$

$$k_{\boldsymbol{\theta}}(x, x') = \sigma_f^2 \exp\left(-\frac{\|x - x'\|_2}{\ell_1}\right) \exp\left(-\frac{2\sin^2\left(\frac{\pi\|x-x'\|_2}{p}\right)}{\ell_2^2}\right)$$

where $\boldsymbol{\theta} = (\sigma_f, \ell_1, p, \ell_2, \sigma_n, c)^T$ is the vector containing the model's hyperparameters. Note that σ_n^2, noise variance, is modeled through an additive component of Gaussian noise to the true observations (not through the kernel function itself), and it is present in the posterior predictive formula. Intuitively speaking, the kernel choice is made with the purpose to model the **periodicity** of the sunspot number time-series and its **non-smoothness**. Other kernel functions were, also, tested, but they produced less good results in terms of prediction accuracy.

During the training phase of the GPR model, our objective is to find the optimal set of hyperparameters, $\boldsymbol{\theta}_*$, that maximizes the log marginal likelihood, or equivalently minimizes the negative log marginal likelihood. Remember that the formula to obtain log marginal likelihood is

$$\begin{aligned}\log\left(p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})\right) = & \underbrace{-\frac{1}{2}(\mathbf{y} - m_{\boldsymbol{\theta}}(\mathbf{X}))^T \left[K_{\boldsymbol{\theta}}(\mathbf{X}, \mathbf{X}) + \sigma_n^2\mathbf{I}\right]^{-1}(\mathbf{y} - m_{\boldsymbol{\theta}}(\mathbf{X}))}_{\text{Data fit}} \\ & \underbrace{-\frac{1}{2}\log\left[\det\left(K_{\boldsymbol{\theta}}(\mathbf{X}, \mathbf{X}) + \sigma_n^2\mathbf{I}\right)\right]}_{\text{Complexity penalty}} \\ & \underbrace{-\frac{n}{2}\log(2\pi)}_{\text{Constant term}}\end{aligned}$$

and then optimal $\boldsymbol{\theta}_*$ is given by

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}}\left[-\log\left(p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})\right)\right]$$

The optimal values of the hyperparameters are shown in Table 10.4.

For the aforementioned minimization, we use the **Global Optimization Toolbox** from **MATLAB**. That toolbox provides several methods to perform global optimization, like "global search", "multi-start", "surrogate optimization", "pattern search", "genetic algorithm", "particle swarm", "simulated annealing", and "multi-objective optimization". One can use more than one method, in a way to "cross-validate" the results.

It is worth noting that the optimal values we get through the optimization for the hyperparameters have a straightforward interpretation. The σ_f is taking a value close to the sample standard deviation of the transformed training data. The p is taking a

Parameter	σ_f	ℓ_1	p	ℓ_2	σ_n	c
Optimal value	2.99	28.20	10.9	1.37	0.01	6.53
Bounds	$(0.01, 200)$	$(0.01, 200)$	$(5, 15)$	$(0.01, 200)$	(1e-16, 0.01)	$(-50, 100)$
Optimal LML	-432.28					

Table 10.4: **Warped GPR**. Optimal set of hyperparameters $\boldsymbol{\theta}_*$, the bounds that were set during the optimization, and the optimal value of the log marginal likelihood (LML).

value near the predominant 11-year periodicity of the sunspot number. The ℓ_1 is taking a value close to 1 corresponding to the smoothness of the sunspot number from year to year, while ℓ_2 is taking a relatively large value indicating that the periodic covariance is significant over a longer lengthscale and keeping the repetitions close to each other more consistent. Finally, c, is getting approximately the value of the sample mean of the transformed training data.

10.1.3 Model evaluation

Now, we will visualize the resulting model in terms of its forecasting capabilities in various scenarios. As forecasting strategies, we will use the “direct one-step ahead” method (see Equation (3.1)), as well as the “multiple output multi-step ahead” method (see Equation (3.3)) and assess their performance. The first one is used to make 1-step ahead predictions (1-SA), while the second one is used to make 6-steps ahead predictions (6-SA), 12-steps ahead predictions (12-SA), and 32-steps ahead predictions (32-SA). Note that in all cases, the trained model remains the **same**, which is defined when the optimal values for the hyperparameters are set.

Before we get through the assessment of each model, it is important to remember a characteristic point of Warped GPR framework. For a Vanilla GPR, the point estimates are “ready to use”, and the predictive distribution is Gaussian, which means that the median and mean lie at the same point. For the Warped GPR, when applying the inverse transformation to the point estimates produced by the model, the distribution that results, in general, has different values for the mean and the median (mainly induced by the distribution of the data in the original domain). Consequently, one can use both estimates (those obtained by the median, and those obtained by the mean) to produce predictions.

Below, we will refer to the predictions, and their accuracy, made by each model (for each forecasting strategy), when we consider the median value as the point estimate. Only for the case of 1-SA prediction, we consider also the mean value as the point estimate. In the Appendix I, there are figures showing the rest of the models, when considering the mean value as the point estimate.

In Table 10.5 and Table 10.6, we can see various evaluation metrics for the predictions made by each model, when we consider both types of point estimates, the median and the mean, respectively.

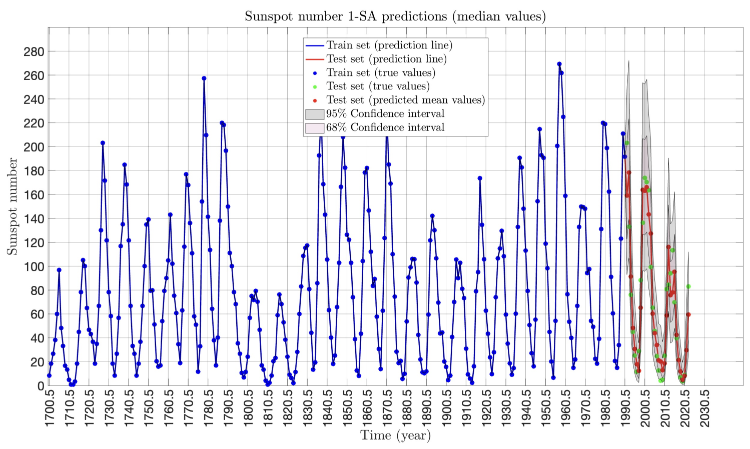
As expected, the prediction accuracy decreases as the forecasting horizon gets larger. It means that, when trying to predict sunspot number for more years ahead, we get larger uncertainty and discrepancies from the true values. It is worth noting that RMSE(%), in all forecasting strategies, is less than 10, which indicates a good fit of the model and a competitive accuracy.

Metric	1-SA	6-SA	12-SA	32-SA
MAE	14.72	23.37	28.16	29.28
RMSE	19.41	28.04	34.20	33.13
RRMSE (%)	3.93	6.30	6.64	6.64
R^2	0.88	0.75	0.63	0.65
Correlation coefficient	0.94	0.88	0.83	0.9
Percentage in 68% C.I. (%)	100	100	100	100
Percentage in 95% C.I. (%)	100	100	100	100

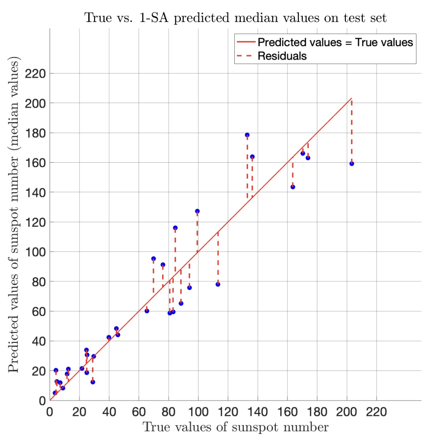
Table 10.5: Evaluation metrics for the predictions produced by each type of model (forecasting strategy) under **Warped GPR** framework, when we consider the **median** value as the point estimate. Each metric is evaluated with respect to all the points of the test set.

Metric	1-SA	6-SA	12-SA	32-SA
MAE	15.00	25.55	31.67	37.85
RMSE	19.59	28.73	38.41	42.11
RRMSE (%)	3.87	5.95	6.85	7.4
R^2	0.88	0.74	0.53	0.44
Correlation coefficient	0.94	0.88	0.83	0.87

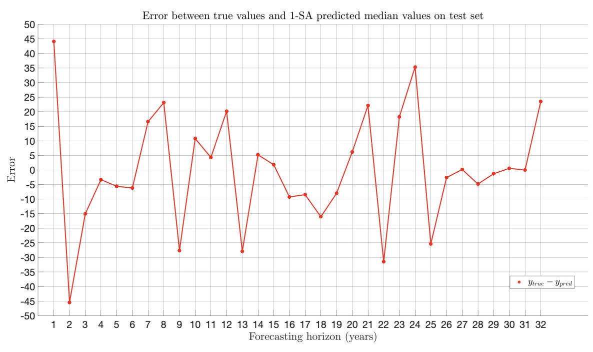
Table 10.6: Evaluation metrics for the predictions produced by each type of model (forecasting strategy) under **Warped GPR** framework, when we consider the **mean** value as the point estimate. Each metric is evaluated with respect to all the points of the test set.



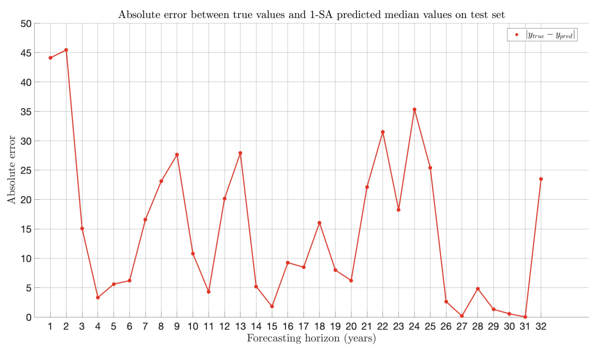
(a) Yearly mean total sunspot number predictions using 1-SA forecasting strategy.



(b) True vs. 1-SA predicted values on the test set.

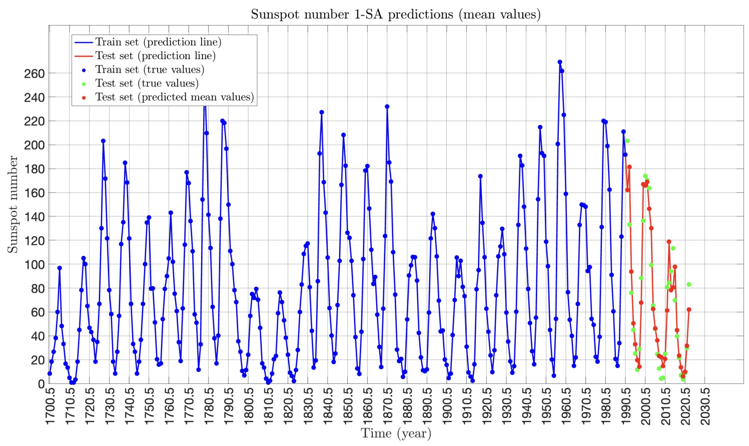


(c) Error between true and 1-SA predicted values on the test set.

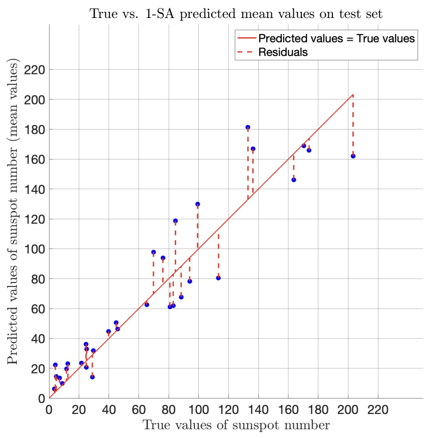


(d) Absolute error between true and 1-SA predicted values on the test set.

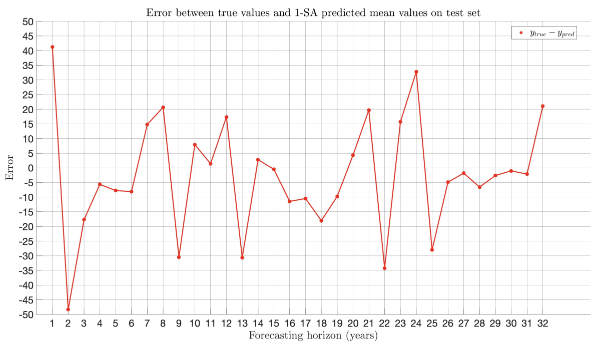
Figure 10.4: **Warped GPR**. Predicting the yearly mean total sunspot number using **1-SA** forecasting strategy. We consider the **median** value as the point estimates.



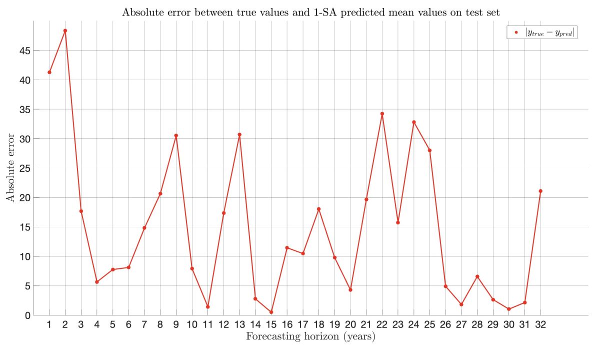
(a) Yearly mean total sunspot number predictions using 1-SA forecasting strategy.



(b) True vs. 1-SA predicted values on the test set.

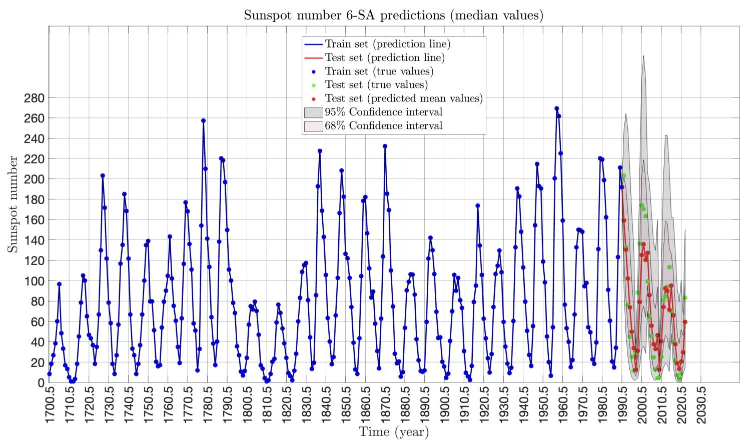


(c) Error between true and 1-SA predicted values on the test set.

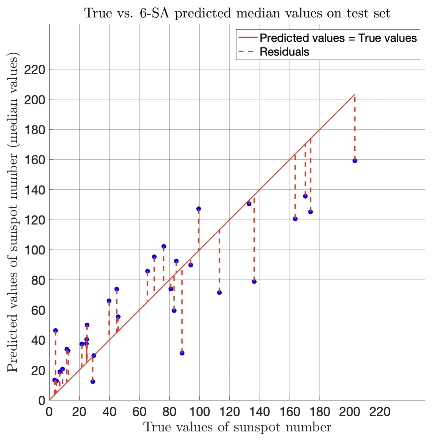


(d) Absolute error between true and 1-SA predicted values on the test set.

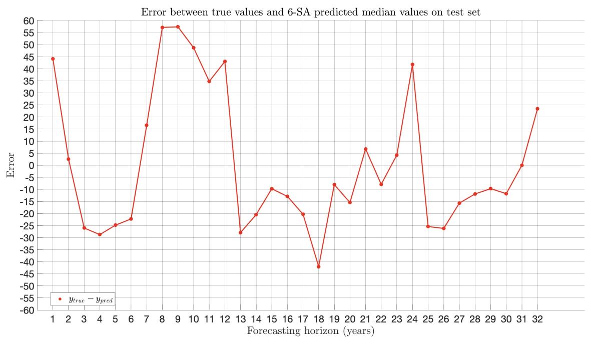
Figure 10.5: **Warped GPR**. Predicting the yearly mean total sunspot number using **1-SA** forecasting strategy. We consider the **mean** value as the point estimates.



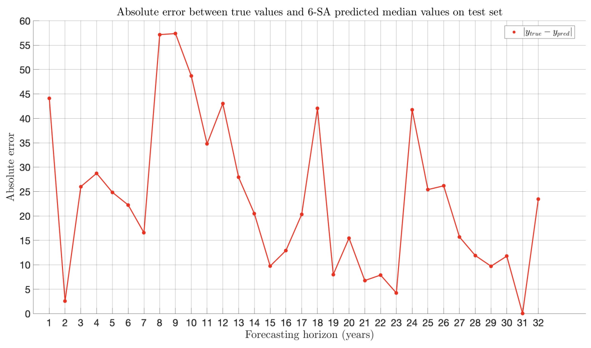
(a) Yearly mean total sunspot number predictions using 6-SA forecasting strategy.



(b) True vs. 6-SA predicted values on the test set.

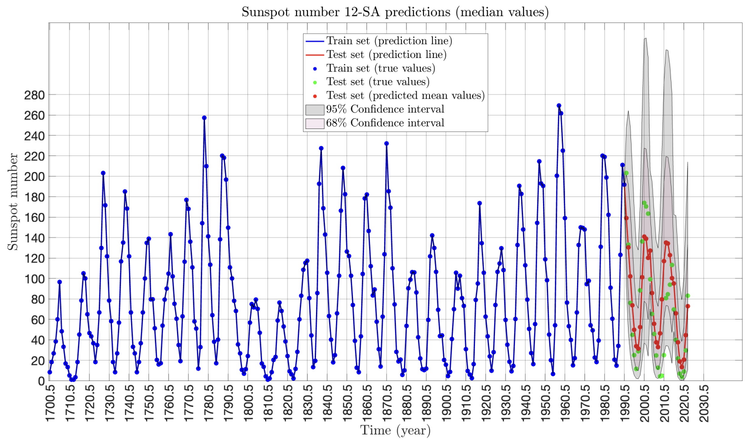


(c) Error between true and 6-SA predicted values on the test set.

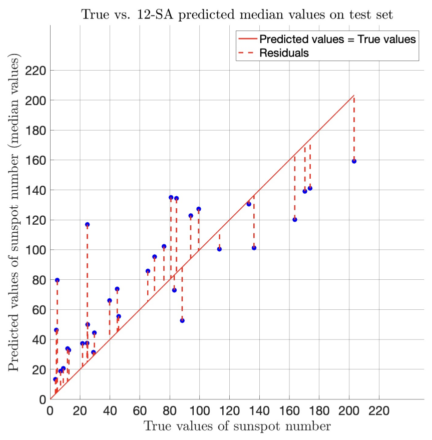


(d) Absolute error between true and 6-SA predicted values on the test set.

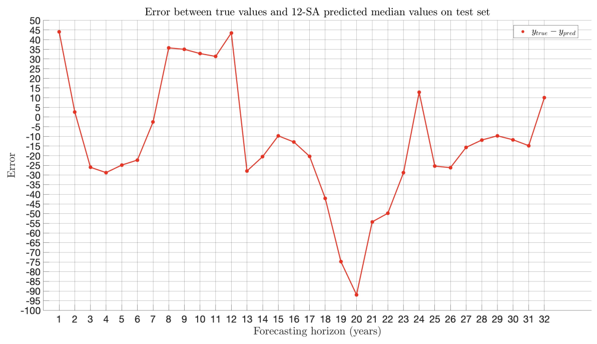
Figure 10.6: **Warped GPR**. Predicting the yearly mean total sunspot number using **6-SA** forecasting strategy. We consider the **median** value as the point estimates.



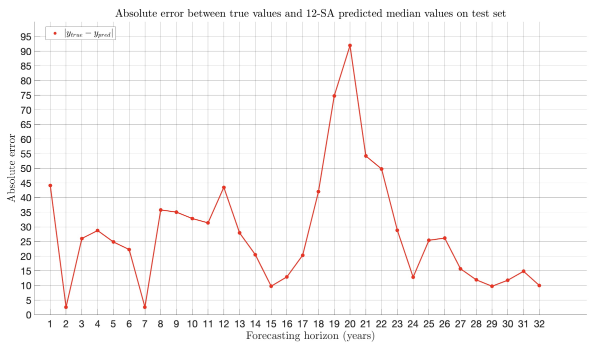
(a) Yearly mean total sunspot number predictions using 12-SA forecasting strategy.



(b) True vs. 12-SA predicted values on the test set.

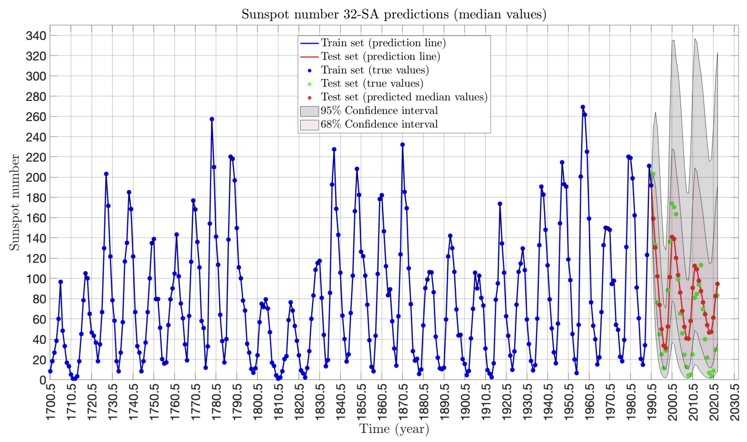


(c) Error between true and 12-SA predicted values on the test set.

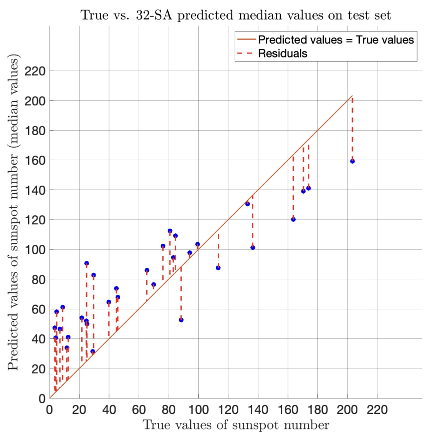


(d) Absolute error between true and 12-SA predicted values on the test set.

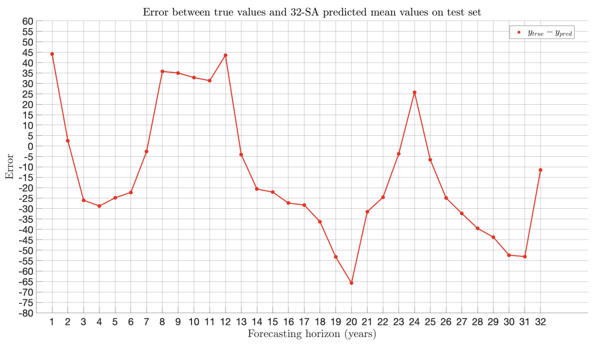
Figure 10.7: **Warped GPR**. Predicting the yearly mean total sunspot number using **12-SA** forecasting strategy. We consider the **median** value as the point estimates.



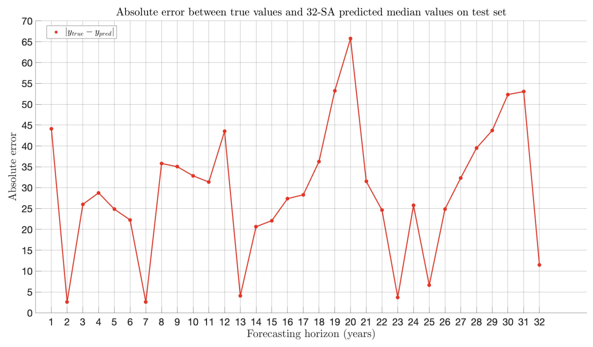
(a) Yearly mean total sunspot number predictions using 32-SA forecasting strategy.



(b) True vs. 32-SA predicted values on the test set.



(c) Error between true and 32-SA predicted values on the test set.



(d) Absolute error between true and 32-SA predicted values on the test set.

Figure 10.8: **Warped GPR**. Predicting the yearly mean total sunspot number using **32-SA** forecasting strategy. We consider the **median** value as the point estimates.

10.2 Light Gradient Boosting Machine (LightGBM)

10.2.1 Data pre-processing

Under the Light Gradient Boosting Machine (LightGBM) framework, the pre-processing comprises the two following steps:

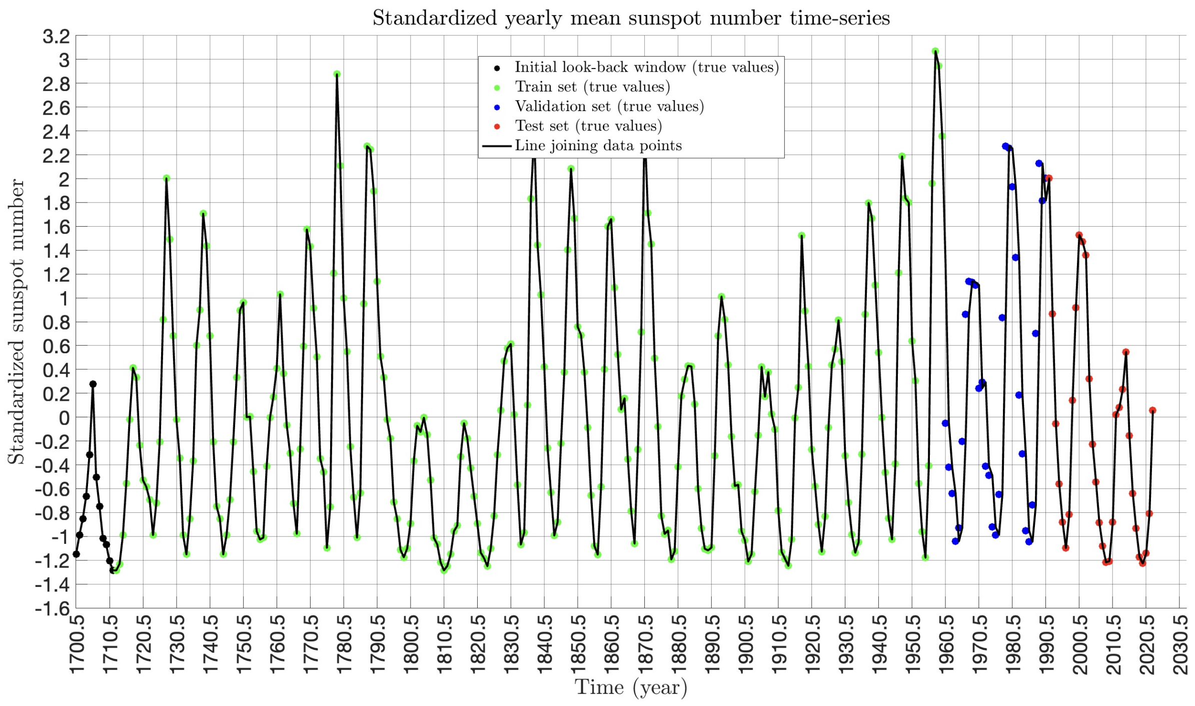
1. Standardize the time-series data. This transformation results in a zero mean and unit variance of the data.
2. Create the data set of pairs (features, target) from the raw (standardized) data of time-series. In time-series forecasting, features represent the look-back window (past observations), and the corresponding target represents the look-front window (subsequent or future observations).

Standardization is often a good practice in machine learning algorithms, and especially in gradient boosting decision trees can help in avoiding numerical instabilities, ensure the regularization effectiveness, and achieve a faster convergence. For the training phase, explained below, the train-validation-test split was $80\% - 10\% - 10\%$, respectively. Note that the choice of the look-back and look-front window plays a crucial role in the final model performance.

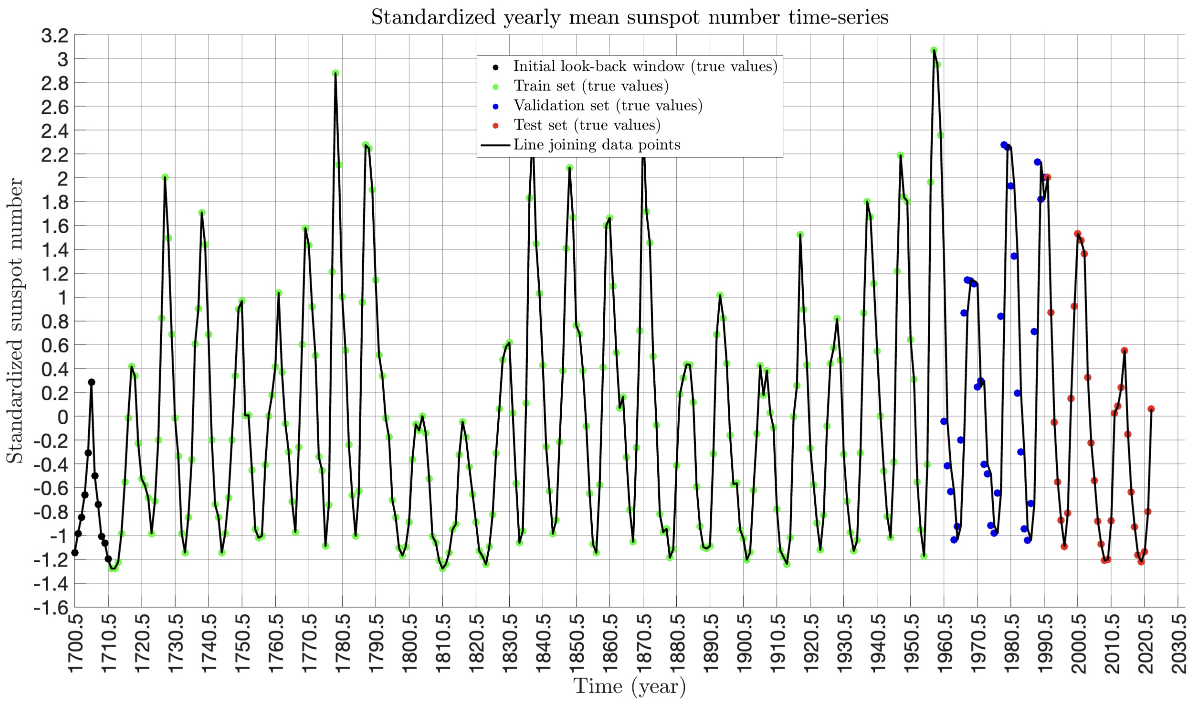
10.2.2 Model selection

Once the data set is "ready to use", we proceed to the training phase that is carried out in parallel with the hyperparameters tuning. First, let's define the hyperparameters of the LightGBM model that are searched and optimized:

- **colsample_bytree**: This hyperparameter controls the fraction of features randomly selected to build each tree during training. It helps in reducing overfitting by introducing randomness into feature selection.
- **learning_rate**: Often referred to as the "eta" parameter, it determines the step size at each iteration during gradient boosting. Lower values make the model more robust but require more iterations.
- **n_estimators**: This parameter specifies the number of boosting rounds or trees to be built. Increasing it can lead to a more complex model, but may risk overfitting if not carefully tuned.
- **num_leaves**: It defines the maximum number of leaves (or terminal nodes) in each tree. Higher values can make the model more expressive, but also increase the risk of overfitting.
- **reg_lambda**: Also known as lambda, this hyperparameter controls the L2 regularization strength on the leaf weights. It helps prevent overfitting by penalizing large weights.
- **subsample**: It determines the fraction of data randomly selected for each tree building iteration. Setting it to values less than 1.0 introduces stochasticity and can improve model generalization.



(a) Standardized yearly mean total sunspot number time-series used for developing 1-SA predictive model.



(b) Standardized yearly mean total sunspot number time-series used for developing 6-SA predictive model.

Figure 10.9: Transformed yearly mean total sunspot number time-series split in train-validate-test sets for usage in **LightGBM** forecasting model development. Subfigure (a) corresponds to the 1-SA forecasting method, and subfigure (b) corresponds to the 6-SA forecasting method. The standardization is done with respect to the train set, which has a mean of 1 and a standard deviation of 1.

- **max_depth**: This hyperparameter defines the maximum depth of each tree in the ensemble. A deeper tree can capture complex relationships in the data, but may also lead to overfitting. Setting an appropriate value for "max_depth" is crucial to balance model complexity and generalization (value of -1 means in general that there is no maximum depth).

The concept of the training phase is to find the optimal set of hyperparameters for the LightGBM model, and train the corresponding model on our sunspot data. For that purpose, one iteration of our "search process" contains the following steps:

1. Select a value for the hyperparameters.
2. Train a LightGBM model (or more than one models, see "direct multi-step ahead forecast" strategy in Equation (3.4)) on the training set.
3. Calculate the value of the loss function on the validation set ("validation loss").

After searching for various values for the hyperparameters, we select the model that produced the smallest validation loss, and that is the optimal model. The "search process" is getting done using Sequential Model Based Optimization (SMBO) [15]. SMBO is a Bayesian optimization technique that uses information from past trials to inform the next set of hyperparameters to explore. The approximation of the user-defined objective function is being done with a Gaussian process. After each run of hyperparameters on the objective function, the algorithm makes an educated guess which set of hyperparameters is most likely to improve the score and should be tried in the next run. This process is being repeated more than one times to ensure the optimal results. The loss function that we use is the RMSE.

We build two models, one for 1-step ahead predictions (1-SA), and one for 6-steps ahead predictions (6-SA). The corresponding forecasting strategies are the "direct one-step ahead" method (see Equation (3.1)), and the "direct multi-step ahead forecast" method (see Equation (3.4)), respectively. In Table 10.7 and Table 10.8, the optimal set of hyperparameters is shown for 1-SA model and 6-SA model, respectively. The look-back window in the 1-SA model is of 12 years length, and the look-front window is of 1 year length. The look-back window in the 1-SA model is of 11 years length, and the look-front window is of 6 years length. The parameter "look-back" for both models is being tuned with trial-and-error.

10.2.3 Model evaluation

Now, we will visualize the trained models in terms of their forecasting capabilities in both scenarios, where the forecasting horizon is 1 year, and 6 years. See Figure 10.10 and Figure 10.11, respectively. In Table 10.9 we can see various evaluation metrics for the predictions made by each model (1-SA, and 6-SA) on the test set.

As expected, the prediction accuracy decreases as the forecasting horizon gets larger. It means that, when trying to predict sunspot number for more years ahead, we get larger discrepancies from the true values. It is worth noting that RRMSE(%), in both forecasting strategies, is less than 10, which indicates a good fit of the model and a competitive accuracy.

[15]We used `scikit-optimize` for implementing Sequential model-based optimization in Python

Parameter	1	Bounds
colsample_bytree	0.80	$(0.7, 1)$
learning_rate	0.5	$(0.01, 1)$
n_estimators	25	$(10, 1000)$
num_leaves	654	$(10, 1000)$
reg_lambda	7.21e-3	$(0.001, 1)$
subsample	0.82	$(0.7, 1)$
max_depth	-1	$(-1, -1)$
Optimal Val Loss	18.78	

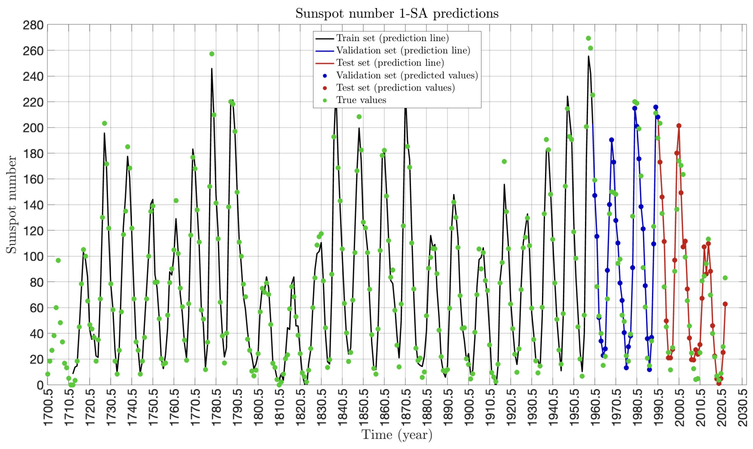
Table 10.7: **LightGBM 1-SA**. The optimal set of model's hyperparameters (second column), the bounds that were set during the optimization (third column), and the optimal value of the loss function calculated on the validation set (bottom row).

Parameter	1	2	3	4	5	6	Bounds
colsample_bytree	0.76	0.78	0.82	0.85	0.74	0.97	$(0.7, 1)$
learning_rate	0.69	0.51	0.53	0.34	0.35	0.44	$(0.01, 1)$
n_estimators	16	10	10	10	106	559	$(10, 1000)$
num_leaves	993	10	10	986	674	1000	$(10, 1000)$
reg_lambda	0.88	0.20	1e-3	0.27	0.81	0.84	$(0.001, 1)$
subsample	0.71	0.76	0.71	0.98	0.82	0.92	$(0.7, 1)$
max_depth	-1	-1	-1	-1	-1	-1	$(-1, -1)$
Optimal Val Loss	30.97						

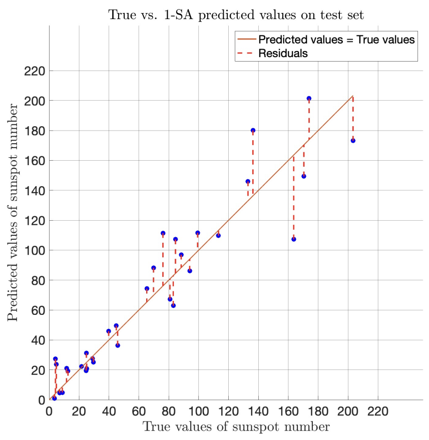
Table 10.8: **LightGBM 6-SA**. The optimal set of model's hyperparameters (second to seventh column), the bounds that were set during the optimization (eighth column), and the optimal value of the loss function calculated on the validation set (bottom row).

Metric	1-SA	6-SA
MAE	14.17	22.05
RMSE	19.17	27.13
RRMSE (%)	3.78	5.27
R^2	0.88	0.77
Correlation coefficient	0.94	0.90

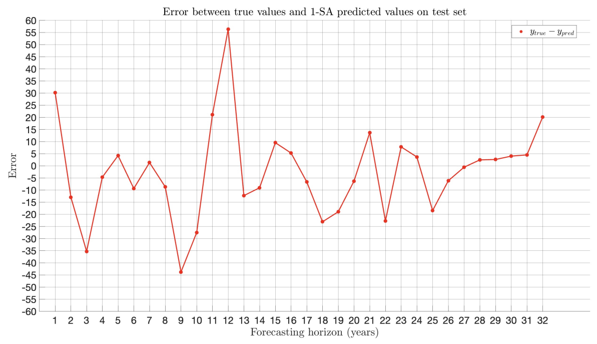
Table 10.9: Evaluation metrics for the predictions produced by each type of model (forecasting strategy) under **LightGBM** framework. Each metric is evaluated with respect to all the points of the test set.



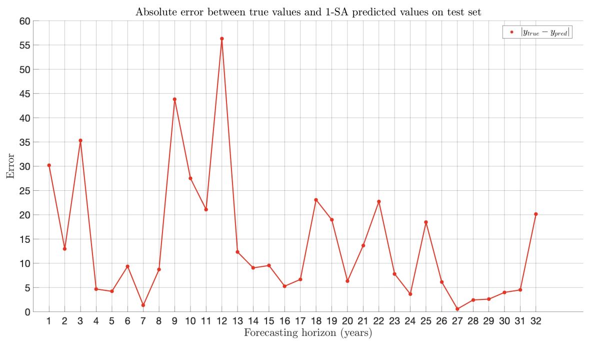
(a) Yearly mean total sunspot number predictions using 1-SA forecasting strategy.



(b) True vs. 1-SA predicted values on the test set.

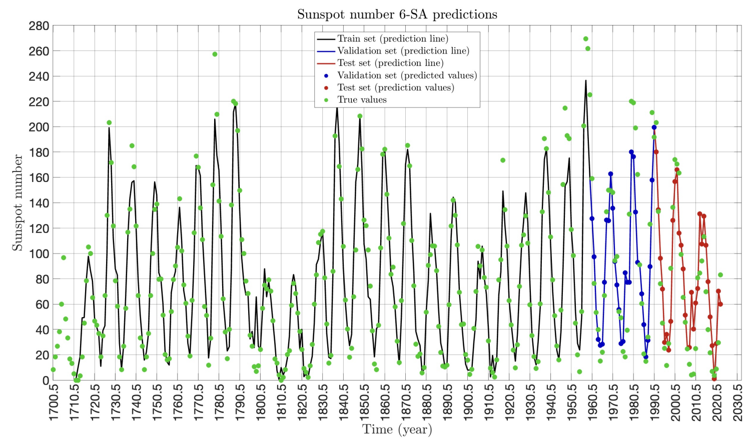


(c) Error between true and 1-SA predicted values on the test set.

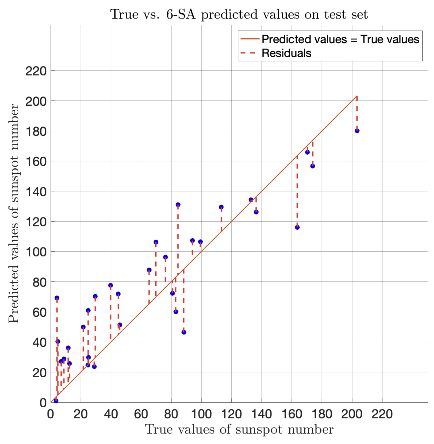


(d) Absolute error between true and 1-SA predicted values on the test set.

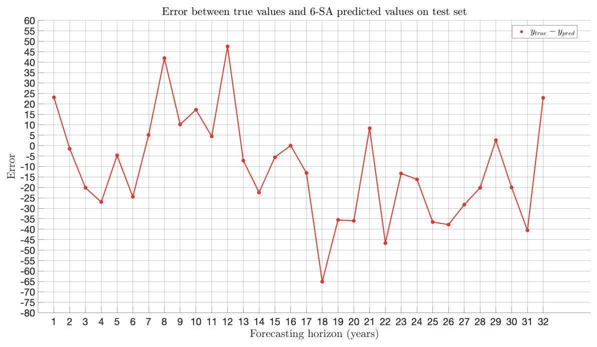
Figure 10.10: **LightGBM 1-SA**. Predicting the yearly mean total sunspot number using 1-SA forecasting strategy. We consider the median value as the point estimates.



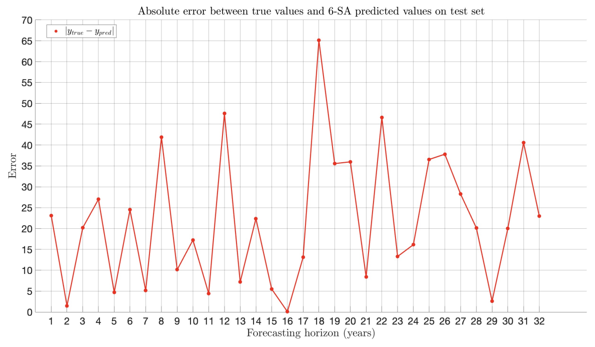
(a) Yearly mean total sunspot number predictions using 6-SA forecasting strategy.



(b) True vs. 6-SA predicted values on the test set.



(c) Error between true and 6-SA predicted values on the test set.



(d) Absolute error between true and 6-SA predicted values on the test set.

Figure 10.11: **LightGBM 6-SA**. Predicting the yearly mean total sunspot number using 6-SA forecasting strategy. We consider the median value as the point estimates.

10.3 Long Short-Term Memory (LSTM)

10.3.1 Data pre-processing

Under the Long Short-Term Memory (LSTM) framework, the pre-processing comprises the two following steps:

1. Standardize the time-series data. This transformation results in a zero mean and unit variance of the data.

2. Create the data set of pairs (features, target) from the raw (standardized) data of time-series. In time-series forecasting, features represent the look-back window (past observations), and the corresponding target represents the look-front window (subsequent or future observations).

Standardization is often a good practice in machine learning algorithms, and especially in LSTM can help in avoiding numerical instabilities, and achieve a faster convergence. For the training phase, explained below, the train-validation-test split was $80\% - 10\% - 10\%$, respectively. As far as the creation of the data set is concerned, the choice of the look-back and look-front window play a pivotal role in the final model's characteristics and performance.

10.3.2 Model selection

Once the data set is "ready to use", we proceed to the training phase that is carried out in parallel with the hyperparameters tuning. We use a (stacked) LSTM model in sequence with a Linear model (dense layer). First, let's define the hyperparameters of the whole structure (see Figure 7.9) that are searched and optimized:

- **input_size**: The number of expected features in the input data. As features here, we mean the dimensionality of each input $\mathbf{x}$ at a time step in the LSTM unit. In our case, it is equal to 1, as at each time step we only have one observation, which is the number of sunspots.

- **hidden_size**: The number of features in the hidden state (output).

- **num_layers**: The number of recurrent layers, i.e. stacked LSTM layers.

- **dropout**: If non-zero, introduces a dropout layer on the outputs of each LSTM layer except the last one, with dropout probability equal to this value.

- **in_features**: The number of input features (input dimensions). It is equal to the "hidden_size".

- **out_features**: The number of output features (output dimensions). In our case, it is equal to 1, as each model predicts only one value.

The concept of the training phase is to find the optimal set of hyperparameters for the LSTM model, and train the corresponding model on our sunspot data. For that purpose, one iteration of our "search process" contains the following steps:

1. Select a value for the hyperparameters.
2. Train an LSTM model (or more than one models, see "direct multi-step ahead forecast" strategy in Equation (3.4)) on the training set.
3. Calculate the value of the loss function on the validation set ("validation loss").

The learning rate is set to 10e-5 after experimenting with various values. The look-back window for each trained model is also set appropriately by experimenting with various values. After these two are fixed, we proceed with hyperparameter tuning. After searching for various values for the hyperparameters, we select the model that produced the smallest validation loss, and that is the optimal model. The "search process" is getting done using a simple grid search. The loss function that we use is the RMSE. Note that during training, we apply "early stopping" by monitoring the validation loss; when it stops improving (we have set some "patience" epochs), we stop the training process.

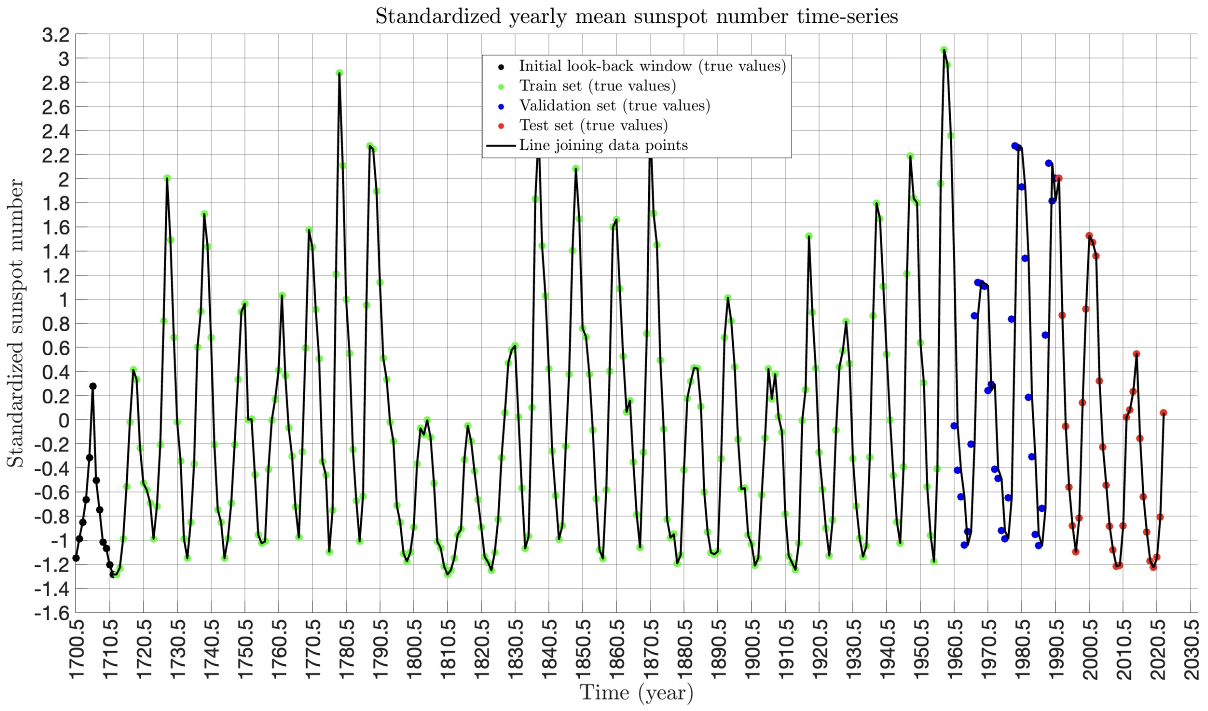
We build two models [16], one for 1-step ahead predictions (1-SA), and one for 6-steps ahead predictions (6-SA). The corresponding forecasting strategies are the "direct one-step ahead" method (see Equation (3.1)), and the "direct multi-step ahead forecast" method (see Equation (3.4)), respectively. In Table 10.10 and Table 10.11, the optimal set of hyperparameters is shown for 1-SA model and 6-SA model, respectively. The look-back window in the 1-SA model is of 12 years length, and the look-front window is of 1 year length. The look-back window in the 1-SA model is of 22 years length, and the look-front window is of 6 years length.

10.3.3 Model evaluation

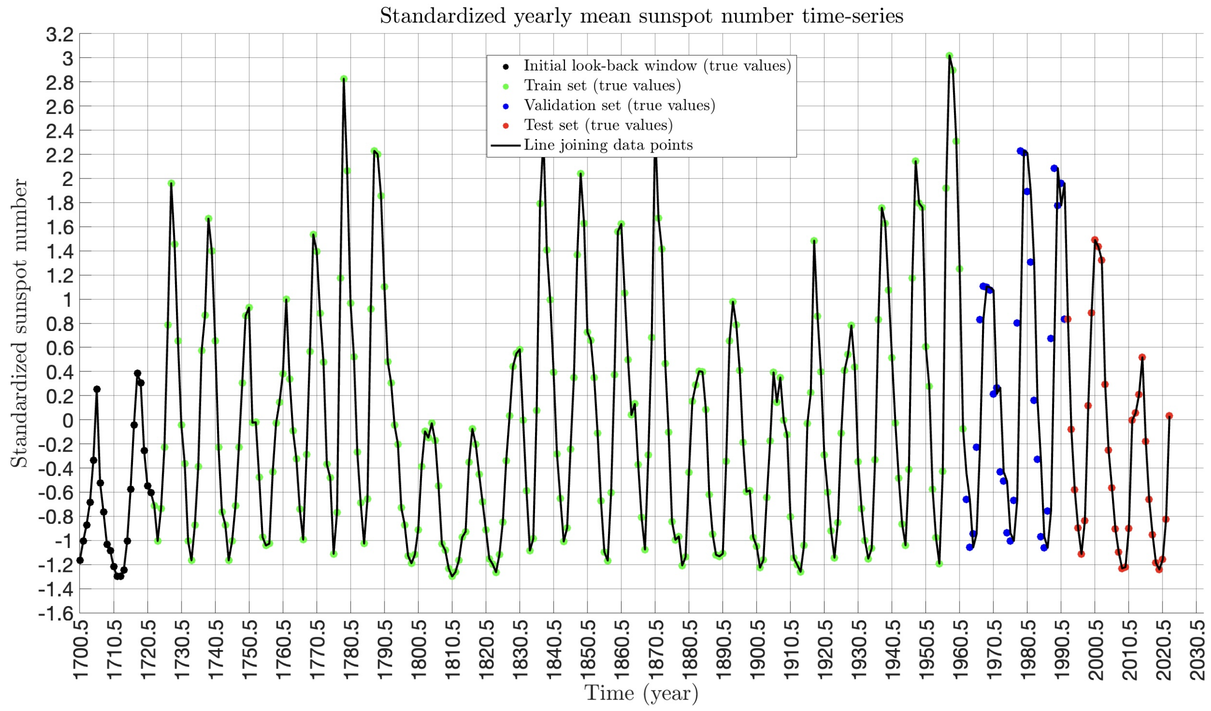
Now, we will visualize the trained models in terms of their forecasting capabilities in both scenarios, where the forecasting horizon is 1 year, and 6 years. See Figure 10.13 and Figure 10.14, respectively. In Table 10.12 we can see various evaluation metrics for the predictions made by each model (1-SA, and 6-SA) on the test set.

As expected, the prediction accuracy decreases as the forecasting horizon gets larger. It means that, when trying to predict sunspot number for more years ahead, we get larger discrepancies from the true values. It is worth noting that RRMSE(%), in both forecasting strategies, is less than 10, which indicates a good fit of the model and a competitive accuracy.

[16]We used PyTorch Lightning as the deep learning framework and especially the modules `torch.nn.LSTM` and `torch.nn.Linear`. This framework uses Tensors and is capable of accelerating the training process using special GPUs (hardware needed).



(a) Standardized yearly mean total sunspot number time-series used for developing 1-SA predictive model.



(b) Standardized yearly mean total sunspot number time-series used for developing 6-SA predictive model.

Figure 10.12: Transformed yearly mean sunspot number time-series split in train-validate-test sets for usage in **LSTM** forecasting model development. Subfigure (a) corresponds to the 1-SA forecasting method, and subfigure (b) corresponds to the 6-SA forecasting method. The standardization is done with respect to the train set, which has a mean of 1 and a standard deviation of 1.

Parameter	1	Grid
input_size	1	–
hidden_size	120	(60 : 20 : 300)
num_layers	2	(1 : 1 : 3)
drop_out	0	(0.1 : 0.1 : 0.2)
in_features	120	–
out_features	1	–
Optimal Val Loss	19.07	

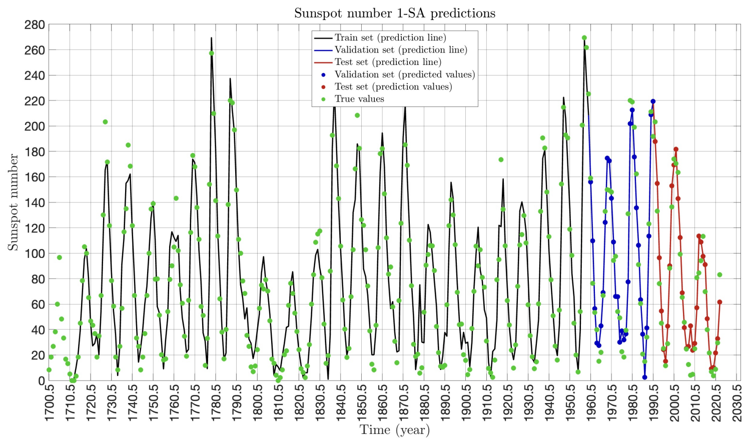
Table 10.10: **LSTM 1-SA**. Optimal set of model's hyperparameters, the bounds that were set during the optimization, and the optimal value of the loss function calculated on the validation set.

Parameter	1	2	3	4	5	6	Grid
input_size	1	1	1	1	1	1	–
hidden_size	200	150	150	200	300	200	(50 : 50 : 400)
num_layers	2	1	1	1	1	1	(1 : 1 : 3)
drop_out	0	0	0	0	0.2	0	(0.1 : 0.1 : 0.2)
in_features	200	150	150	200	300	200	–
out_features	1	1	1	1	1	1	–
Optimal Val Loss	31.62						

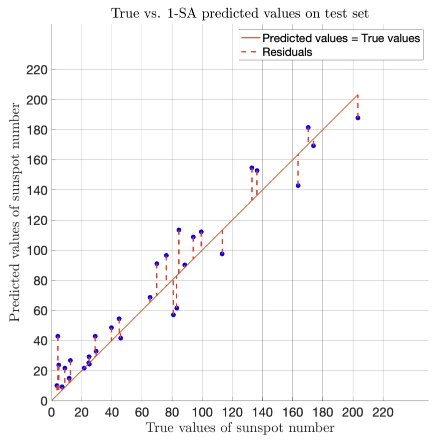
Table 10.11: **LSTM 6-SA**. Optimal set of model's hyperparameters, the bounds that were set during the optimization, and the optimal value of the loss function calculated on the validation set.

Metric	1-SA	6-SA
MAE	12.41	23.50
RMSE	15.44	31.97
RRMSE (%)	3.00	6.17
R^2	0.93	0.62
Correlation coefficient	0.97	0.85

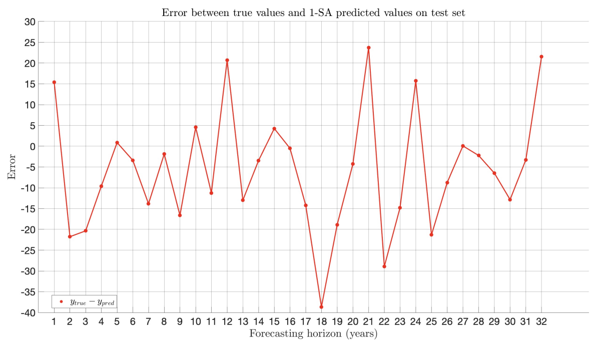
Table 10.12: Evaluation metrics for the predictions produced by each type of model (forecasting strategy) under **LSTM** framework. Each metric is evaluated with respect to all the points of the test set.



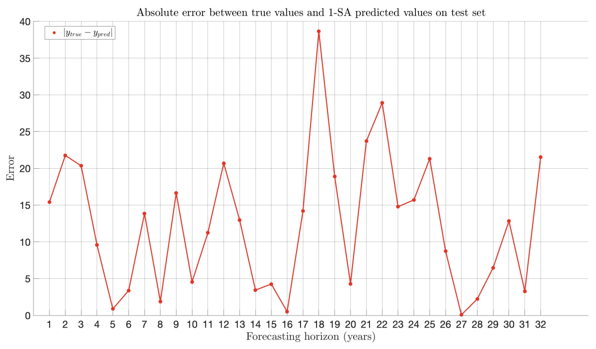
(a) Yearly mean total sunspot number predictions using 1-SA forecasting strategy.



(b) True vs. 1-SA predicted values on the test set.

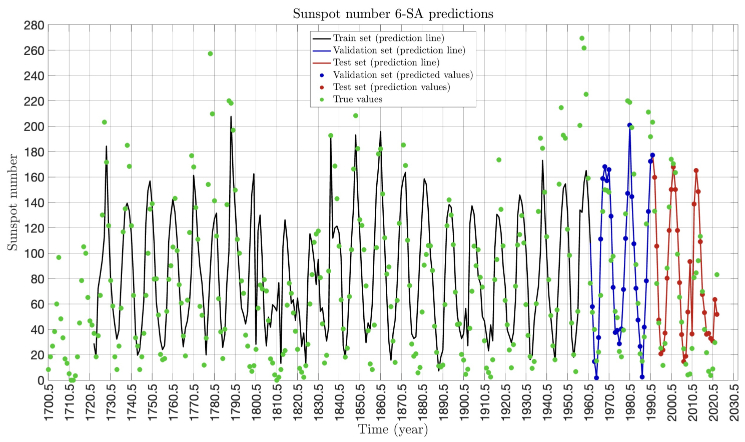


(c) Error between true and 1-SA predicted values on the test set.

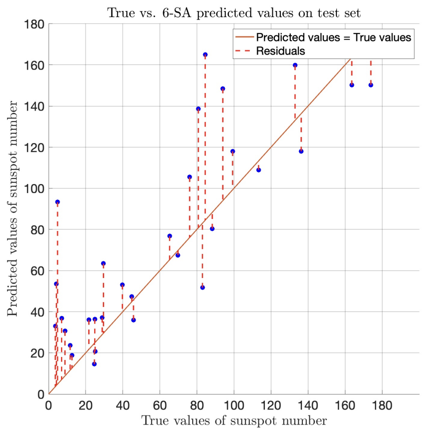


(d) Absolute error between true and 1-SA predicted values on the test set.

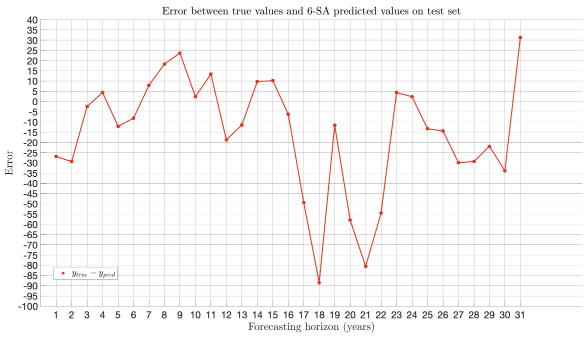
Figure 10.13: **LSTM**. Predicting the yearly mean total sunspot number using **1-SA** forecasting strategy. We consider the median value as the point estimates.



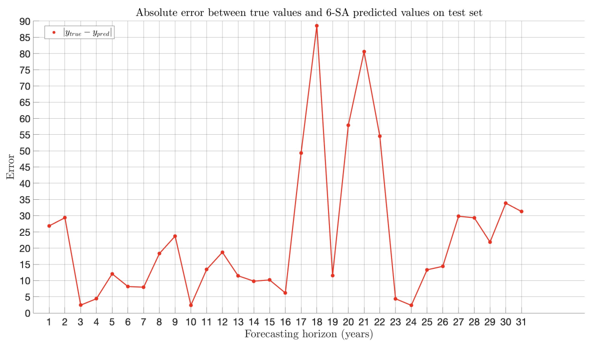
(a) Yearly mean total sunspot number predictions using 6-SA forecasting strategy.



(b) True vs. 6-SA predicted values on the test set.



(c) Error between true and 6-SA predicted values on the test set.



(d) Absolute error between true and 6-SA predicted values on the test set.

Figure 10.14: **LSTM**. Predicting the yearly mean total sunspot number using **6-SA** forecasting strategy. We consider the median value as the point estimates.

10.4 Findings

In the above subsections, we applied three supervised machine learning methods on yearly sunspots data with the purpose to model them and be able to make accurate forecasts. These models revealed unique strengths and weaknesses, each contributing to our understanding of time-series forecasting in its own way. Below, we provide a comparative analysis of these three machine learning models with respect to the yearly sunspot number forecasting:

For **Gaussian Process Regression (GPR)** we note the following:

- **Strengths**:
  - GPR is a powerful model for capturing complex relationships within time-series data, especially when dealing with non-linear patterns.
  - It provides probabilistic predictions, offering not only point forecasts but also uncertainty estimates, which can be valuable for decision-making and risk assessment.
  - GPR is highly interpretable, allowing us to understand the influence of each input feature on the predictions.
- **Weaknesses**:
  - Computationally intensive, GPR can be slow for large datasets, making it less practical in real-time forecasting scenarios.
  - It may not perform as well when dealing with high-dimensional input data or when assumptions of Gaussianity are violated.

For **Gradient Boosting Decision Trees (GBDTs)** we note the following:

- **Strengths**:
  - Gradient Boosting Decision Trees, represented by models like XGBoost or LightGBM, excel at capturing complex relationships and interactions in the data.
  - They are computationally efficient and can handle large datasets with many features.
  - Feature importance analysis is straightforward, aiding in variable selection and model interpretation.
- **Weaknesses**:
  - They may struggle to capture long-term dependencies and intricate temporal patterns in time-series data, especially when faced with irregularly spaced observations.
  - These models typically do not provide probabilistic forecasts, which limits the quantification of uncertainty.

For **Long Short-Term Memory (LSTM)** we note the following:

- **Strengths**:
  - LSTMs are designed to model sequences and are well-suited for time-series data, capable of capturing long-range dependencies.
  - They excel when dealing with irregularly spaced data or missing values, making them versatile for real-world applications.
  - LSTMs can provide multi-step ahead forecasts, which are essential in many forecasting scenarios.
- **Weaknesses**:
  - Training LSTMs can be computationally expensive, and they may require substantial amounts of data to generalize well.
  - Interpreting the inner workings of LSTMs can be challenging, which limits their transparency and interpretability.

For our sunspot forecasting task, we observed that LSTM outperformed the other models in capturing the intricate patterns and long-range dependencies inherent in sunspot data when the objective was to forecast 1 step ahead. When we tried to forecast 6 steps ahead, the LightGBM revealed a better performance. Also, GPR produced competitive predictions, along with the advantage that its training is independent of the forecasting horizon. Once we have tuned the model, we can make predictions as deeply in the future as we desire. However, the choice of model should always be guided by the specific needs and constraints of the forecasting problem at hand. We realized, also, that in general, predictions with a deeper forecasting horizon have larger uncertainty and come with greater errors. In general, the predictions of all models gave RRMSE less than 10%, which indicates a very good fit. Finally, as far as computational complexity is concerned, LightGBM is the most efficient algorithm, whereas LSTM can become significantly costly. GPR is performing well, given that the dataset of the yearly sunspots is not large and the matrix manipulations are not too intensive.

10.5 Forecasting the peak of Solar Cycle 25

Using our proposed aforementioned machine learning models, we can predict the upcoming peak of the Solar Cycle 25. We have:

- 112 sunspots in 2024 (LightGBM 6-SA)
- 118 sunspots in 2024 (Warped GPR using median value)
- 122 sunspots in 2024 (Warped GPR using mean value)
- 131 sunspots in 2024 (LSTM 6-SA)
- 115 sunspots in 2025 (NASA)

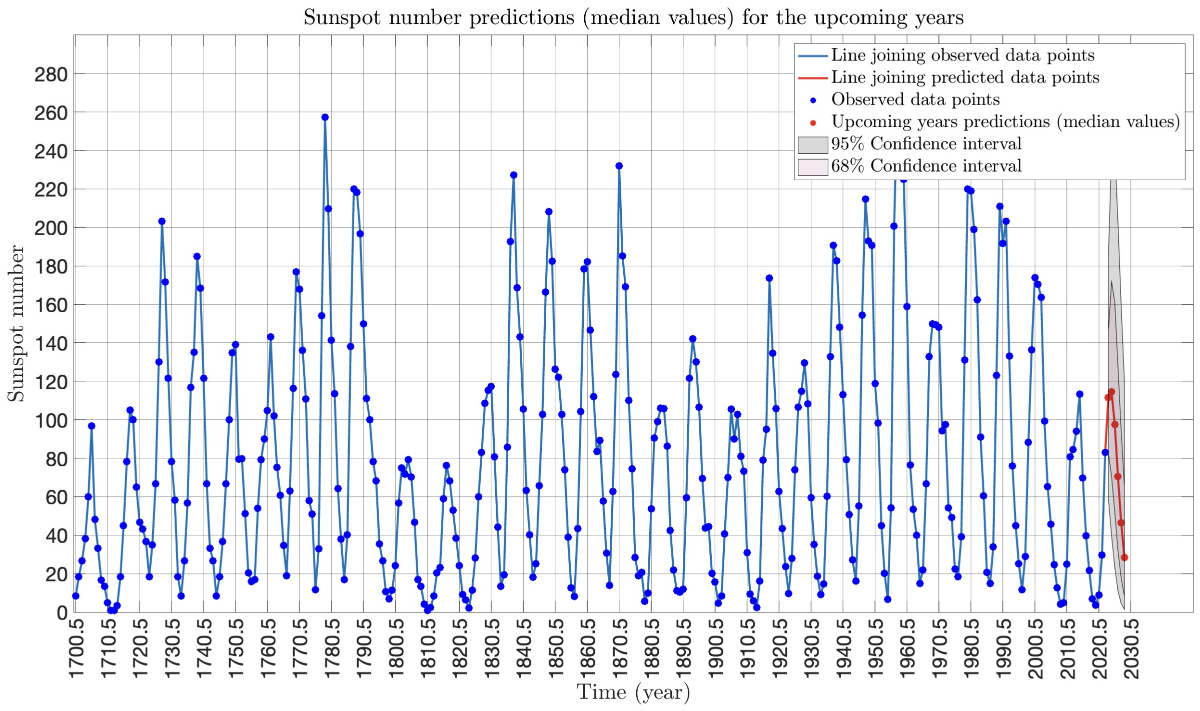


Figure 10.15: **Warped GPR using median value**. Forecast of the next 6 years of the time-series of the yearly mean total sunspot number. The predictions correspond to the years 2023-2028.

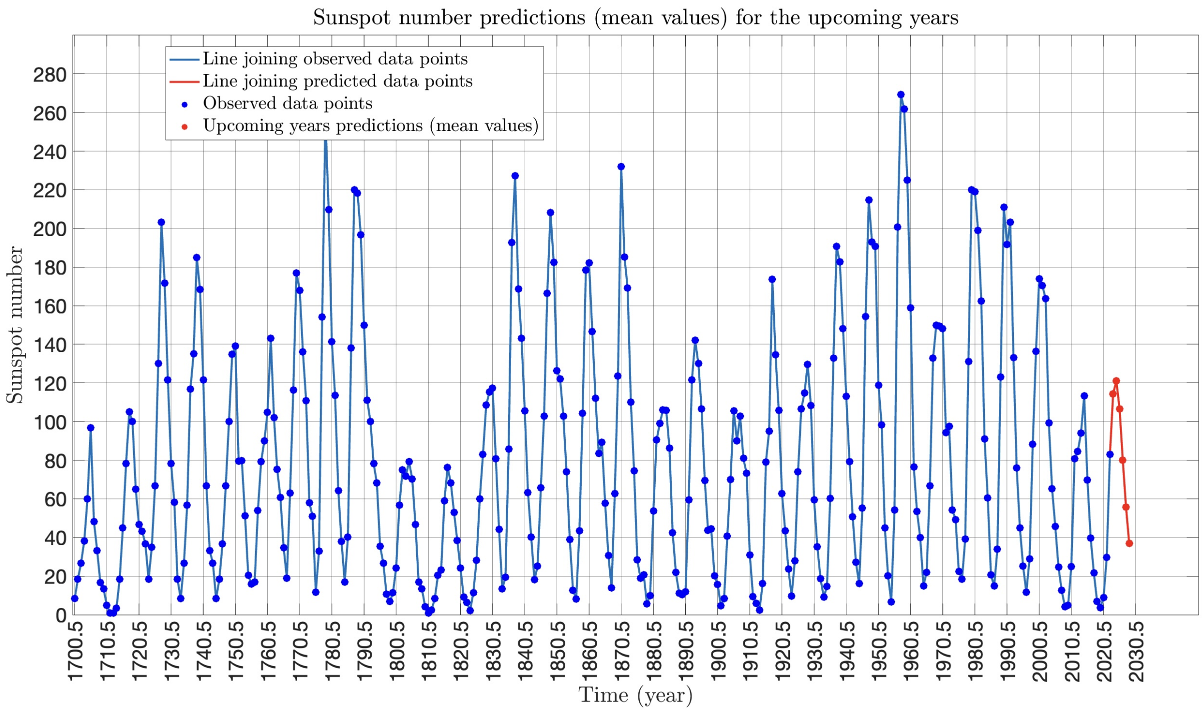


Figure 10.16: **Warped GPR using mean value**. Forecast of the next 6 years of the time-series of the yearly mean total sunspot number. The predictions correspond to the years 2023-2028.

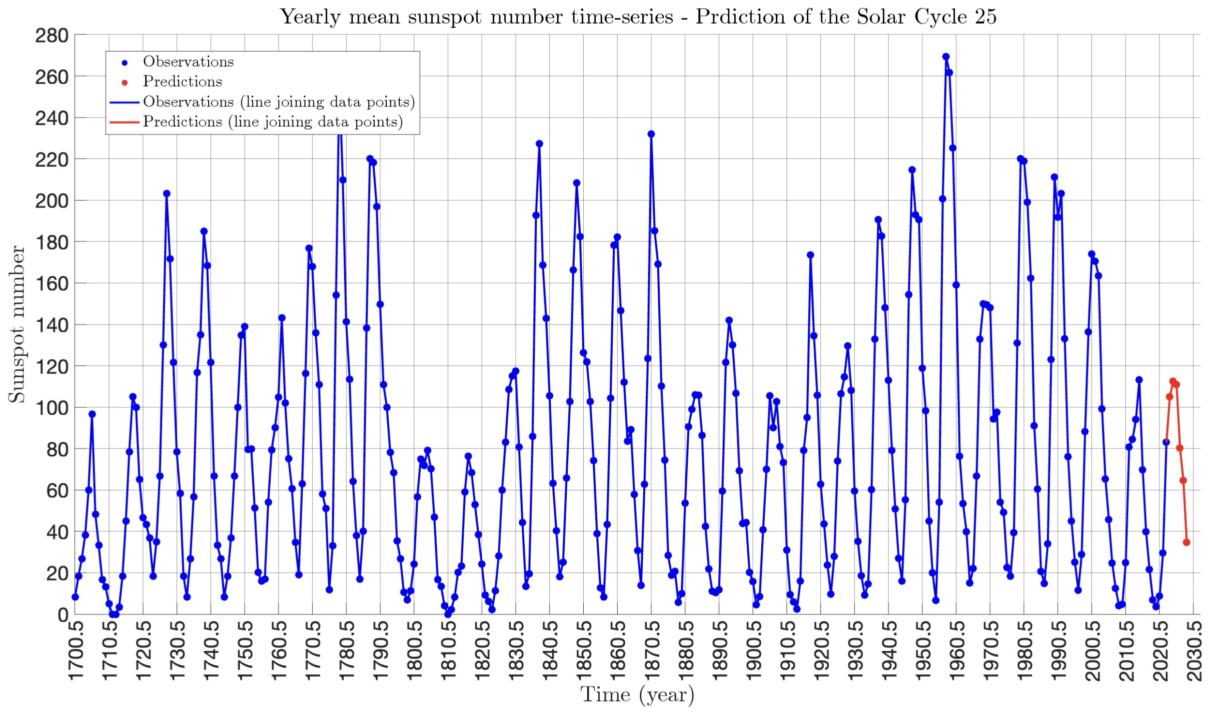


Figure 10.17: **LightGBM 6-SA**. Forecast of the next 6 years of the time-series of the yearly mean total sunspot number. The predictions correspond to the years 2023-2028.

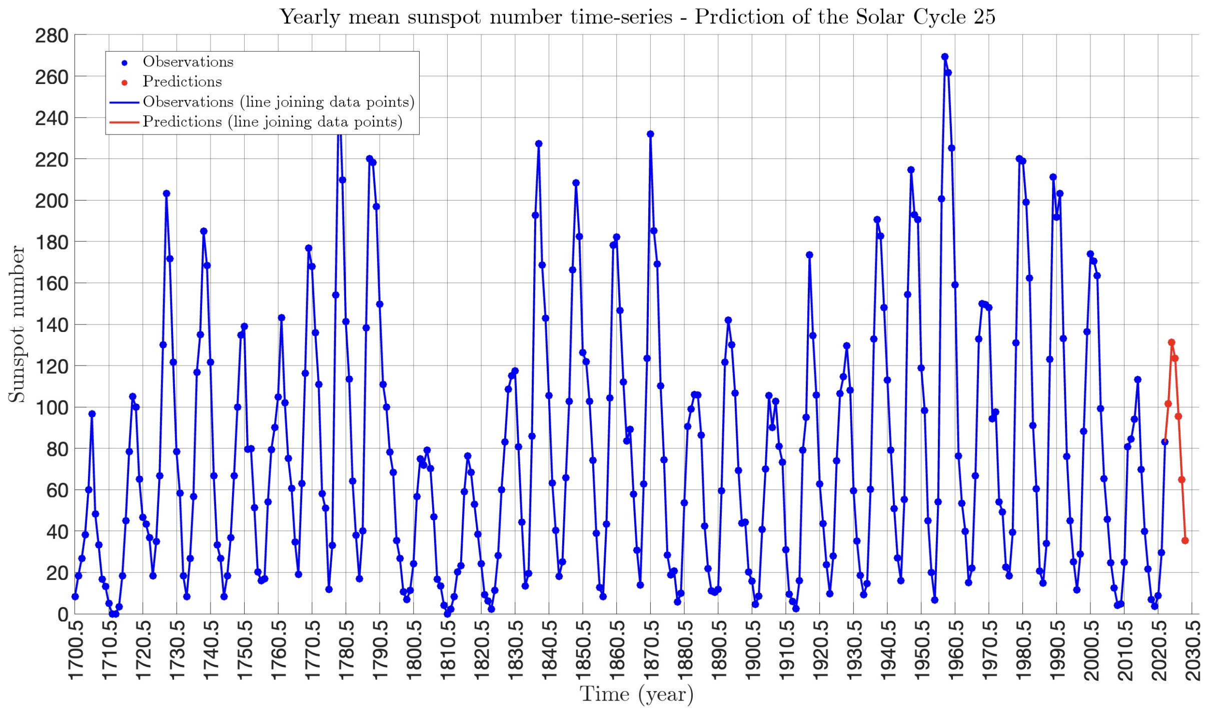


Figure 10.18: **LSTM 6-SA**. Forecast of the next 6 years of the time-series of the yearly mean total sunspot number. The predictions correspond to the years 2023-2028.

10.6 Related work

Below, we refer to some related work on forecasting the time-series of the sunspots number (monthly and yearly):

- Werner *et al.* [99] proposes an autoregressive AR(9) model capable of predicting the yearly sunspot number. The forecast quality is being estimated by the sum of the square residuals and by the mean standard deviation. For forecasting horizons 1, 2, 3, and 4, they obtain mean standard deviations of 14, 23, 29, and 32, respectively. For greater forecasting horizons, it is said that the aforementioned metric gets greater than 30.

- Gonçalves *et al.* [100] uses a warped Gaussian process regression framework in order to make predictions on the yearly number of sunspots. Tests using holdout data yielded a root mean squared error of 10 within 5 years and 25–35 within 10 years.

- Covas *et al.* [101] forecasts the sunspot butterfly diagram using feed-forward neural networks. In that way, it attempts to demonstrate that forecasting of the Sun's sunspot time-series can be extended to the spatial-temporal case. In that way, it predicts that the upcoming Cycle 25 maximum sunspot number will be around 57 ± 17. That implies a very weak cycle and that it will be the weakest cycle on record.

- Upton *et al.* [102] used a flux transport model and predicted that Solar Cycle 25 would be similar in size to Solar Cycle 24 with a 15% uncertainty. It refers that Cycle 25 will be slightly weaker than Cycle 24, making it the weakest cycle on record in the last hundred years.

- Labonville *et al.* [103] used a dynamo-based model to forecast the upcoming solar cycle and predicted a maximum sunspot number of $89+29/-15$.

- Pala *et al.* [104] used two layers of stacked LSTMs and predicted that the upcoming Solar Cycle 25 would have a maximum sunspot number of 167.3 with the peak being reached in 2023.2 ± 1.1. The trained LSTM gives a RMSE of 36 for a forecasting horizon of 5 years.

- Okoh *et al.* [105] used a method called the Hybrid Regression Neural Network, which combines regression analysis and neural network learning to estimate SSN. They predicted that the end of Solar Cycle 24 would be in March 2020 (± seven months) with a SSN of 5.4 and the maximum of Solar Cycle 25 would be January 2025 (± six months) with a SSN of 122.1 (±18.2).

- Rigozo *et al.* [106] carried out a study to estimate the strength of solar activity in both Solar Cycle 24 and 25 based on extrapolation of spectral components. They estimated that the maximum number of sunspots in Cycle 25 will occur in April 2023 with a sunspot number of 132.1 (with a solar-cycle length of 118 months or 9.8 years). In the same study, Solar Cycle 24 was also estimated, and the maximum SSN value was projected to be 113.3 in November 2013. However, when the actual SSN values were examined, the maximum SSN value was reached in February 2014 with a SSN value of 146.1. Thus, it can be said that Rigozo *et al.* [106] reached their maximum SSN estimation with an about 30% deviation from the actual SSN value, so the estimation methods are good methods.

11 Conclusion and Future Work

In this diploma thesis, we embarked on a journey through the world of time-series analysis using various machine learning methods. We covered fundamental concepts in time-series analysis, delved into forecasting strategies, explored non-linear data transformations, and applied three distinct supervised machine learning models to forecast the yearly mean sunspot number. The choice of machine learning model for time-series forecasting depends on the specific characteristics of the data and the forecasting objectives. Gaussian Process Regression (GPR) offers interpretability and probabilistic forecasts, while Gradient Boosting Decision Trees (GBDTs) are efficient and perform well with complex, high-dimensional data. Long Short-Term Memory (LSTM) neural networks, on the other hand, shine in capturing long-term dependencies and handling irregular data, but they require significant computational resources.

About the forecasting of the yearly mean total sunspot number, we found that the aforementioned machine learning algorithms, with the appropriate data transformations and selection of hyperparameters, can be used to model accurately time-series data. In particular, they give competitive predictions of the yearly sunspot number, one of them can provide uncertainty estimates, and forecast that the upcoming peak will occur in 2024. This estimate agrees with recent articles which propose earlier appearance of the peak than the existing forecasts of NASA and NOAA that have predicted occurrence of the peak in 2025 or later. Regarding the predictive performance of our methods compared to the related work mentioned before, we note that similar GPR implementation gives a RMSE of 10 within 5 years and 25–35 within 10 years, while our GPR give 28 for 6 years ahead and 34 for 12 years ahead. Furthermore, we stated that an AR(9) model for forecasting horizons 1, 2, 3, and 4, produced mean standard deviations of 14, 23, 29, and 32, respectively. Another LSTM implementation in literature gave an RMSE of 36 for a forecasting horizon of 5 years, while our stacked LSTM gave 32. However, it is worth noting that in this work, the literature LSTM was built based on monthly sunspot data (as the majority of scientific works on sunspots), so the comparison cannot be done in an absolute way.

With respect to future work, first of all, it is worth testing the same models on the time-series of the 13-month smoothed monthly total sunspot number and analyzing the results. In addition, further research could explore other more advanced machine learning models. A good idea, for example, would be to experiment with bidirectional LSTMs. Bidirectional LSTMs add one more LSTM layer, which reverses the direction of information flow and the input sequence flows now both ways, forwards and backwards. Then the outputs are combined from both LSTM layers in several ways, such as average, sum, multiplication, or concatenation. One could explore to what extent additional layers of training of data would be beneficial to tune the involved parameters. Furthermore, investigating other ensemble methods that blend multiple forecasting models, such as bagging or stacking, can lead to improved prediction accuracy. In that context, one could experiment with a more sophisticated way of bagging. For example, one could train a meta-model (often a simple linear regression or neural network) to combine the predictions of multiple base models. The meta-model learns how to weigh the predictions of the base models based on their performance on a validation set. Another approach would be to use gradient-based optimization techniques to learn how to aggregate predictions, e.g. train a neural network to learn the optimal combination (weights) of base model predictions by minimizing a loss function. Additionally, instead of predicting a single point estimate (in LightGBM, or LSTM), we can predict multiple quantiles of the target variable using quantile regression forests. This would provide a range of predictions, allowing us to estimate prediction intervals along with point estimates. As far as the modeling of sunspots is concerned, another interesting approach would be to create a model that can handle possible interactions between two or more variables (correlated phenomena) and utilize their relationship to gain greater insight into the real underlying procedure, resulting in even better predictions. For example, such variables could be the total sunspot area, the strength of the Sun's polar magnetic fields during a solar cycle minimum, or other physical phenomena related to the Sun.

To draw to a close, in the dynamic intersection of time-series analysis and machine learning, the possibilities are boundless. The daily growth of data produced and collected is profound, and it is becoming urgent to not only marvel at the scale of all these data, but also to understand how they can be exploited and interpreted. As we conclude this thesis, let us be inspired by the endless potential that lies ahead in solving complex problems, making informed decisions, creating data-driven models, and driving innovation.

A OLS Linear Regression

A linear regression model establishes the relation between a dependent variable, y, and at least one independent variable, x, as:

$$\hat{y} = b_1 x + b_0 \tag{A.1}$$

In Ordinary Least Squares (OLS) method, we have to choose the values of b_1 and b_0 such that, the total sum of squares of the difference between the calculated values of $\hat{y}$, and observed values of y, is minimized. The formula for OLS is

$$\begin{aligned}\min_{b_0,b_1}(S) &= \min_{b_0,b_1} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \qquad (A.2)\\ &= \min_{b_0,b_1} \sum_{i=1}^{N} (y_i - b_1 x_i - b_0)^2 \\ &= \min_{b_0,b_1} \sum_{i=1}^{N} (\hat{\epsilon}_i)^2\end{aligned}$$

where, $\hat{y}_i$ is the predicted value for the i-th observation, y_i is the actual value for the i-th observation, ϵ_i is the error or residual for the i-th observation, and N is total number of observations. To get the values of b_0 and b_1 which minimize S, we can take the partial derivative for each coefficient and equate it to zero.

We use Python's `statsmodels` module to implement ordinary least squares method of linear regression. `Statsmodels` provides classes and functions for the estimation of many different statistical models. The result of the corresponding function call of OLS is a table containing various information. Below, we give a description of some of the terms in the table:

- **R-squared**: This is the proportion of the variance in the dependent variable that is predictable from the independent variable.
- **Adjusted R-squared**: This is the R-squared that has been adjusted based on the number of predictors in the model. It is a more accurate measure of the goodness of fit, especially when comparing models with different numbers of predictors.
- **F-statistic** and **Prob(F-statistic)**: The F-statistic is a measure of how significant the fit of the model is. It is calculated by dividing the mean squared error of the model by the mean squared error of the residuals. The p-value associated with the F-statistic is the probability that one would get the calculated value of F if the null hypothesis of no relationship between the variables were true.
- **coef (const, x1)**: These are the coefficients of the linear regression equation. The constant term is the y-intercept, and x1 is the slope of the line.
- **Condition Number**: This is a measure of the sensitivity of the function's output to its input. When the condition number is large, the function is ill-conditioned, meaning the output can be highly sensitive to changes in the input. In the context of regression analysis, a high condition number is a warning sign for multicollinearity among the predictors.

- **Omnibus** and **Prob(Omnibus)**: This is a test of the skewness and kurtosis of the residual (the difference between the observed and predicted values). The null hypothesis is that the residuals are normally distributed. The p-value associated with the Omnibus test is the probability that one would get the calculated value of Omnibus if the null hypothesis were true.
- **Skew**: This is a measure of data symmetry. A value of 0 indicates perfect symmetry. Positive skewness indicates a distribution with an asymmetric tail extending towards more positive values. Negative skewness indicates a distribution with an asymmetric tail extending towards more negative values.
- **Kurtosis**: This is a measure of the "tailedness" of the data. A high kurtosis may indicate that the data have heavy tails or outliers.
- **Durbin-Watson**: This is a test for auto-correlation in the residuals from a regression analysis. The test statistic ranges from 0 to 4, with a value around 2 indicating no auto-correlation. Values less than 2 and greater than 2 indicate positive and negative auto-correlation, correspondingly.
- **Jarque-Bera (JB)** and **Prob(JB)**: This is another test of the skewness and kurtosis of the residuals. The test compares the shape of the distribution of residuals to a normal distribution. The p-value associated with the Jarque-Bera test is the probability that one would get the calculated value of JB if the null hypothesis were true. A p-value less than the chosen significance level would indicate that the residuals are not normally distributed.

B Mann-Kendall Trend Test

The Mann-Kendall trend test (MK test) is used to analyze time series data for consistently increasing or decreasing trends (monotonic trends). It is a non-parametric test, which means it works for all distributions (i.e. data does not have to meet the assumption of normality), but data should have no serial correlation. If the data has a serial correlation, it could affect in significant level (p-value). It could lead to misinterpretation. To overcome this problem, researchers proposed several modified Mann-Kendall tests (Hamed and Rao Modified MK Test, Yue and Wang Modified MK Test, Modified MK test using Pre-Whitening method, etc.). Seasonal Mann-Kendall test also developed to remove the effect of seasonality. The two hypotheses of the Mann-Kendall test are as follows:

- **Null hypothesis** ($\mathcal{H}_o$): There is no trend present in the data.
- **Alternative hypothesis** ($\mathcal{H}_\alpha$): A trend is present in the data.

We use Python's `pyMannKendall` library to implement the Mann-Kendall trend test and its variants. Currently, this package has 11 Mann-Kendall tests and 2 Theil-Sen's slope estimator functions. Below, it is a brief description of these functions:

- **Original Mann-Kendall test** (`original_test`): Original Mann-Kendall test is a non-parametric test, which does not consider serial correlation or seasonal effects.

- **Hamed and Rao Modified MK Test** (`hamed_rao_modification_test`): This modified MK test proposed by Hamed and Rao (1998) to address serial autocorrelation issues. They suggested a variance correction approach to improve trend analysis. User can consider first n significant lag by insert lag number in this function. By default, it considered all significant lags.

- **Yue and Wang Modified MK Test** (`yue_wang_modification_test`): This is also a variance correction method for considered serial autocorrelation proposed by Yue, S., & Wang, C. Y. (2004). User can also set their desired significant n lags for the calculation.

- **Modified MK test using Pre-Whitening method** (`pre_whitening_modification_test`): This test suggested by Yue and Wang (2002) to using Pre-Whitening the time series before the application of trend test.

- **Modified MK test using Trend free Pre-Whitening method** (`trend_free_pre_whitening_modification_test`): This test also proposed by Yue and Wang (2002) to remove trend component and then Pre-Whitening the time series before application of trend test.

- **Multivariate MK Test** (`multivariate_test`): This is an MK test for multiple parameters proposed by Hirsch (1982). He used this method for seasonal MK test, where he considered every month as a parameter.

- **Seasonal MK Test** (`seasonal_test`): For seasonal time series data, Hirsch, R.M., Slack, J.R. and Smith, R.A. (1982) proposed this test to calculate the seasonal trend.

- **Regional MK Test** (`regional_test`): Based onHirsch (1982) proposed seasonal MK test, Helsel, D.R. and Frans, L.M., (2006) suggest regional MK test to calculate the overall trend in a regional scale.

- **Correlated Multivariate MK Test** (`correlated_multivariate_test`): This multivariate MK test proposed by Hipel (1994) where the parameters are correlated.

- **Correlated Seasonal MK Test** (`correlated_seasonal_test`): This method proposed by Hipel (1994) used, when time series significantly correlated with the preceding one or more months/seasons.

- **Partial MK Test** (`partial_test`): In a real event, many factors are affecting the main studied response parameter, which can bias the trend results. To overcome this problem, Libiseller (2002) proposed this partial MK test. It required two parameters as input, where, one is response parameter and other is an independent parameter.

- **Theil-Sen's Slope Estimator** (`sens_slope`): This method proposed by Theil (1950) and Sen (1968) to estimate the magnitude of the monotonic trend. Intercept is calculate using Conover, W.J. (1980) method.

- **Seasonal Theil-Sen's Slope Estimator** (`seasonal_sens_slope`): This method proposed by Hipel (1994) to estimate the magnitude of the monotonic trend, when data has seasonal effects. Intercept is calculate using Conover, W.J. (1980) method.

The result of any function implementing a Mann-Kendall test is a table containing various information. Below, we give a description of some of the terms in the table:

- **Trend**: This value tells us the existence of any trend.
- **Hypothesis Test Result (h)**: This value is true or false, depending on the rejection of the null hypothesis or not based on the data.
- **p-value**: The p-value tells us if we have strong evidence to reject the null hypothesis of no trend or not, depending on the chosen significance level.
- **Test Statistic (z)**: The z-score is a measure of how many standard deviations the observed trend is away from what we would expect if there were no trend. A larger absolute value of z would provide stronger evidence against the null hypothesis.
- **Kendall's Tau**: This is a measure of the correlation between the observed data and the time variable. A positive value indicates a positive correlation, suggesting an upward trend over time. A negative value indicates a negative correlation, suggesting a downward trend over time.
- **Mann-Kendall Score (s)**: This is the test statistic for the Mann-Kendall test. A positive value indicates an upward trend and a negative value indicates a downward trend.
- **Variance (var_s)**: This is the variance of the test statistic. It is used in the calculation of the z-score and the p-value.
- **Slope**: The slope is the estimated rate of change in the values of y, according to the Theil-Sen Estimator (or to the Seasonal Theil-Sen Estimator).
- **Intercept**: The intercept is the estimated value of y at the start of the time-series, according to the Theil-Sen Estimator (or to the Seasonal Theil-Sen Estimator).

C Kolmogorov - Smirnov Test

The Kolmogorov-Smirnov test (KS test) is used to compare two distributions to determine if they are pulling from the same underlying distribution. The KS test is non-parametric, which means we do not need to rely on assumptions that the data are drawn from a given family of distributions. This is good, since we often will not know the underlying distribution beforehand in the real world. There are two versions of the KS test:

- **Two-Sample**: eCDF of A compared to eCDF of B
- **One-Sample**: eCDF of A compared to CDF of B

The one-sample KS test is used when we are comparing a single empirical sample to a theoretical parameterized distribution. The two-sample KS test is used when we are comparing two empirical samples. When comparing two samples, we are trying to answer the following question: “What is the probability that these two sets of samples were drawn from the same probability distribution?”.

The KS test uses the following hypotheses:

- **Null hypothesis** ($\mathcal{H}_o$): The two samples of the data at hand are from the same distribution.
- **Alternative hypothesis** ($\mathcal{H}_\alpha$): The two samples of the data at hand are not from the same distribution.

The KS statistic can be expressed as:

$$D = \sup_x(|F_1(x) - F_2(x)|)$$

where F_1 and F_2 are the two cumulative distribution functions of the first and second samples, respectively. Another way to put it is that the KS statistic is the maximum absolute difference between the two cumulative distributions. To calculate this value, the KS statistic is taken into account along with the sample size of both distributions. Typical thresholds for rejecting the null hypothesis are 1% and 5%, implying that any p-value less than or equal to these values would lead to the rejection of the null hypothesis. The image below shows an example of the statistic, depicted as a black arrow.

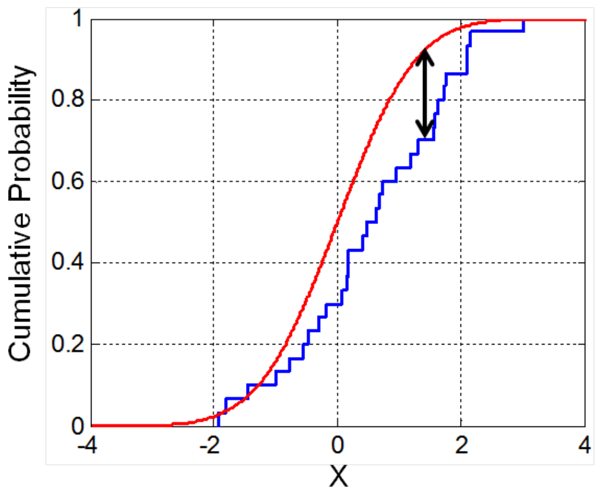


Figure C.1: Illustration of the Kolmogorov–Smirnov statistic. The red line is a model CDF, the blue line is an empirical CDF, and the black arrow is the KS statistic.

D Cholesky Decomposition

The Cholesky decomposition of an asymmetric, positive definite matrix $\mathbf{A}$ decomposes $\mathbf{A}$ into a product of a lower triangular matrix $\mathbf{L}$ and its transpose

$$\mathbf{L}\mathbf{L}^T = \mathbf{A}$$

where L is called the Cholesky factor. The Cholesky decomposition is useful for solving linear systems with symmetric, positive definite coefficient matrix $\mathbf{A}$. To solve $\mathbf{Ax} = \mathbf{b}$ for

$\mathbf{x}$, first solve the triangular system $\mathbf{Ly} = \mathbf{b}$ by forward solving linear systems substitution and then the triangular system $\mathbf{L}^T\mathbf{x} = \mathbf{y}$ by back substitution. Using the backslash operator, we write the solution as $\mathbf{x} = \mathbf{L}^T\backslash(\mathbf{L}\backslash\mathbf{b})$, where the notation $A\backslash b$ is the vector $\mathbf{x}$ which solves $\mathbf{Ax} = \mathbf{b}$. Both the forward and backward substitution steps require $n^2/2$ operations, when $\mathbf{A}$ is of size $n \times n$. The computation of the Cholesky factor $\mathbf{L}$ is considered numerically extremely stable and takes time $n^3/6$, so it is the method of choice when it can be applied.

Note, also, that the determinant of a positive definite symmetric matrix can be calculated efficiently by

$$\det(\mathbf{A}) = |\mathbf{A}| = \prod_{i=1}^{n} L_{ii}^2$$

or

$$\log(\det(\mathbf{A})) = \log(|\mathbf{A}|) = 2\sum_{i=1}^{n} \log(L_{ii})$$

where $\mathbf{L}$ is the Cholesky factor from $\mathbf{A}$.

E Gauss-Hermite Quadrature

In numerical analysis, Gauss–Hermite quadrature is a form of Gaussian quadrature for approximating the value of integrals of the following kind

$$\int_{-\infty}^{+\infty} e^{-x^2} f(x)\, dx.$$

In this case

$$\int_{-\infty}^{+\infty} e^{-x^2} f(x)\, dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

where n is the number of sample points used. The x_i, $i = 1, 2, \ldots, n$, are the roots of the physicists' version of the Hermite polynomial $H_n(x)$, and the associated weights w_i are given by

$$w_i = \frac{2^{n-1} n! \sqrt{\pi}}{n^2 [H_{n-1}(x_i)]^2}$$

The "physicist's Hermite polynomials" are given by

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

Consider a function $h(y)$, where the variable y is normally distributed

$$y \sim \mathcal{N}(\mu, \sigma^2)$$

The expectation of h corresponds to the following integral

$$E[h(y)] = \int_{-\infty}^{+\infty} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) h(y) dy$$

As this does not exactly correspond to the Hermite polynomial, we need to change variables

$$x = \frac{y-\mu}{\sqrt{2}\sigma} \Leftrightarrow y = \sqrt{2}\sigma x + \mu$$

Coupled with the integration by substitution, we obtain

$$E[h(y)] = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{\pi}} \exp(-x^2) h(\sqrt{2}\sigma x + \mu) dx$$

leading to

$$E[h(y)] \approx \frac{1}{\sqrt{\pi}} \sum_{i=1}^{n} w_i h(\sqrt{2}\sigma x_i + \mu)$$

F Matrix Derivatives

Derivatives of the elements of an inverse matrix:

$$\frac{\partial}{\partial\theta} K^{-1} = -K^{-1} \frac{\partial K}{\partial\theta} K^{-1}$$

where $\frac{\partial K}{\partial\theta}$ is a matrix of element-wise derivatives. For the log determinant of a derivative of log determinant positive definite symmetric matrix we have

$$\frac{\partial}{\partial\theta} \log(|K|) = \operatorname{tr}\left(K^{-1}\frac{\partial K}{\partial\theta}\right)$$

G Marginal and Conditional Distributions of MVN

Suppose $X = (x_1, x_2)$ is a joint Gaussian with parameters

$$\begin{bmatrix}\mu_1 \\ \mu_2\end{bmatrix}, \quad \Sigma = \begin{bmatrix}\Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22}\end{bmatrix}, \quad \Lambda = \Sigma^{-1} = \begin{bmatrix}\Lambda_{11} & \Lambda_{12} \\ \Lambda_{21} & \Lambda_{22}\end{bmatrix}$$

then the marginals are given by

$$p(x_1) = \mathcal{N}(x_1|\mu_1, \Sigma_{11})$$

$$p(x_2) = \mathcal{N}(x_2|\mu_2, \Sigma_{22})$$

and the posterior conditional is given by

$$p(x_1|x_2) = \mathcal{N}(x_1|\mu_{1|2}, \Sigma_{1|2})$$

where

$$\begin{aligned}\mu_{1|2} &= \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2) \\ &= \mu_1 - \Lambda_{11}^{-1}\Lambda_{12}(x_2 - \mu_2) \\ &= \Sigma_{1|2}(\Lambda_{11}\mu_1 - \Lambda_{12}(x_2 - \mu_2))\end{aligned}$$

$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} = \Lambda_{11}^{-1}$$

H Sunspot Number Datasets

H.1 13-month smoothed monthly total sunspot number

Credit: WDC-SILSO, Royal Observatory of Belgium, Brussels. Info link: 13-month smoothed monthly total sunspot number (info). Download link: 13-month smoothed monthly total sunspot number (csv file).

H.1.1 Data description

The 13-month smoothed monthly total sunspot number is derived by a "tapered-boxcar" running mean of monthly sunspot numbers over 13 months centered on the corresponding month (Smoothing function: equal weights = 1, except for first and last elements (-6 and $+6$ months) = 0.5, Normalization by 1/12 factor). There are no smoothed values for the first 6 months and last 6 months of the data series: columns 4, 5 and 6 are set to -1 (no data). The time range of the data set is from 01/1749 until 12/2022 (constantly renewed).

H.1.2 Choice of smoothing

This 13-month smoothed series is provided only for backward compatibility with a large number of past publications and methods resting on this smoothed series. It has thus become a base standard (e.g. for the conventional definition of the times of minima and maxima of solar cycles).

However, a wide range of other smoothing functions can be used, often with better low-pass filtering and anti-aliasing properties. As the optimal filter choice depends on the application, we thus invite users to start from the monthly mean Sunspot Numbers and apply the smoothing function that is most appropriate for their analyses. The classical smoothed series included here should only be used when direct comparisons with past published analyses must be made.

H.1.3 Error values

The standard deviations in this file are obtained from the weighted mean of the variances of the 13 months in the running mean value:

$$\sigma_{ms} = \sqrt{\frac{\sum (w_M \cdot \sigma_M^2)}{\sum (w_M)}}$$

where σ_M is the standard deviation for a single month, w_M is 1 or 0.5 and $M = 13$ in this case. As successive monthly means are highly correlated, the standard error on the smoothed values can be estimated by the same formula as for a single month: $\sigma/\sqrt{N}$ where σ is the listed standard deviation and N the total number of observations in the month. The number of observations given in column 6 is the number of observations of the corresponding (middle) month. This thus gives a smoothed mean of monthly standard deviations, i.e. with the same low-pass filtering as the data value itself. Further autocorrelation analyses will be needed to derive a conversion of this standard deviation to a standard error of the 13-month smoothed number.

H.1.4 File format

Below, we give some details about the csv file containing the dataset about the 13-month smoothed monthly total sunspot number:

- Filename: SN_ms_tot_V2.0.csv
- Format: Comma Separated Values (CSV)
- Contents:
  - Column 1-2: Gregorian calendar date (year - month).
  - Column 3: Date in fraction of year.
  - Column 4: Monthly smoothed total sunspot number.
  - Column 5: Monthly mean standard deviation of the input sunspot numbers.
  - Column 6: Number of observations used to compute the corresponding monthly mean total sunspot number.
  - Column 7: Definitive/provisional marker.'1' indicates that the value is definitive. '0' indicates that the value is still provisional.

H.2 Yearly mean total sunspot number

Credit: WDC-SILSO, Royal Observatory of Belgium, Brussels. Info link: Yearly mean total sunspot number (info). Download link: Yearly mean total sunspot number (csv file).

H.2.1 Data description

Yearly mean total sunspot number obtained by taking a simple arithmetic mean of the daily total sunspot number over all days of each year. (NB: in early years in particular before 1749, the means are computed on only a fraction of the days in each year because on many days, no observation is available). A value of -1 indicates that no number is available (missing value).

H.2.2 Error values

The yearly standard deviation of individual data is derived from the daily values by the same formula as the monthly means:

$$\sigma_m = \sqrt{\frac{\sum (N_d \cdot \sigma_d^2)}{\sum (N_d)}}$$

where σ_d is the standard deviation for a single day and N_d is the number of observations for that day. The standard error on the yearly mean values can be computed by: $\sigma/\sqrt{N}$ where σ is the listed standard deviation and N the total number of observations in the year. NB: this standard error gives a measure of the precision, i.e. the sensitivity of the yearly value to different samples of daily values with random errors. The uncertainty on the mean (absolute accuracy) is only determined on longer time scales, and is thus not given here for individual yearly values.

H.2.3 File format

Below, we give some details about the csv file containing the dataset about the yearly mean total sunspot number:

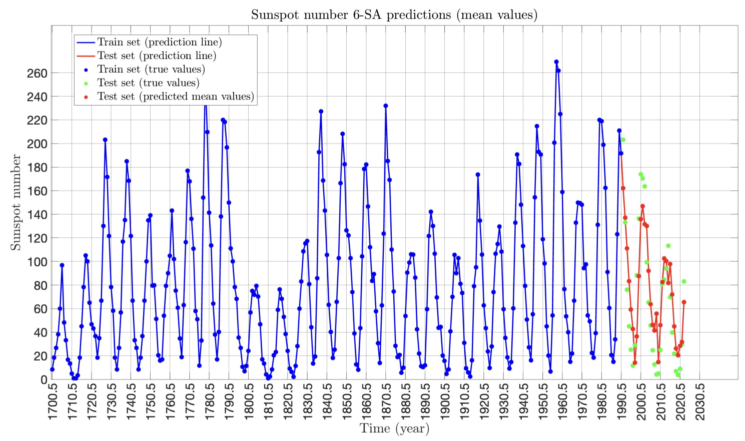
- Filename: SN_y_tot_V2.0.csv
- Format: Comma Separated Values (CSV)
- Contents:
  - Column 1: Gregorian calendar year (mid-year date).
  - Column 2: Yearly mean total sunspot number.
  - Column 3: Yearly mean standard deviation of the input sunspot numbers from individual stations.
  - Column 4: Number of observations used to compute the yearly mean total sunspot number.
  - Column 5: Definitive/provisional marker. '1' indicates that the value is definitive. '0' indicates that the value is still provisional.

Solar Cycle	Minimum (Y-M)	Smoothed minimum SSN	Maximum (Y-M)	Smoothed maximum SSN	Average spots per day	Time of Rise (Y-M)	Duration (Y-M)	Spotless days
1	1755-02	14.0	1761-06	144.1	70	06-04	11-04	-
2	1766-06	18.6	1769-09	193.0	99	03-03	09-00	-
3	1775-06	12.0	1778-05	264.2	111	02-11	09-03	-
4	1784-09	15.9	1788-02	235.3	103	03-05	13-07	-
5	1798-04	5.3	1805-02	82.0	38	06-10	12-03	-
6	1810-07	0.0	1816-05	81.2	31	05-10	12-10	-
7	1823-05	0.1	1829-11	119.2	63	06-06	10-06	-
8	1833-11	12.2	1837-03	244.9	112	03-04	09-08	-
9	1843-07	17.6	1848-02	219.9	99	04-07	12-05	-
10	1855-12	6.0	1860-02	186.2	92	04-02	11-03	561
11	1867-03	9.9	1870-08	234.0	89	03-05	11-09	942
12	1878-12	3.7	1883-12	124.4	57	05-00	11-03	872
13	1890-03	8.3	1894-01	146.5	65	03-10	11-10	782
14	1902-01	4.5	1906-02	107.1	54	04-01	11-06	1007
15	1913-07	2.5	1917-08	175.7	73	04-01	10-01	640
16	1923-08	9.3	1928-04	130.2	68	04-08	10-01	514
17	1933-09	5.8	1937-04	198.6	96	03-07	10-05	384
18	1944-02	12.9	1947-05	218.7	109	03-03	10-02	382
19	1954-04	5.1	1958-03	285.0	129	03-11	10-06	337
20	1964-10	14.3	1968-11	156.6	86	04-01	11-05	285
21	1976-03	17.8	1979-12	232.9	111	03-09	10-06	283
22	1986-09	13.5	1989-11	212.5	106	03-02	09-11	257
23	1996-08	11.2	2001-11	180.3	82	05-03	12-04	619
24	2008-12	2.2	2014-04	116.4	49	05-04	11-00	914
25	2019-12	1.8	-	-	-	-	-	-

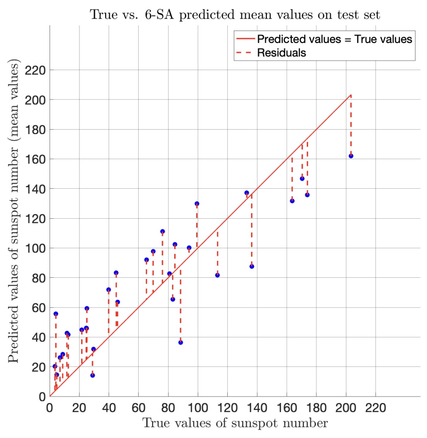
Table H.1: Details of solar cycles 1 to 25. Based on the 13-month smoothed monthly total sunspot number time series. Credit: WDC-SILSO, Royal Observatory of Belgium, Brussels.

I Warped GPR (Predictions using the mean value)

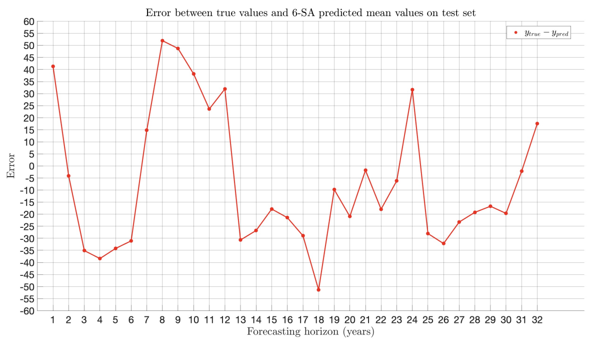
Here, we give the figures showing the predictions made by the warped GPR model, when we consider the mean value as the point estimate, for forecasting horizons 6, 12, and 32.



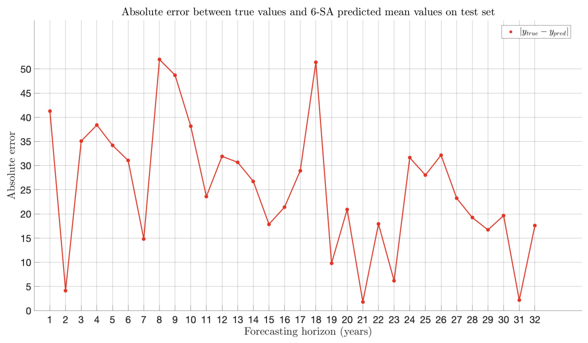
(a) Yearly mean total sunspot number predictions using 6-SA forecasting strategy.



(b) True vs. 6-SA predicted values on the test set.

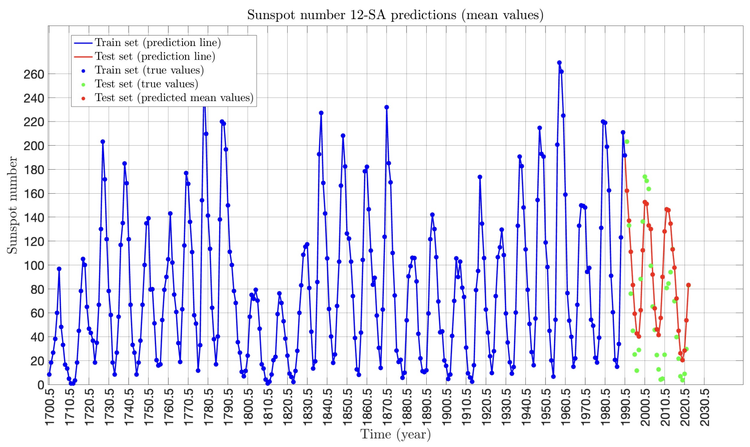


(c) Error between true and 6-SA predicted values on the test set.

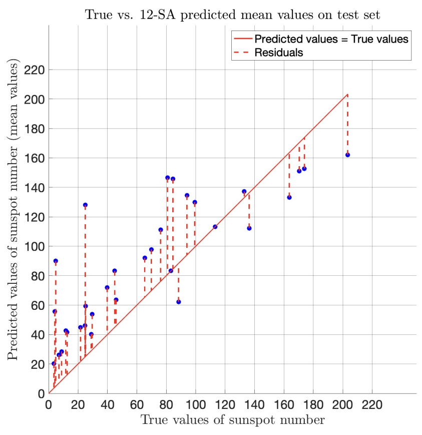


(d) Absolute error between true and 6-SA predicted values on the test set.

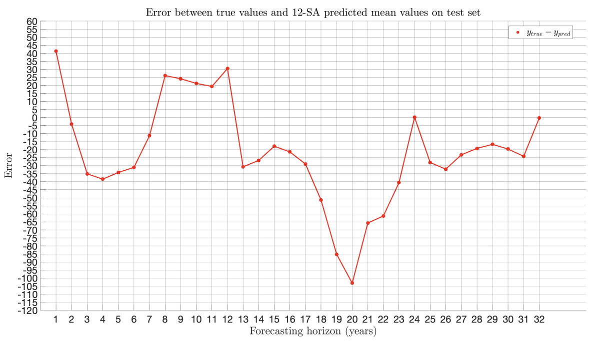
Figure I.1: **Warped GPR**. Predicting the yearly mean total sunspot number using **6-SA** forecasting strategy. We consider the **mean** value as the point estimates.



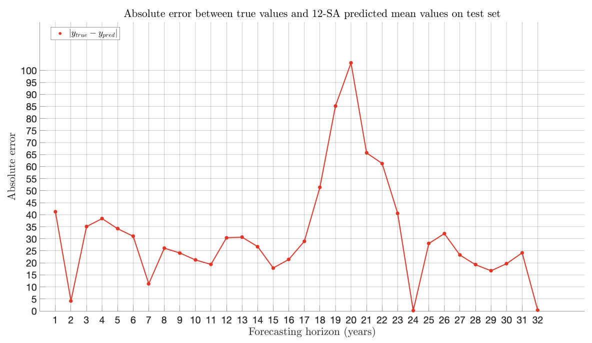
(a) Yearly mean total sunspot number predictions using 12-SA forecasting strategy.



(b) True vs. 12-SA predicted values on the test set.

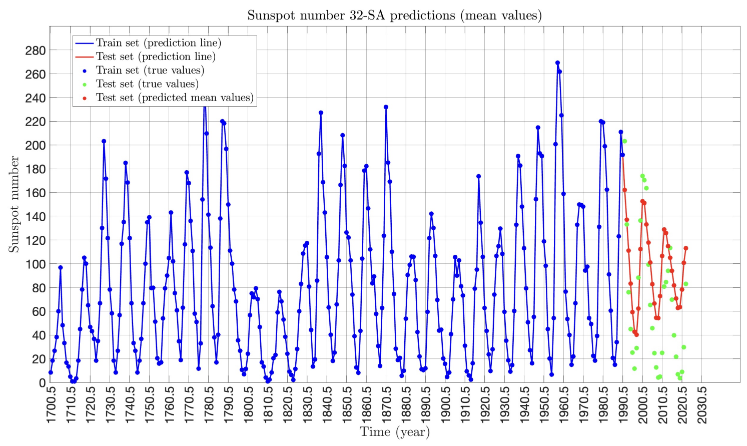


(c) Error between true and 12-SA predicted values on the test set.

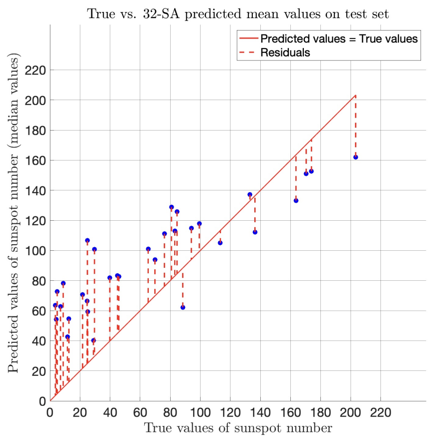


(d) Absolute error between true and 12-SA predicted values on the test set.

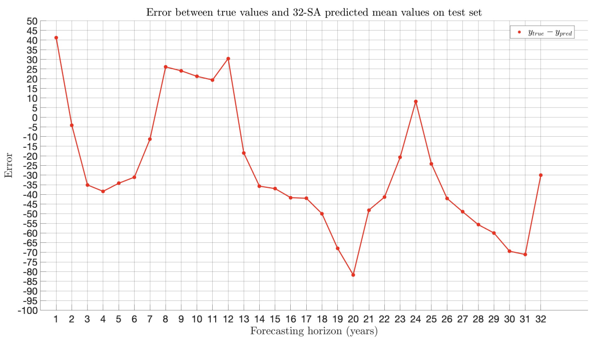
Figure I.2: **Warped GPR**. Predicting the yearly mean total sunspot number using **12-SA** forecasting strategy. We consider the **mean** value as the point estimates.



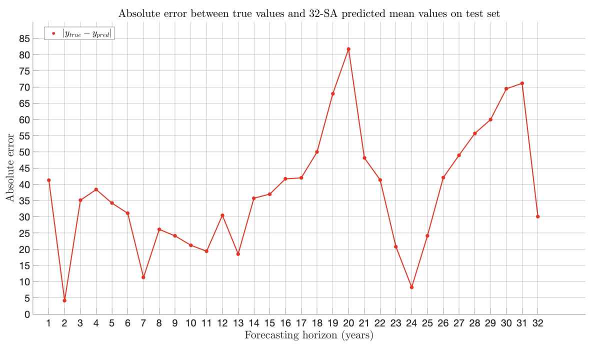
(a) Yearly mean total sunspot number predictions using 32-SA forecasting strategy.



(b) True vs. 32-SA predicted values on the test set.



(c) Error between true and 32-SA predicted values on the test set.



(d) Absolute error between true and 32-SA predicted values on the test set.

Figure I.3: **Warped GPR**. Predicting the yearly mean total sunspot number using **32-SA** forecasting strategy. We consider the **mean** value as the point estimates.

References

[1] T. Hastie, R. Tibshirani, J. Friedman, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics Ser., Springer, Jun 2017.

[2] W. A. Fuller, *Introduction to Statistical Time Series.* Wiley-Interscience, Apr 1996.

[3] A. Kumar, "Difference between Parametric vs Non-Parametric Models." `https://vitalflux.com/difference-between-parametric-vs-non-parametric-models/`, Jun 2023. [Online; accessed 10-September-2023].

[4] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning.* Adaptive Computation and Machine Learning Ser., MIT Press, Nov 2005.

[5] Wikipedia contributors, "Gradient boosting — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=Gradient_boosting&oldid=1174466888`, 2023. [Online; accessed 10-September-2023].

[6] Wikipedia contributors, "XGBoost — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=XGBoost&oldid=1164850953`, 2023. [Online; accessed 10-September-2023].

[7] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*, vol. 30. Curran Associates, Inc., 2017.

[8] X. S. Liang, F. Xu, Y. Rong, R. Zhang, X. Tang, and F. Zhang, "El niño modoki can be mostly predicted more than 10 years ahead of time," *Scientific Reports*, vol. 11, p. 17860, Sep 2021.

[9] S. Scoles, "Scientists tackle a burning question: When will our quiet sun turn violent?," *Science*, Mar 2021.

[10] Z. Savitsky, "Peak solar activity is arriving sooner than expected, reaching levels not seen in 20 years," *Science*, Sep 2023.

[11] Wikipedia contributors, "Stochastic process — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=Stochastic_process&oldid=1172842195`, 2023. [Online; accessed 10-September-2023].

[12] U. Triacca, "Lesson 3: Basic theory of stochastic processes." `https://www.lem.sssup.it/phd/documents/Lesson3.pdf`, 2015. [Online; accessed 10-September-2023].

[13] F. Pelgrin, "Lecture 1: Fundamental concepts in time series analysis (part 1)." `https://math.univ-cotedazur.fr/~frapetti/CorsoP/chapitre_1_part_1_IMEA_1.pdf`, 2011. [Online; accessed 10-September-2023].

[14] R. Molinari, H. Xu, Y. Zhang, and S. Guerrier, "Applied Time Series Analysis with R." `https://smac-group.github.io/ts/fundtimeseries.html`, Aug 2019. [Online; accessed 10-September-2023].

[15] Wikipedia contributors, "Granger causality — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Granger_causality&oldid=1170235412, 2023. [Online; accessed 10-September-2023].

[16] J. Geweke, "Measurement of Linear Dependence and Feedback between Multiple Time Series," *Journal of the American Statistical Association*, vol. 77, no. 378, pp. 304–313, 1982.

[17] A. Seth, "Granger causality," *Scholarpedia*, vol. 2, no. 7, p. 1667, 2007. revision #127333.

[18] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*. 3rd edition, OTexts: Melbourne, Australia, May 2021. https://OTexts.com/fpp3 [Online; accessed 10-September-2023].

[19] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning, "STL: A seasonal-trend decomposition," *J. Off. Stat*, vol. 6, no. 1, pp. 3–73, 1990.

[20] A. Singh, "A Gentle Introduction to Handling a Non-Stationary Time Series in Python." https://www.analyticsvidhya.com/blog/2018/09/non-stationary-time-series-python/, Sep 2018. [Online; accessed 10-September-2023].

[21] D. A. Dickey and W. A. Fuller, "Distribution of the estimators for autoregressive time series with a unit root," *Journal of the American Statistical Association*, vol. 74, no. 366, pp. 427–431, 1979.

[22] Wikipedia contributors, "Dickey–fuller test — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Dickey%E2%80%93Fuller_test&oldid=1149660427, 2023. [Online; accessed 10-September-2023].

[23] G. Elliott, T. J. Rothenberg, and J. H. Stock, "Efficient tests for an autoregressive unit root," *Econometrica*, vol. 64, no. 4, pp. 813–836, 1996.

[24] Greene, *Econometric Analysis*. Pearson, Aug 2007.

[25] Wikipedia contributors, "Augmented dickey–fuller test — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Augmented_Dickey%E2%80%93Fuller_test&oldid=1124807741, 2022. [Online; accessed 10-September-2023].

[26] G. Elliott, T. J. Rothenberg, and J. H. Stock, "Efficient tests for an autoregressive unit root," *Econometrica*, vol. 64, p. 813, Jul 1996.

[27] Wikipedia contributors, "ADF-GLS test — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=ADF-GLS_test&oldid=1080038141, 2022. [Online; accessed 10-September-2023].

[28] D. Kwiatkowski, P. C. Phillips, P. Schmidt, and Y. Shin, "Testing the null hypothesis of stationarity against the alternative of a unit root," *Journal of Econometrics*, vol. 54, p. 159–178, Oct 1992.

[29] H. B. Nielsen, "Non-Stationary Time Series and Unit Root Tests." https://studylib.net/doc/18145075/non-stationary-time-series-and-unit-root-tests, 2005. [Online; accessed 10-September-2023].

[30] Wikipedia contributors, "KPSS test — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=KPSS_test&oldid=1162349563, 2023. [Online; accessed 10-September-2023].

[31] C. B. Phillips, Peter and P. Perron, "Testing for a unit root in time series regression," *Biometrika*, vol. 75, no. 2, p. 335–346, 1988.

[32] Wikipedia contributors, "Phillips–Perron test — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Phillips%E2%80%93Perron_test&oldid=1080038074, 2022. [Online; accessed 10-September-2023].

[33] R. Davidson and J. G. MacKinnon, *Econometric Theory and Methods.* Oxford University Press, Nov 2004.

[34] C. M. Bishop, *Pattern Recognition and Machine Learning.* Information Science and Statistics Ser., Springer, Apr 2011.

[35] Wikipedia contributors, "Training, validation, and test data sets — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Training,_validation,_and_test_data_sets&oldid=1162801019, 2023. [Online; accessed 10-September-2023].

[36] Oxford Dictionaries - English. https://web.archive.org/web/20171107014257/https://en.oxforddictionaries.com/definition/overfitting. [Online; accessed 10-September-2023].

[37] B. S. Everitt and A. Skrondal, *The Cambridge Dictionary of Statistics.* Cambridge University Press, Aug 2010.

[38] K. P. Burnham and D. R. Anderson, *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach.* Springer, Dec 2010.

[39] Wikipedia contributors, "Overfitting — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Overfitting&oldid=1171074384, 2023. [Online; accessed 10-September-2023].

[40] Wikipedia contributors, "Bias–variance tradeoff — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Bias%E2%80%93variance_tradeoff&oldid=1170800717, 2023. [Online; accessed 10-September-2023].

[41] I. Logunova and A. Khaciyants, "Bias-Variance Tradeoff in Machine Learning." https://serokell.io/blog/bias-variance-tradeoff, Mar 2023. [Online; accessed 10-September-2023].

[42] J. Wang, "An Intuitive Tutorial to Gaussian Processes Regression," 2022.

[43] Wikipedia contributors, "Multivariate normal distribution — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Multivariate_normal_distribution&oldid=1171661589, 2023. [Online; accessed 10-September-2023].

[44] F. Dablander, "Two properties of the Gaussian distribution." https://fabiandablander.com/statistics/Two-Properties.html, Feb 2019. [Online; accessed 10-September-2023].

[45] K. Manohar, "Marginalization vs Conditioning for Multivariate Gaussian Distribution." https://kusemanohar.wordpress.com/2020/04/22/marginalization-vs-conditioning-for-multivariate-gaussian-distribution/, Apr 2020. [Online; accessed 10-September-2023].

[46] T. Beckers, "An Introduction to Gaussian Process Models," 2021.

[47] E. Snelson, Z. Ghahramani, and C. Rasmussen, "Warped gaussian processes," in *Advances in Neural Information Processing Systems* (S. Thrun, L. Saul, and B. Schölkopf, eds.), vol. 16, MIT Press, 2003.

[48] V. D. Agou, A. Pavlides, and D. T. Hristopulos, "Spatial modeling of precipitation based on data-driven warping of gaussian processes," *Entropy*, vol. 24, no. 3, 2022.

[49] M. Ebden, "Gaussian Processes: A Quick Introduction," 2015.

[50] R. B. Gramacy, *Surrogates: Gaussian Process Modeling, Design and Optimization for the Applied Sciences.* Boca Raton, Florida: Chapman Hall/CRC, 2020. http://bobby.gramacy.com/surrogates/.

[51] E. P. Xing, K. Genin, and Y. Zheng, "10-708: Probabilistic Graphical Models, 21: Advanced Gaussian Processes." https://www.cs.cmu.edu/~epxing/Class/10708-15/notes/10708_scribe_lecture21.pdf, 2015. [Online; accessed 10-September-2023].

[52] D. Duvenaud, "Automatic model construction with Gaussian processes," 2014.

[53] A. Onken and A. Vergari, "Gaussian Processes and Kernels." https://mlpr.inf.ed.ac.uk/2022/notes/w6a_gaussian_process_kernels.pdf. [Online; accessed 10-September-2023].

[54] P. Akilashri, S. Bharathi, G. Nithya, A. Parveen, and B. Prabha, *Big Data for Analytics.* SK Research Group of Companies, 2023.

[55] V. Kurama, "An Introduction to Decision Trees." https://blog.paperspace.com/decision-trees/, Feb 2020. [Online; accessed 10-September-2023].

[56] Geeks for Geeks, "Decision Tree." https://www.geeksforgeeks.org/decision-tree/, Aug 2023. [Online; accessed 10-September-2023].

[57] L. Breiman and R. Ihaka, *Nonlinear Discriminant Analysis Via Scaling and ACE.* Technical report (University of California, Berkeley. Department of Statistics), Department of Statistics, University of California, 1984.

[58] B. Ripley, *Pattern Recognition and Neural Networks.* Cambridge University Press, 2007.

[59] Project Pro, "What is Bagging vs Boosting in Machine Learning?." `https://www.projectpro.io/article/bagging-vs-boosting-in-machine-learning/579`, Jul 2023. [Online; accessed 10-September-2023].

[60] Wikipedia contributors, "Random forest — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=Random_forest&oldid=1174573646`, 2023. [Online; accessed 10-September-2023].

[61] D. Mwiti, "Random Forest Regression: When Does It Fail and Why?." `https://neptune.ai/blog/random-forest-regression-when-does-it-fail-and-why`, Jul 2022. [Online; accessed 10-September-2023].

[62] J. Brownlee, "A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning - MachineLearningMastery.com." `https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/`, Aug 2020. [Online; accessed 10-September-2023].

[63] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," Mar 2016.

[64] A. Hachcham, "XGBoost: Everything You Need to Know." `https://neptune.ai/blog/xgboost-everything-you-need-to-know`, Jul 2022. [Online; accessed 10-September-2023].

[65] Geeks for Geeks, "XGBoost." `https://www.geeksforgeeks.org/xgboost/`, Sep 2021. [Online; accessed 10-September-2023].

[66] R. C. Staudemeyer and E. R. Morris, "Understanding lstm – a tutorial into long short-term memory recurrent neural networks," 2019.

[67] Geeks for Geeks, "Introduction to Deep Learning." `https://www.geeksforgeeks.org/introduction-deep-learning/`, Apr 2023. [Online; accessed 10-September-2023].

[68] Amazon Web Services, Inc., "What is Deep Learning?." `https://aws.amazon.com/what-is/deep-learning/`. [Online; accessed 10-September-2023].

[69] M. Minsky and S. Papert, "An introduction to computational geometry," *Cambridge tiass., HIT*, vol. 479, no. 480, p. 104, 1969.

[70] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[71] R. J. Williams and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *Neural Computation*, vol. 1, pp. 270–280, 06 1989.

[72] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[73] M. I. Jordan, *Attractor Dynamics and Parallelism in a Connectionist Sequential Machine*, p. 112–127. IEEE Press, 1990.

[74] D. E. Rumelhart and J. L. McClelland, *Learning Internal Representations by Error Propagation*, pp. 318–362. MIT Press, 1987.

[75] R. J. Williams and D. Zipser, *Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity*, p. 433–486. USA: L. Erlbaum Associates Inc., 1995.

[76] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural Computation*, vol. 12, pp. 2451–2471, 10 2000.

[77] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[78] S. Hochreiter and J. Schmidhuber, "Lstm can solve hard long time lag problems," in *Proceedings of the 9th International Conference on Neural Information Processing Systems*, NIPS'96, (Cambridge, MA, USA), p. 473–479, MIT Press, 1996.

[79] F. Informatik, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," *A Field Guide to Dynamical Recurrent Neural Networks*, 03 2003.

[80] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.

[81] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *J. Mach. Learn. Res.*, vol. 3, p. 115–143, mar 2003.

[82] O. Calzone, "An intuitive explanation of lstm." `https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c`, Apr 2022. [Online; accessed 10-September-2023].

[83] A. Graves, A. rahman Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," 2013.

[84] A. Graves, "Generating sequences with recurrent neural networks," 2014.

[85] NASA - Solar System Exploration, "Our Sun - In Depth." `https://solarsystem.nasa.gov/solar-system/sun/in-depth`, Oct 2021. [Online; accessed 10-September-2023].

[86] NASA, SouthWest Research Institute, and Interstellar Boundary Explorer, "How does the Sun's magnetic field work?." `http://ibex.swri.edu/students/How_does_the_Sun.shtml`. [Online; accessed 10-September-2023].

[87] NASA Scientific Visualization Studio (SVS) at the Goddard Space Flight Center (GSFC), "The Sun's Magnetic Field." `https://svs.gsfc.nasa.gov/4124`, Dec 2013. [Online; accessed 10-September-2023].

[88] NASA, "Understanding the Magnetic Sun." `http://www.nasa.gov/feature/goddard/2016/understanding-the-magnetic-sun`, Jan 2016. [Online; accessed 10-September-2023].

[89] Space Weather Prediction Center (SWPC) at National Oceanic and Atmoshperic Administration (NOAA), "Solar Wind." `https://www.swpc.noaa.gov/phenomena/solar-wind`. [Online; accessed 10-September-2023].

[90] D. Dobrijevic, "Solar wind: What is it and how does it affect Earth?." `https://www.space.com/22215-solar-wind.html`, Jul 2022. [Online; accessed 10-September-2023].

[91] Space Weather Prediction Center (SWPC) at National Oceanic and Atmoshperic Administration (NOAA), "Coronal Holes." `https://www.swpc.noaa.gov/phenomena/coronal-holes`. [Online; accessed 10-September-2023].

[92] NASA Scientific Visualization Studio (SVS) at the Goddard Space Flight Center (GSFC), "Coronal Holes at Solar Minimum and Solar Maximum." `https://svs.gsfc.nasa.gov/4854`. [Online; accessed 10-September-2023].

[93] NASA's Jet Propulsion Laboratory (JPL), "Coronal Mass Ejections on the Sun." `https://www.jpl.nasa.gov/nmp/st5/SCIENCE/cme.html`. [Online; accessed 10-September-2023].

[94] Space Weather Prediction Center (SWPC) at National Oceanic and Atmoshperic Administration (NOAA), "Coronal Mass Ejections." `https://www.swpc.noaa.gov/phenomena/coronal-mass-ejections`. [Online; accessed 10-September-2023].

[95] Wikipedia contributors, "Coronal mass ejection — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=Coronal_mass_ejection&oldid=1172379629`, 2023. [Online; accessed 10-September-2023].

[96] Center of Science Education at University Corporation for Atmospheric Research, "Coronal Mass Ejections (CME)." `https://scied.ucar.edu/learning-zone/sun-space-weather/coronal-mass-ejection`. [Online; accessed 10-September-2023].

[97] D. Dobrijevic, "Sunspots: What are they, and why do they occur?." `https://www.space.com/sunspots-formation-discovery-observations`. [Online; accessed 10-September-2023].

[98] Wikipedia contributors, "Solar cycle 25 — Wikipedia, the free encyclopedia." `https://en.wikipedia.org/w/index.php?title=Solar_cycle_25&oldid=1173655094`, 2023. [Online; accessed 10-September-2023].

[99] R. Werner, "Sunspot Number Prediction by an Autoregressive Model," *Sun and Geosphere*, vol. 7, pp. 75–80, Nov. 2012.

[100] Ítalo G. Gonçalves, E. Echer, and E. Frigo, "Sunspot cycle prediction using warped gaussian process regression," *Advances in Space Research*, vol. 65, no. 1, pp. 677–683, 2020.

[101] E. Covas, N. Peixinho, and J. Fernandes, "Neural network forecast of the sunspot butterfly diagram," *Solar Physics*, vol. 294, feb 2019.

[102] L. A. Upton and D. H. Hathaway, "An updated solar cycle 25 prediction with AFT: The modern minimum," *Geophysical Research Letters*, vol. 45, pp. 8091–8095, aug 2018.

[103] F. Labonville, P. Charbonneau, and A. Lemerle, "A dynamo-based forecast of solar cycle 25," *Solar Physics*, vol. 294, 06 2019.

[104] Z. Pala and R. Atıcı, "Forecasting sunspot time series using deep learning methods," *Solar Physics*, vol. 294, 05 2019.

[105] D. I. Okoh, G. K. Seemala, A. B. Rabiu, J. Uwamahoro, J. B. Habarulema, and M. Aggarwal, "A hybrid regression-neural network (hr-nn) method for forecasting the solar activity," *Space Weather*, vol. 16, no. 9, pp. 1424–1436, 2018.

[106] N. Rigozo, M. Souza Echer, H. Evangelista, D. Nordemann, and E. Echer, "Prediction of sunspot number amplitude and solar cycle length for cycles 24 and 25," *Journal of Atmospheric and Solar-Terrestrial Physics*, vol. 73, no. 11, pp. 1294–1299, 2011. Influence of Solar Activity on Interplanetary and Geophysical Phenomena.

[107] Space Weather Prediction Center (SWPC) at National Oceanic and Atmospheric Administration (NOAA), "Solar Flares (Radio Blackouts)." `https://www.swpc.noaa.gov/phenomena/solar-flares-radio-blackouts`. [Online; accessed 10-September-2023].

[108] D. Dobrijevic, "Coronal mass ejections: What are they and how do they form?." `https://www.space.com/coronal-mass-ejections-cme`, June 2022. [Online; accessed 10-September-2023].