High Integrity Systems Project Time Series Analysis

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Assignment 3

Chitra Khatri
Hellyben Shah
Aniket Nighot
Mehjabeen Khan
Hardikumar Khunt

November 12, 2024

1 Summary of Modern Time Series Forecasting with Python- Chapter 4

After understanding some important concepts about Data Pre-Processing and Exploratory Data Analysis in previous chapters, this chapter introduces the most crucial step - time series forecasting.

Baseline Forecast

A baseline is a simple model that provides reasonable results without requiring much time to develop them. By simple model, what is meant here is not just something derived from common sense (average or any rule of thumb), but it can be very sophisticated as long as it is easy and quick to implement. Baselines should be strong enough as any further progress we want to make will be in terms of the performance of this baseline.

Setting up a Test Harness

Before setting up baselines, we need to set up a test harness. In terms of machine learning, a test harness is a set of code and data that can be used to evaluate algorithms. It is important to set up a test harness so that we can evaluate all future algorithms in a standard and quick way.

Splitting Dataset

As a standard practice, we divide our dataset into three parts to assess the accuracy and generalization ability of models:

1. **Training dataset:** to train the model
2. **Validation dataset:** to assess the quality of the model, to select between different model classes, tune the hyperparameters, perform feature selection etc.
3. **Test dataset:** to test how well your model is doing in unseen data.

Time series data presents unique challenges when it comes to model validation. Unlike traditional datasets, time series data must be split in a way that preserves chronological order and prevents data leakage.

- **Time Series Split:** This reliable method divides data into sequential folds, ensuring each training set is formed from past data and each test set from future data. It is particularly useful for maintaining the temporal structure of your data [1].
- **Sliding/Rolling Window Split:** This approach moves forward in time with a fixed-size training window that slides along the dataset. It is useful when you want to maintain a consistent training set size throughout the validation process [1].
- **Expanding Window Split:** In this method, the model starts with a small training set and gradually includes more observations as it progresses. This technique is effective for capturing long-term trends as it allows the model to learn progressively from more data points [1].

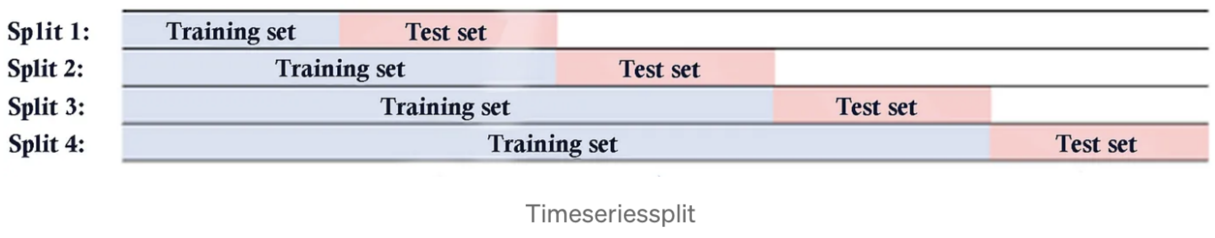


Figure 1: Time Series Split [1]

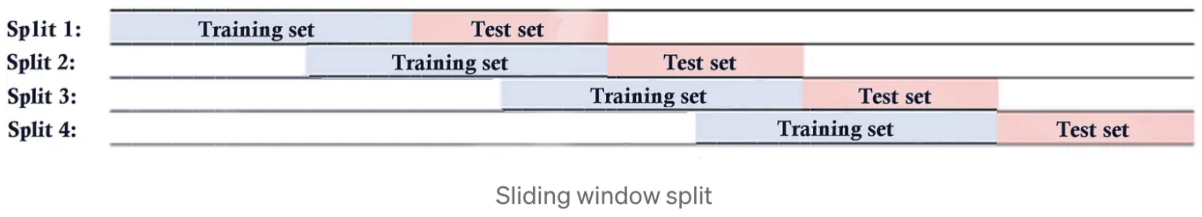


Figure 2: Sliding/Rolling Window Split [1]

- **Sliding Window with Gap Split:** This method introduces a buffer zone between the training and validation sets, ensuring no information from the future leaks into the model's training. It is particularly useful for simulating real-world forecasting scenarios where there might be a gap between training and prediction periods [1].
- **Group Time Series Split:** This scikit-learn compatible version of time series validation is ideal for non-overlapping groups .It ensures that the training and test sets do not overlap, which is crucial for time series data to avoid data leakage [1].

Choosing the Right Technique

The choice of splitting technique depends on your specific use case and the nature of your data such as the length of your time series, the presence of seasonality, and the specific forecasting task at hand. It's important to experiment with different methods to find what works best for your time series forecasting needs. Each technique has its strengths, the expanding window split is particularly effective for capturing long-term trends, while the sliding window approach works well for managing computational resources with a fixed-size training set [1].

Evaluation Metric

Mean Absolute Error (MAE)

MAE is a straightforward and intuitive metric that measures the average magnitude of errors in a set of predictions, without considering their direction.

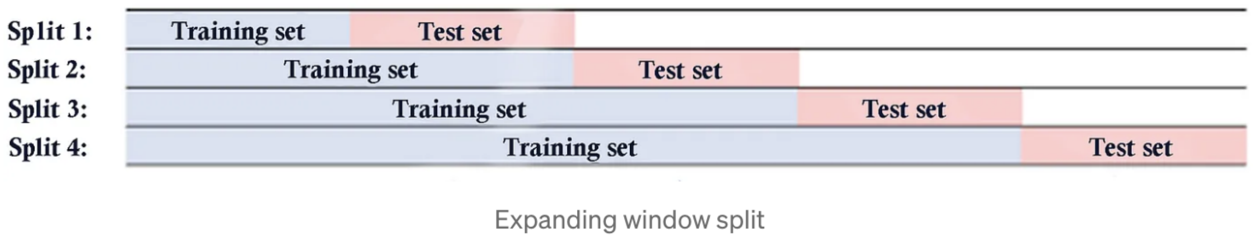


Figure 3: Expanding Window Split [1]

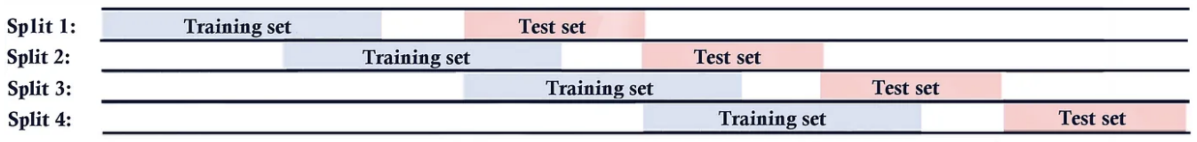


Figure 4: Sliding Window with Gap Split [1]

Formula:

$$MAE = \frac{1}{N \times L} \sum_{i=1}^{N} \sum_{j=1}^{L} |f_{i,j} - y_{i,j}| \tag{1}$$

Where:

- N is the number of time series; L is the length of the time series (test period); $f_{i,j}$ is the forecast value; $y_{i,j}$ is the observed value

Advantages:

- Easy to understand and interpret
- Less sensitive to outliers compared to MSE

Limitations:

- Scale-dependent, making it difficult to compare across different scales

Mean Squared Error (MSE)

MSE measures the average of the squares of the errors. It gives higher weight to larger errors due to the squaring.

Formula:

$$MSE = \frac{1}{N \times L} \sum_{i=1}^{N} \sum_{j=1}^{L} (f_{i,j} - y_{i,j})^2 \tag{2}$$

Advantages:

- Penalizes larger errors more heavily
- Useful when large errors are particularly undesirable

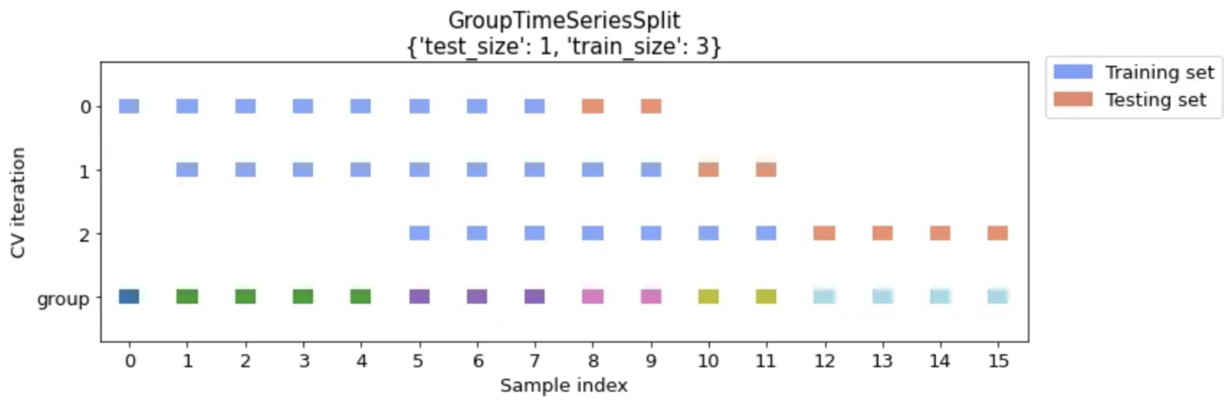


Figure 5: Group Time Series Split [1]

Limitations:

- More sensitive to outliers than MAE
- Scale-dependent
- Less interpretable due to squared units

Mean Absolute Scaled Error (MASE)

MASE is a scale-free error metric that compares the forecast errors to the errors of a naïve forecast.

Formula:

$$MASE = \frac{\frac{1}{L}\sum_{i=1}^{L}|f_i - y_i|}{\frac{1}{L-1}\sum_{j=2}^{L}|y_j - y_{j-1}|} \tag{3}$$

Advantages:

- Scale-independent, allowing comparison across different time series
- Interpretable: MASE < 1 indicates the forecast is better than the naïve method
- Handles time series with zeros or near-zero values well

Limitations:

- Can be undefined if the naïve method produces perfect forecasts
- May be less intuitive for non-experts

Forecast Bias (FB)

Forecast Bias measures the tendency of a model to consistently over- or under-forecast.

Formula:

$$FB = \frac{\sum_{i=1}^{N}\sum_{j=1}^{L} f_{i,j} - \sum_{i=1}^{N}\sum_{j=1}^{L} y_{i,j}}{\sum_{i=1}^{N}\sum_{j=1}^{L} y_{i,j}} \tag{4}$$

Advantages:

- Provides insight into systematic errors in the forecast
- Expressed as a percentage, making it easy to interpret

Limitations:

- Does not capture the magnitude of errors, only the direction
- Can be misleading if positive and negative errors cancel out

Choosing the Right Metrics

When evaluating time series forecasts, it's often beneficial to use a combination of these metrics:

1. Use MAE or MSE to get an overall sense of forecast accuracy.
2. Use MASE to compare performance across different time series or against a naïve benchmark.
3. Use Forecast Bias to check for systematic over- or under-forecasting.

Generating strong baseline forecasts

For this section, we are going to use a python library - *Darts*. Darts is used for time series forecasting and identifying anomalies. Further details about the useful methods of this library is discussed in the code section.

Below few simple methods forecasting is listed

Naïve Forecast

The naïve method, also known as the persistence forecast, is the simplest forecasting technique:

- It uses the last observed value as the forecast for all future time steps.
- Mathematically, it can be expressed as:

$$F_{t+h} = Y_t$$

  Where F_{t+h} is the forecast for h steps ahead, and Y_t is the last observed value.

Advantages:

- Extremely simple to implement
- Surprisingly effective for some financial time series (e.g., stock prices)

Limitations:

- Doesn't account for trends or seasonality
- Sensitive to outliers and noise in the last observation

Seasonal Naïve Forecast

This method extends the naïve approach to account for seasonality:

- It uses the last observed value from the same season as the forecast.
- Mathematically:

$$F_{t+h} = Y_{t-m+h_m}$$

  Where m is the seasonal period, and $h_m = [(h-1) \bmod m] + 1$

Advantages:

- Simple yet accounts for seasonality
- Effective for strongly seasonal time series with minimal trend

Limitations:

- Doesn't account for trends
- Assumes perfect seasonality (same pattern repeats exactly)

Moving Average Forecast

The mean method uses the historical average as the forecast:

- It calculates the mean of all past observations and uses this as the forecast for all future time steps.
- Mathematically:

$$F_{t+h} = \frac{1}{T}\sum_{i=1}^{T} Y_i$$

  Where T is the total number of past observations.

Advantages:

- Provides a single, easy-to-understand central tendency measure
- Works well for stationary time series with no clear trend or seasonality

Limitations:

- Sensitive to outliers, which can significantly skew the forecast
- Doesn't account for trends or seasonality
- Can be slow to adapt to changes in the time series

Exponential smoothing (ETS)

ETS combines näive method and moving average and says that the history is important but the recent history is more important. it uses weighted averages of past observations, with weights decreasing exponentially as observations get older.

Simple Exponential Smoothing (SES)

SES is the most basic form of exponential smoothing and is suitable for time series without clear trends or seasonality. The forecast equation for SES is:

$$F_{t+1} = \alpha Y_t + (1 - \alpha)F_t \tag{5}$$

Where:

- F_{t+1} is the forecast for the next period; Y_t is the actual value at time t; F_t is the forecast for the current period; α is the smoothing parameter $(0 < \alpha < 1)$

The α parameter determines how much weight is given to recent observations versus older ones. A higher α gives more weight to recent data.

Double Exponential Smoothing (DES)

DES, also known as Holt's linear trend method, extends SES to handle time series with trends. It uses two smoothing equations:

Level equation:

$$L_t = \alpha Y_t + (1 - \alpha)(L_{t-1} + T_{t-1}) \tag{6}$$

Trend equation:

$$T_t = \beta(L_t - L_{t-1}) + (1 - \beta)T_{t-1} \tag{7}$$

The forecast is then calculated as:

$$F_{t+h} = L_t + hT_t \tag{8}$$

Where:

- L_t is the estimate of the level at time t; T_t is the estimate of the trend at time t; β is the trend smoothing parameter $(0 < \beta < 1)$; h is the number of periods to forecast ahead

Damped Trend Method

To prevent over-forecasting in the long term, a damping factor ϕ can be introduced:

$$F_{t+h} = L_t + (\phi + \phi^2 + ... + \phi^h)T_t \tag{9}$$

Where $0 < \phi < 1$. As h increases, the trend component approaches a constant.

Triple Exponential Smoothing (Holt-Winters)

Holt-Winters method extends DES to include seasonality. There are two variations:

- Additive seasonality
- Multiplicative seasonality

For the additive method, the equations are:

Level:

$$L_t = \alpha(Y_t - S_{t-m}) + (1-\alpha)(L_{t-1} + T_{t-1}) \tag{10}$$

Trend:

$$T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1} \tag{11}$$

Seasonal:

$$S_t = \gamma(Y_t - L_t) + (1-\gamma)S_{t-m} \tag{12}$$

The forecast is then:

$$F_{t+h} = L_t + hT_t + S_{t-m+h_m} \tag{13}$$

Where:

- m is the length of the seasonal cycle; γ is the seasonal smoothing parameter ($0 < \gamma < 1$); $h_m = [(h-1) \bmod m] + 1$

For multiplicative seasonality, the equations are slightly different, with the seasonal component being multiplied rather than added.

Advantages:

- Computationally efficient
- Can handle various time series patterns (level, trend, seasonality)
- Often provide competitive forecasts for many types of data

Limitations:

- Assumes future patterns will resemble past patterns
- May not capture complex, non-linear relationships in the data
- Can be sensitive to outliers and can not perfectly model sudden changes (Peaks) in the time series

Arima Model

ARIMA model is a class of linear models that utilizes historical values to forecast future values. ARIMA stands for Autoregressive Integrated Moving Average, each of which technique contributes to the final forecast. Let's understand it one by one.

Autoregressive (AR)

In an autoregression model, we forecast the variable of interest using a linear combination of past values of that variable. The term autoregression indicates that it is a regression of the variable against itself. That is, we use lagged values of the target variable as our input variables to forecast values for the future. An autoregression model of order p will look like:

mt = 0 + 1mt-1 + 2mt-2 + 3mt-3+. . . + pmt-p

In the above equation, the currently observed value of m is a linear function of its past p values. [0, p] are the regression coefficients that are determined after training. There are some standard methods to determine optimal values of p one of which is, analyzing Autocorrelation and Partial Autocorrelation function plots.

I (Integrated):

This part makes the time series stationary by differencing the data. Stationarity means the statistical properties of the series (like mean and variance) are constant over time. If a series is not stationary, differencing helps by subtracting the previous value from the current value until the series becomes stationary.

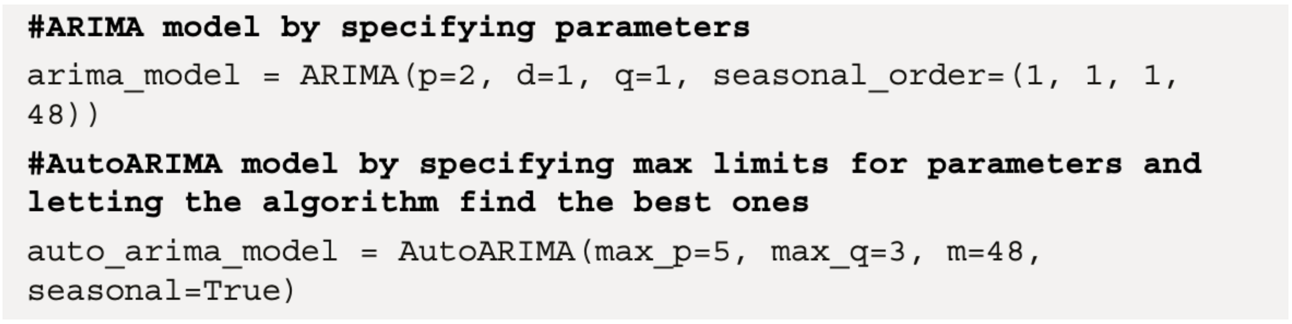
MA (Moving Average):

This part uses the dependency between an observation and a residual error from a moving average model applied to lagged observations. This component helps to smooth out short-term fluctuations.

Steps in Using an ARIMA Model:

1.Identify Stationarity: Before using ARIMA, ensure the series is stationary. If it's not, apply differencing (Integrated component) to achieve this.
2.Model Selection: Use techniques like the autocorrelation function (ACF) and partial autocorrelation function (PACF) plots to identify the values of AR and MA components.
3.Parameter Estimation: Fit the ARIMA model to the historical data using techniques like maximum likelihood estimation.
4.Validation: Check the model's performance by analyzing residuals (the difference between actual and predicted values) to ensure they resemble white noise (i.e., no autocorrelation).
5.Forecasting: Use the model to project future values based on the historical pattern identified.

Let's see how we can apply ARIMA and AutoARIMA using darts:



```
#ARIMA model by specifying parameters
arima_model = ARIMA(p=2, d=1, q=1, seasonal_order=(1, 1, 1,
48))
#AutoARIMA model by specifying max limits for parameters and
letting the algorithm find the best ones
auto_arima_model = AutoARIMA(max_p=5, max_q=3, m=48,
seasonal=True)
```

Figure 6: Arima and Auto Arima

Why ARIMA Works Well for Baseline Forecasts

The ARIMA model is effective for baseline forecasts because it:

- Captures Trends and Patterns in a time series without external data, making it a straightforward approach for projecting forward.
- Adapts to Different Time Series by tuning parameters (p, d, q), where p is the order of the AR term, d is the degree of differencing, and q is the order of the MA term.
- Handles Non-Stationarity through differencing, making it versatile for various types of time series.

The ARIMA model provides a strong basis for making informed projections, even when external factors or significant changes are not included in the analysis.

***NOTE**: The corresponding code for ARIMA Model is added in the shared Github repository*

Associated Metrics for Evaluating Baseline Forecasts

Metrics help quantify the accuracy of baseline forecasts and determine if they are sufficient or if adjustments are needed. Common metrics include:

- **Mean Squared Error (MSE):** Squares each error before averaging, placing extra emphasis on larger errors. This metric helps reveal cases where the baseline model struggles with significant deviations from actual values.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{14}$$

- **Root Mean Squared Error (RMSE):**Root Mean Squared Error (RMSE): The square root of MSE, RMSE provides error measurements in the same units as the data, making it more interpretable than MSE.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{15}$$

- **Mean Absolute Percentage Error (MAPE):** Expresses errors as a percentage of actual values, allowing for easy comparison across different scales. This is especially useful in understanding the forecast's performance across different magnitudes.
- **Bias:** Measures the tendency of forecasts to over- or underestimate values. Calculated as the average of all forecast errors, a near-zero bias indicates an unbiased forecast.
- **R-squared ($\mathbf{R^2}$):** Although more common in regression analysis, R^2 can be useful in understanding how much variance in data is explained by the baseline model. Higher values indicate a closer fit to the actual data.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{16}$$

Theta Forecast

Theta forecast is a simple but powerful forecasting method commonly used for time series data. It is known for its accuracy and ease of interpretation.

How Theta Forecast Works

1. **Theta Lines:** The original data is transformed into multiple *theta lines* by adjusting the curvature or trend. The theta parameter smooths local curvature, depending on the chosen theta value.
   - Higher θ = More smoothing, less local detail.
   - Lower θ = Less smoothing, more local detail.
2. **Decomposition:** The data is split into two main parts:
   - **Trend:** The overall direction of the data trend (e.g., upward, downward).
   - **Seasonality:** Patterns around the trend.

   This step creates a new deseasonalized time series.
3. **Single Exponential Smoothing (SES):** SES is applied to the deseasonalized time series to estimate the smoothing parameter α.
   - **Smoothing Parameter (α):** Controls the level of smoothing:
     - Higher α: More weight on recent data points.
     - Lower α: More weight on older data points, making the series smoother and less sensitive to recent changes.
   - **Season Mode and Seasonality Period:** Used for initial seasonal decomposition. These parameters can be set with domain knowledge, or the implementation can auto-detect seasonality.

Theta Forecast Advantages and Implementation

- **Advantages:**
  - Robust against noise and anomalies.
  - Simple to implement, often outperforming complex models.
  - Effective in handling various time series types with trend and seasonality.
- **Implementation:** Use libraries like Darts for easy Theta forecast implementation, tuning parameters for optimal results.

Fast Fourier Transform (FFT) Forecast

The Fast Fourier Transform (FFT) is a powerful tool for time series forecasting, helping to identify and analyze periodic components for more accurate forecasts.

How FFT is Applied

- **Transforming Data:** Apply FFT to transform the data from the time domain to the frequency domain.
- **Identifying Significant Frequencies:** Analyze the frequency spectrum to identify dominant frequencies.
- **Filtering Noise:** Remove noise by filtering out less significant frequencies.
- **Inverse FFT:** Transform the filtered frequency data back to the time domain, resulting in a smoothed version of the series.

Forecasting

Use the smoothed time series data to make forecasts based on identified periodic components.

Process

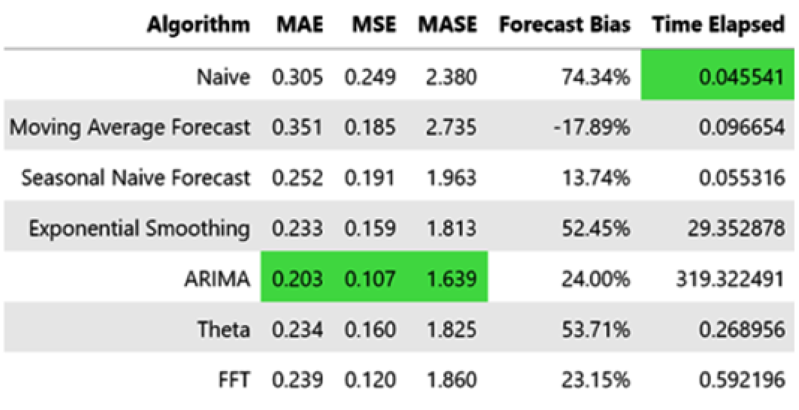
1. **Fourier Transform:** Converts the time series into sinusoidal components.
2. **Frequency Domain Analysis:** Identifies significant frequencies.
3. **Filtering:** Filters out noise by retaining significant frequencies.
4. **Inverse Fourier Transform:** Converts data back to the time domain, yielding a smoothed series.

Advantages

- Effective in isolating periodic patterns.
- Useful for time series with strong seasonal components.

Summary of Baseline Models

Among the baseline algorithms tested, ARIMA performs best on MAE and MSE but takes more time. The next best algorithms are Theta, ETS, and FFT, with Theta and FFT much faster than ETS.

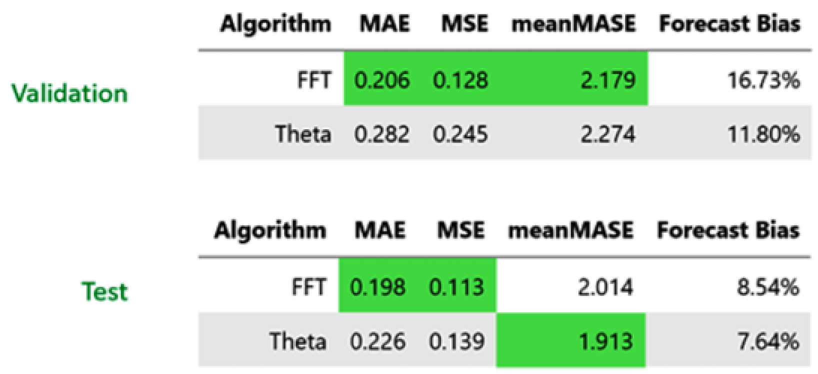


Algorithm	MAE	MSE	MASE	Forecast Bias	Time Elapsed
Naive	0.305	0.249	2.380	74.34%	0.045541
Moving Average Forecast	0.351	0.185	2.735	-17.89%	0.096654
Seasonal Naive Forecast	0.252	0.191	1.963	13.74%	0.055316
Exponential Smoothing	0.233	0.159	1.813	52.45%	29.352878
ARIMA	0.203	0.107	1.639	24.00%	319.322491
Theta	0.234	0.160	1.825	53.71%	0.268956
FFT	0.239	0.120	1.860	23.15%	0.592196

Figure 4.9 – Summary of all the baseline algorithms

Evaluating Baseline Forecasts

FFT performs well across all metrics. Since we have forecasts from both Theta and FFT, evaluation of these forecasts is also essential.



Validation

Algorithm	MAE	MSE	meanMASE	Forecast Bias
FFT	0.206	0.128	2.179	16.73%
Theta	0.282	0.245	2.274	11.80%

Test

Algorithm	MAE	MSE	meanMASE	Forecast Bias
FFT	0.198	0.113	2.014	8.54%
Theta	0.226	0.139	1.913	7.64%

Figure 4.10 – The aggregate metrics of all the selected households (both validation and test)

Forecastability of a Time Series

Assessing the forecastability of a time series is crucial for understanding how predictable a time series is and setting realistic expectations for the performance of forecasting models. This section explores two primary measures for assessing forecastability: Coefficient of Variation (CoV) and Residual Variability (RV).

Coefficient of Variation (CoV)

- **Definition:** o CoV is the ratio of the standard deviation to the mean of the time series. It provides a normalized measure of dispersion around the mean, making it easier to compare the variability of series with different units or scales.
- **Formula:** CoV = $\frac{\sigma}{\mu}$ where σ is standard deviation and μ is the mean.
- **Advantages** Simple to Compute: CoV is straightforward to calculate and interpret. Normalized Measure: By normalizing the standard deviation, CoV allows for the comparison of time series with different units or scales.
- **Limitation** Ignores Seasonality: CoV does not account for seasonal patterns in the data. For example, a sine wave with regular periodic fluctuations will have a higher CoV compared to a flat line, even though both are equally predictable. Ignores Trend: CoV does not consider the trend in the time series. A series with a strong upward or downward trend will have a higher CoV, even if it is predictable. Negative Values: If the time series contains negative values, the mean can become small or negative, causing the CoV to become inflated or undefined.

Residual Variability and Forecastability in Time Series Analysis

Residual Variability (RV)

1. **Motivation:**

   To overcome the limitations of CoV, RV focuses on the variability of the residuals after removing trend and seasonality from the time series. The underlying assumption is that trend and seasonality are predictable components, while the residuals represent the unpredictable part of the series.

2. **Calculation Process:**

   (a) **Seasonal Decomposition:**

   Decompose the time series into three components: trend, seasonality, and residuals. This can be done using methods like STL (Seasonal and Trend decomposition using Loess).

   $$\text{Time Series} = \text{Trend} + \text{Seasonality} + \text{Residuals}$$

   (b) **Standard Deviation of Residuals:**

   Calculate the standard deviation of the residuals.

   (c) **Scaling:**

   Scale the standard deviation of the residuals by the mean of the original time series to get RV.

   $$\text{RV} = \frac{\sigma_{\text{residuals}}}{\mu_{\text{original}}}$$

3. **Advantages:**

   - Considers Seasonality and Trend: By focusing on the residuals after decomposing the time series, RV accounts for both seasonality and trend, providing a more accurate measure of forecastability.
   - Normalized Measure: Like CoV, RV is also a normalized measure, allowing for comparison across different time series.
   - More Reliable: RV provides a more reliable assessment of forecastability, especially for time series with strong seasonal or trend components.

4. **Implementation:**

   - Use a decomposition method (e.g., STL) to separate the time series into trend, seasonality, and residuals.
   - Calculate the standard deviation of the residuals and scale it by the mean of the original series.
   - Implement RV calculation in a systematic way for all time series in the dataset to compare their forecastability.

Practical Application

1. **Comparing Time Series:**

   Use CoV and RV to compare different time series and identify which ones are more predictable. Lower values of CoV and RV indicate higher forecastability.

2. **Setting Expectations:**

   Use the forecastability measures to set realistic expectations for the performance of forecasting models. Time series with low forecastability (high CoV or RV) may require more sophisticated models or may inherently have higher forecast errors.

3. **Prioritizing Forecasting Efforts:**

   Focus forecasting efforts and resources on time series with higher forecastability (low CoV or RV) for better results. Use simpler models for these series and invest in more complex models for those with lower forecastability.

Example Calculation

- For a given time series, perform STL decomposition to obtain trend, seasonal, and residual components.
- Calculate the standard deviation of the residuals.
- Compute the mean of the original time series.
- Use the formula

  $$\text{RV} = \frac{\sigma_{\text{residuals}}}{\mu_{\text{original}}}$$

  to get the RV value.
- Compare the RV values across multiple time series to assess their relative forecastability.

Entropy-based Measures

Entropy-based measures assess the degree of unpredictability, disorder, or complexity in a time series. These measures are grounded in the concept of entropy from information theory, which quantifies the uncertainty of random variables. For time series data, entropy can reveal insights about the underlying dynamics, signal complexity, and randomness.

Spectral Entropy

Spectral entropy is an entropy-based measure derived from the power spectral density (PSD) of a time series. It combines information theory and spectral analysis, offering a way to measure the complexity of a time series in the frequency domain.

- **Computation:**

  Spectral entropy is computed by first transforming the time series into the frequency domain (e.g., using the Fourier transform) to obtain the PSD. This spectrum is then normalized to create a probability distribution, from which the Shannon entropy is calculated.

- **Applications:**

  It is often used in fields like neuroscience (e.g., for EEG signals), economics, and engineering to identify states of the system (such as sleep stages or market regimes) by measuring signal complexity. Low spectral entropy indicates a more predictable and less complex signal, while high spectral entropy indicates a more complex and less predictable signal.

Kaboudan Metric

The Kaboudan metric is a statistical measure used in time series analysis to assess forecast accuracy and signal characteristics. It was introduced by **Mak Kaboudan** and focuses on capturing both predictability and similarity of time series data.

- **Purpose:**

  This metric is designed to evaluate the quality of time series forecasts or models by comparing their structure with the actual data. It is particularly useful in contexts where capturing the specific shape or sequence of time series data points is crucial.

- **Method:**

  The Kaboudan metric uses a combination of similarity measures (such as Euclidean distance) and alignment techniques to quantify the closeness of two time series. It's often employed in financial markets and other fields where pattern recognition and forecast similarity are important.

Each of these methods provides unique insights into time series behavior. Entropy-based measures help understand the unpredictability and randomness, spectral entropy assesses complexity in the frequency domain, and the Kaboudan metric emphasizes the accuracy and structural similarity of time series models. These metrics are particularly valuable in applications where the goal is to model, predict, or classify time series based on their dynamic and complex properties.

2 Summary of Modern Time Series Forecasting with Python- Chapter 5

Understanding the basics of Machine learning

In traditional programming, we start with a known set of rules or logic to transform input data into a desired output. Essentially, we tell the computer exactly *how* to achieve the result based on predefined instructions.

Machine learning, on the other hand, reverses this approach. Instead of providing rules, we start with data and the desired output and ask the computer to *learn* the underlying patterns or rules that connect the data to the output. The machine, through training, discovers the relationships and develops a model that can apply these learned patterns to new data to make predictions or perform tasks.

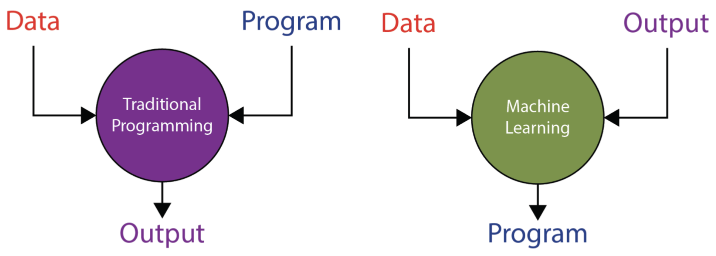


Figure 7: Traditional programming versus machine learning

- **Supervised Machine Learning**: Supervised machine learning is a type of machine learning where a model is trained on a labeled dataset, meaning each input data point is paired with a known output (label). The goal is for the model to learn the relationship between the input data and the output so that it can accurately predict the output for new, unseen data.

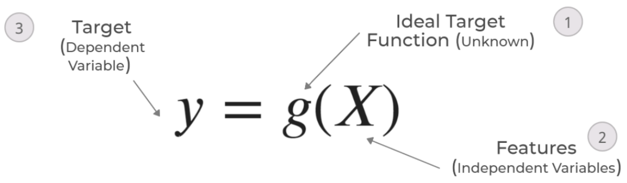

3
Target (Dependent Variable)
Ideal Target Function (Unknown)
1
y = g(X)
Features (Independent Variables)
2

Figure 8: Supervised Machine Learning- the ideal function

where X is the set of features and g is the ideal target function (denoted by 1 in Figure 2) that maps the X input (denoted by 2 in the figure) to the target (ideal) output, y (denoted by 3 in the figure).

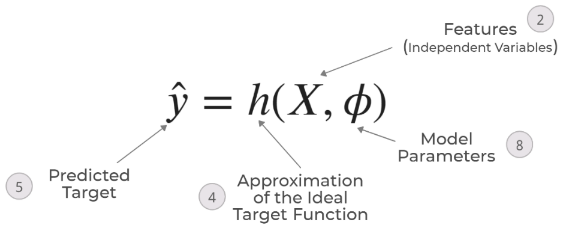

Features (Independent Variables)
2
ŷ = h(X, ϕ)
Model Parameters
8
5
Predicted Target
4
Approximation of the Ideal Target Function

Figure 9: Supervised Machine Learning- the learned approximation

This approximation of the ideal target function is denoted by another function, h (4 in the schematic), which takes in the same set of features, X, and outputs a predicted target, $\hat{y}$ (5 in the schematic). Φ are the parameters of the h function (or model parameters). In supervised learning, we start with a dataset consisting of features (X) and targets (y), which are also known as labels. The goal is for the machine to learn the function h that approximates the relationship between the features and the target. The model's parameters (denoted as Φ) are adjusted to make the function h produce accurate predictions.

In this process, the ideal target function (g) is unknown, but we use the training data D to generate predicted targets for each data point. The accuracy of these predictions is measured by comparing the predicted values to the actual targets. This comparison is done using a loss function, which quantifies how far off the predictions are from the actual values.

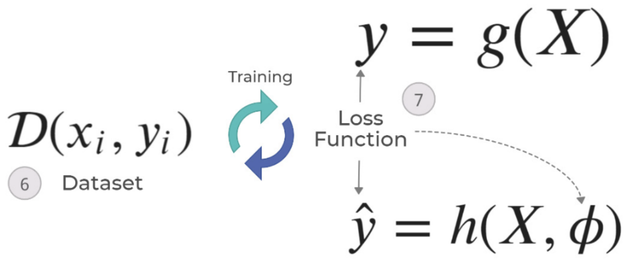


Figure 10: Supervised Machine Learning- putting it all together

The function h can be chosen from a set of known functions, called H, such as linear functions or decision tree-based functions. This set of functions represents the types of models we can use. The process of selecting an appropriate function h from the set H involves training the model using the dataset.Now, the remaining task is to test various functions to identify the best approximation function, h, that minimizes the loss. This process is known as training, and it is essentially an optimization task.

Supervised Machine learning- Regression and Classification

Machine learning can address various tasks like regression, classification, and recommendation. However, since classification and regression are the most common types of problems, we will briefly review them.

- **Classification:** In machine learning, Classification is a type of machine learning task where the goal is to predict a category or class label for a given input.The output is discrete, meaning it belongs to one of several predefined classes or categories. Example : Email spam detection: The model predicts whether an email is "spam" or "not spam" (binary classification).
- **Regression:** Regression is a machine learning task where the goal is to predict a continuous value based on input features.The output is continuous, meaning it can take any value within a range. Example : Predicting house prices: The model predicts the price of a house based on features like size, location, and number of rooms.

Overfitting and Underfitting

- **Generalization:** In machine learning, generalization is a model's ability to perform well on new, unseen data. Unlike traditional optimization, where the goal is to find the maximum or minimum in the training dataset, machine learning aims to achieve low test error by using training error as a guide.
- **Underfitting:** Underfitting occurs when a model doesn't learn enough from the training data, leading to high errors on both training and test data.
- **Overfitting:** Overfitting, on the other hand, happens when a model memorizes the training data without understanding underlying patterns, resulting in very low training error but high test error.

  A **model's capacity** is its flexibility to fit different functions, influencing whether it underfits or overfits. Low-capacity models may underfit by failing to capture data patterns, while high-capacity models risk overfitting by memorizing the data too closely. For instance, moving from linear to polynomial regression increases capacity, allowing the model to fit more complex curves. Achieving a balance between capacity and the problem at hand helps a model generalize well.

- **Regularization:** Regularization is a technique used to guide the model toward simpler solutions, even when it has the same capacity. It does this by adding constraints during the learning process to reduce the complexity of the learned function. This preference for simpler functions is often achieved through weight decay, which discourages overly large weights in the model.

  In the case of linear regression, the model can be represented as:

  $$\hat{y} = c + \sum_{i=1}^{N} w_i \cdot x_i$$

  where N is the number of features, c is the intercept, x_i is the i-th feature, and w_i is the weight associated with that feature. We find the optimal weight L by treating it as an optimization problem that minimizes the difference between the predicted output $\hat{y}$ and the actual output y.

  With regularization, we add an extra term to the loss function L that encourages the weights to be smaller. This is typically done using L1 or L2 regularization. In L1 regularization, the sum of squared weights is added to the loss function.

  $$L + \lambda \sum_{i=1}^{N} w_i^2 \quad \text{(L1 regularization)}$$

  while in L2 regularization, the sum of the absolute weights is added. The regularization strength is controlled by the coefficient λ.

  $$L + \lambda \sum_{i=1}^{N} |w_i| \quad \text{(L2 regularization)}$$

  In both cases, regularization encourages smaller weights to prevent the model from relying too heavily on any single feature, promoting a more balanced and generalizable function.

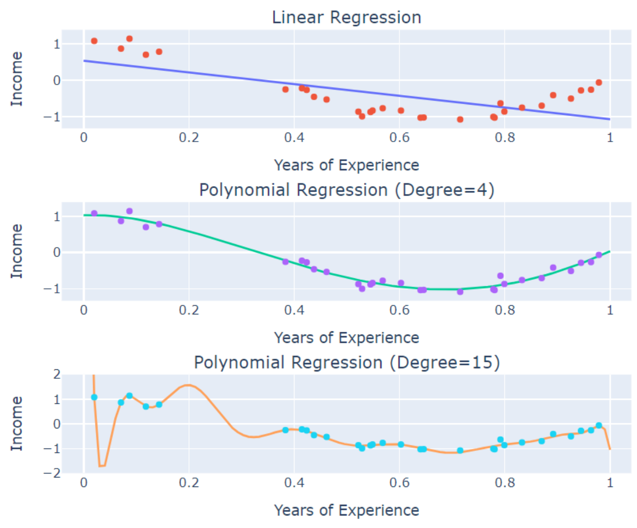


Figure 11: Underfitting versus overfitting

HyperParameters and Validation Sets

Most machine learning models come with hyperparameters—settings that are defined before training rather than learned from the data. For example, the regularization weight is a hyperparameter. These parameters allow control over model capacity and regularization, helping to balance between underfitting and overfitting.

To determine the best hyperparameters, we rely on data; however, if we only use the training data, it will tend to select maximum capacity, leading to overfitting. Therefore, a validation set, which is separate from the training data, is used to optimize hyperparameters without biasing the model. For smaller datasets, a single validation set may not provide a reliable evaluation, so cross-validation is commonly used. In k-fold cross-validation, the dataset is divided into k non-overlapping, equal subsets. The model is trained on $k-1$ subsets and validated on the remaining one, with this process repeated across all subsets to improve robustness. This approach will later be explored in the context of time series, where cross-validation has unique considerations.

Time series forecasting as regression

Time series forecasting involves predicting future observations based on historical data collected sequentially over time. Unlike regression, which predicts a continuous variable from independent, identically distributed (IID) examples, forecasting relies on a sequence of dependent data points, making it an extrapolation problem rather than interpolation.

In traditional regression, models are trained with independent examples of inputs and outputs, which is incompatible with time series data due to its sequential dependencies. Additionally, regression methods assume IID samples, while time series observations violate this assumption because they are interdependent.

To leverage machine learning techniques for time series forecasting, we can transform the problem into a regression-compatible format. This is done by creating features that introduce memory to the model, allowing it to account for the dependencies in time series data.

Time delay embedding

In time series forecasting, instead of conditioning each prediction on all past observations (which may be impractical due to potentially long histories), we often use only the most recent **M observations**, where **M** $<$ **L** (L is the full history length). This approach, called a **finite memory model** or **Markov model**, assumes a fixed-length sliding window of size **M**. Using this window, we create subsequences by moving it one step backward in time, capturing each set of M observations as features (lags) and the next time step as the target.

For example, with a memory size of 3, we take three recent observations (lags) as input features for predicting the next time step. This process continues until we have a full dataset of feature-target pairs, making the time series compatible with standard regression models. Each feature is labeled by how many time steps it lags behind the target (e.g., Lag 1, Lag 2), encoding the **autoregressive structure** in a way suitable for machine learning.

Temporal embedding

In **temporal embedding models**, instead of using past observations in an autoregressive manner, we assume each value in a time series depends solely on time itself. These models derive features from timestamps to capture aspects like the passage of time, periodicity, and other time-based patterns, which are then used in regression models to predict future values. Methods range from simple time-based counters to advanced techniques like Fourier terms to model periodic patterns. This approach represents a shift in time series forecasting, focusing on time-based features rather than historical values, and is a key concept gaining traction in the field.

Global forecasting models – a paradigm shift

Traditionally, forecasting for time series data focused on analyzing each series in isolation, creating a model based on the individual series' historical data. However, with the growth of digital data collection, organizations now gather extensive, related time series data from similar sources. For instance, companies like Walmart, Uber, and energy providers collect data across multiple locations or consumers, leading to "related time series," which exhibit similar patterns or behaviors.

There are two main approaches to forecasting with this data. The **local approach** treats each time series as coming from its own unique data-generating process (DGP) and models each one separately. While this is straightforward, it can lead to **overfitting** due to limited data length, causing poor generalization. Traditional models often imposed strict assumptions to counteract this, risking underfitting and limited accuracy.

In contrast, the **global approach** assumes that all related time series are generated by a single DGP, enabling the use of a single forecasting model for all series. This approach increases the dataset's "width" by combining multiple time series, allowing machine learning models to utilize the added data to learn more complex patterns. Global models have shown significant success in competitions (e.g., Kaggle's Rossman Store

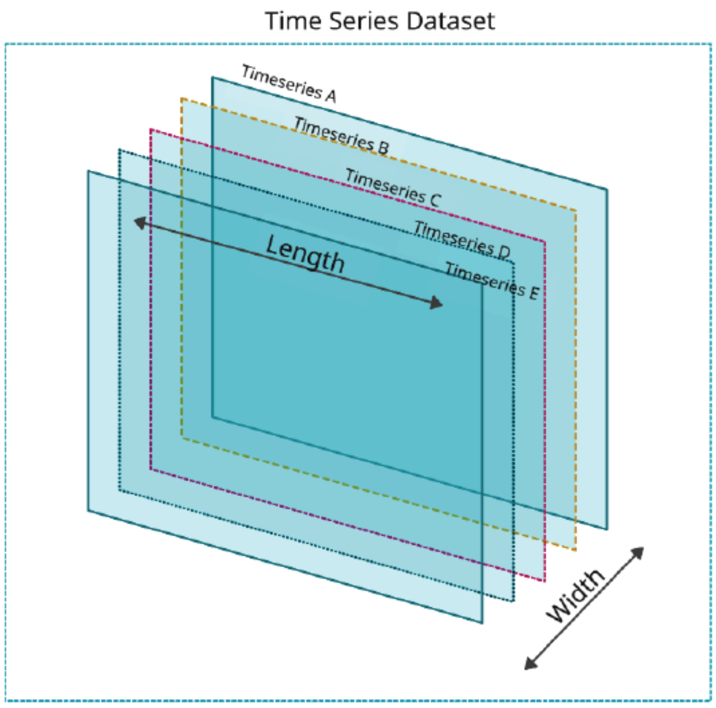


Figure 12: The length and Width of a timeseries dataset

Sales, Wikipedia WebTraffic, M5 Competition), consistently outperforming local models. Recent studies suggest that global models can approximate local models' accuracy with sufficient complexity and may even outperform them with unrelated time series. Although still a developing area, global models are proving to be a powerful alternative in time series forecasting.

3 Time Series Analysis in MATLAB

Introduction

In MATLAB, time series data can be handled using the `timeseries` object, which provides a convenient way to store, manipulate, and plot time series data. The `timeseries` object in MATLAB is designed specifically for time-stamped data, allowing you to manage your data efficiently. It supports various functions for accessing, processing, and visualizing time series data, making it easier to work with temporal datasets.

To create a `timeseries` object, you can use the following syntax:

```
ts = timeseries(data, time);
```

where `data` is the vector of values and `time` is the corresponding vector of time points. This object stores both the data values and their corresponding timestamps, enabling a wide range of time series operations, such as resampling, interpolation, and plotting.

Accessing MATLAB Documentation

You can access the documentation for time series and related functions in MATLAB in several ways. MATLAB offers built-in commands that allow you to search for and open the official documentation directly from the MATLAB command window. This documentation provides detailed information on the syntax, usage, and examples of time series functions, enabling you to quickly find guidance on various time series operations.

For example, to access the documentation for the `timeseries` object, you can enter the following command in the MATLAB command window:

```
doc timeseries
```

This command opens the MATLAB documentation page for the `timeseries` object, where you can explore its properties, methods, and examples of usage. Additionally, MATLAB provides the `help` command, which displays brief descriptions of functions and their syntax directly in the command window. For instance:

```
help timeseries
```

This command shows a summary of the `timeseries` function, allowing you to quickly review its basic usage.

Implementing ARIMA in MATLAB

To implement an ARIMA model in MATLAB, we will follow these steps:

1. First, we load or create our time series data. For example:

```
data = [time series data here];
```

2. Next, we define the ARIMA model. If we want to use an ARIMA(1,1,1) model, we write:

```
model = arima(1, 1, 1);
```

3. After defining the model, we estimate its parameters using our data:

```
estimatedModel = estimate(model, data);
```

4. Finally, we can forecast future values based on the estimated model. For instance, to forecast the next 10 time points, we use:

```
numForecastSteps = 10;
forecastedValues = forecast(estimatedModel, numForecastSteps, 'Y0', data);
```

Matlab application examples of the FFT

Overview

The Fast Fourier Transform (FFT) is a computational method for transforming a signal from the time domain to the frequency domain. FFT is widely used in signal processing to identify frequency components in applications like audio analysis, vibration monitoring, and filtering.

Basic MATLAB Functions for FFT

1. **fft**: Computes the Fast Fourier Transform.
   - **Syntax**: `Y = fft(X)`
   - **Parameters**:
     - `X`: The input time-domain signal (array of data points).
     - `Y`: The output of the FFT, representing complex frequency components.
2. **abs**: Computes the magnitude of complex numbers, used to obtain the magnitude spectrum of the FFT output.
   - **Syntax**: `M = abs(Y)`
3. **length**: Determines the number of data points in the signal, useful for setting up the frequency axis.
   - **Syntax**: `N = length(X)`

4. **Frequency Calculation**: To set up the frequency axis, divide the sampling rate Fs by the number of points N.

$$f = (0 : N - 1) \cdot \frac{Fs}{N} \tag{17}$$

   where f is the frequency axis.

5. **plot**: Plots the frequency spectrum.

   - **Syntax**: `plot(f, M)`
   - **Parameters**:
     - `f`: Frequency axis.
     - `M`: Magnitude spectrum of the signal.

Example Code in MATLAB

Below is a simple example to perform FFT in MATLAB.

```
data = [your time series data here]; % Load or define the data
Fs = 1; % Define the sampling frequency
N = length(data); % Number of data points
Y = fft(data); % Compute the FFT
f = (0:N-1)*(Fs/N); % Create frequency axis
M = abs(Y); % Compute the magnitude of the FFT
plot(f(1:floor(N/2)), M(1:floor(N/2))); % Plot first half of the spectrum
xlabel('Frequency (Hz)');
ylabel('Magnitude');
title('Magnitude Spectrum of the Signal');
```

Applications

FFT is useful in analyzing frequency components in various fields:

- **Audio Processing**: Identifying specific frequencies in sound signals.
- **Vibration Analysis**: Detecting mechanical vibrations to assess equipment health.
- **Signal Filtering**: Isolating specific frequency bands or removing unwanted noise.

E) Matlab Application Examples of the Wavelet Transform

Overview

The wavelet transform is a method for analyzing signals that change over time, breaking them down into both their frequency (what "pitches" or patterns exist) and time (when they happen) components. Unlike the Fourier transform, which only tells you the frequencies in a signal, the wavelet transform shows when these frequencies occur. This makes it especially useful for analyzing signals with bursts or changes, like heartbeats in an ECG, sudden sounds in audio, or faults in machinery vibrations..

Basic MATLAB Functions for Wavelet Transform

1. **cwt**: Computes the Continuous Wavelet Transform.
   - **Syntax**: `[cfs, freq] = cwt(X, 'WaveletName', Fs)`
   - **Parameters**:
     - `X`: The input time-domain signal (array of data points).
     - `WaveletName`: Specifies the mother wavelet type (e.g., 'morse', 'amor').
     - `Fs`: Sampling frequency of the signal.
     - `cfs`: Coefficients of the wavelet transform, representing both frequency and time information.
     - `freq`: Associated frequencies for each scale in the wavelet transform.
2. **abs**: Computes the magnitude of complex wavelet coefficients to get the magnitude spectrum.
   - **Syntax**: `M = abs(cfs)`
3. **plot3**: For a 3D plot of time, frequency, and coefficient magnitude.
   - **Syntax**: `plot3(time, freq, M)`
   - **Parameters**:
     - `time`: Time axis, derived from signal length and sampling frequency.
     - `freq`: Frequency axis, provided by the `cwt` function.
     - `M`: Magnitude spectrum of the wavelet coefficients.

Example Code in MATLAB

Below is an example code for performing the Continuous Wavelet Transform (CWT) in MATLAB.

```
data = [time series data here]; % Load or define the data
Fs = 1; % Define the sampling frequency
[cfs, freq] = cwt(data, 'morse', Fs); % Compute the CWT using 'morse' wavelet
M = abs(cfs); % Compute the magnitude of the CWT coefficients
time = (0:length(data)-1)/Fs; % Create time axis
scalogram('cwt', data, 'WaveletName', 'morse', 'SamplingFrequency', Fs); % Plot scalogram
xlabel('Time (s)');
ylabel('Frequency (Hz)');
title('Time-Frequency Representation using CWT');
```

Applications

Wavelet Transform is widely used in various fields where time-localized frequency information is essential:

- **Image Compression**: Used in JPEG2000 for efficient image compression.
- **Biomedical Signal Analysis**: Identifying non-stationary features in ECG, EEG, and other physiological signals.
- **Fault Detection in Machinery**: Analyzing vibration signals for early fault detection in rotating machinery.
- **Geophysics**: Analyzing seismic signals for event detection and characterization.

References

[1] M. En-nasiry, "Time series splitting techniques: Ensuring accurate model validation," Jun 2024. [Online]. Available: https://medium.com/@mouadenna/time-series-splitting-techniques-ensuring-accurate-model-validation-5a3146db3088