
Self-Supervised Learning for Time Series

A Review & Critique of FITS

Andreas Løvendahl Eefsen*
DTU
s224223@dtu.dk

Nicholas Erup Larsen
DTU
s224175@dtu.dk

Oliver Glozmann Bork Hansen
DTU
s194591@dtu.dk

Thor Højhus Avenstrup
DTU
s224233@dtu.dk

June 24, 2024

Abstract

Accurate time series forecasting is a highly valuable endeavour with applications across many industries. Despite recent deep learning advancements, increased model complexity, and larger model sizes, many state-of-the-art models often perform worse or on par with simpler models. One of those cases is a recently proposed model, FITS, claiming competitive performance with significantly reduced parameter counts. By training a one-layer neural network in the complex frequency domain, we are able to replicate these results. Our experiments on a wide range of real-world datasets further reveal that FITS especially excels at capturing periodic and seasonal patterns, but struggles with trending, non-periodic, or random-resembling behavior. With our two novel hybrid approaches, where we attempt to remedy the weaknesses of FITS by combining it with DLinear, we achieve the best results of any known open-source model on multivariate regression and promising results in multiple/linear regression on price datasets, on top of vastly improving upon what FITS achieves as a standalone model.

Our code developed and used for the project is available in the following GitHub Repository: github.com/thorhojhus/ssl_fts/

*Listing is in alphabetical order. We are especially thankful to our supervisor, Tommy Sonne Alstrøm, for sparking the inspiration to do time series, his thought-provoking questions that sparked new research directions, and his invaluable guidance and feedback throughout our project.

This document originated as a more comprehensive report, akin to a group bachelor thesis for a university project, which included sections on Stochastic Gradient Descent (SGD), choice of optimizer, Autoregressive Integrated Moving Average (ARIMA), and other topics. To adhere to the conventional format of papers posted on arXiv, the main content has been streamlined to focus primarily on novel or relevant information, with the additional sections cut or moved to the appendix.

1 Introduction

In this project, we aim to explore time series forecasting with the use of state of the art machine learning models. Time series analysis and forecasting is an important tool in many fields of science and business, which leads to an interest in developing more effective and precise time series forecasting models.

In the financial sector, forecasting stock prices, exchange rates, and economic indicators can bring a competitive edge to investors by managing risks and optimizing their portfolio returns. Healthcare providers can employ forecasting to predict diseases and take preventive measures. Additionally, the integration of sensor data in healthcare enables continuous monitoring of patient health, allowing for real-time analysis and more accurate predictions of medical conditions. And in the energy sector, forecasting electricity demand and renewable energy production is essential for grid stability and resource planning. Accurate forecasting is critical because electrical grids require a delicate balance between supply and demand. If the grid receives too much electricity, it can lead to overloading and potential damage to transmission lines, transformers, and other equipment. On the other hand, if there is insufficient electricity supply to meet the demand, it can result in power outages, blackouts, and disruptions to businesses and households. This can have massive negative cascading economic and social effects, as modern society relies heavily on a stable and reliable power supply.

The primary assertion of the FITS paper is that with just 10k-50k parameters, their method reaches comparable performance with previous state-of-the-art models of much larger parameter size, namely TimesNet (300.6M) [Wu+22b], Pyraformer (241.4M), FEDformer (20.68M), Autoformer (13.61M), and PatchTST (1M) [XZX24] on datasets from the aforementioned sectors. This is a substantial claim and we aim to reproduce their results and afterwards use the model on other datasets.

1.1 Time Series Data & Forecasting

A *time series* is a continuous or discrete series or sequence of observations over a time frame. A lot of real world data would conform to the format of a *time series*, examples are rainfall volumes over a given day or month, temperature over a year, sales of products over months, continuous sensor data, as well as changes in concentrations in chemical substances. There are examples of time series data abound in many fields including economics and business, chemistry and natural sciences, engineering as well as social sciences.

A time series, denoted Z_T , will be sampled such that the the discrete time series will have equidistant time points with a total of T observations, for which we denote the observed value at time $t \leq T$ as z_t .

For the forecasting task, the forecasted time series at time step $t + l > T$ is denoted as \hat{z}_{t+h} . The *forecast horizon*, also known as the *leading time*, is here denoted l . The goal is to minimize the error from the forecasted values to that of the real time series for example the mean squared error:

$$\min_l \frac{\sum_{t=k}^{k+l} (\hat{z}_{t+l} - z_{t+l})^2}{l} \quad (1)$$

The difficulty in forecasting varies greatly and depends on the type of time series one wishes to predict. Data with clear harmonics, trends and little volatility will be easier and more simple to forecast than more volatile and chaotic data such as stock prices and noisy data.

A time series can also express properties of *stationarity* and *seasonality*. These two properties are critical in helping to understand how to build a model for the problem.

Seasonality in Time Series

There exists an abundant amount of time series data, where patterns repeat in intervals. Populations of certain animals, the average temperature during a month and even sales of merchandise can show signs around holidays and other times of the year.

What they all share, is a *repeating* pattern. In literature, the usual letter denoting the length of the pattern, or seasonality length, is m ; a simple example would be a sine curve, whose seasonality would be $m = 2\pi$

Stationarity in Time Series

A *weakly* stationary time series exhibits no different behaviour when shifted in time. In simple probability terms, we require that the probability distribution over the time series should not depend on the time i.e., $f_{z|T=t}(z|T=t) = f_z(z)$.

In other words, if we write the time series as:

$$z_t = \mu + \epsilon_t + \psi_1 \epsilon_{t-1} + \psi_2 \epsilon_{t-2} + \dots$$

Then a requirement of stationarity is that the weights, ψ , have to be *absolutely summable* meaning $\sum |\psi| < \infty$, and as such, the parameter μ has a meaning denoting the level the time series varies about. The covariance between z_t and z_{t-j} should also be equal, and only depend on the relative time difference.[BJ76][Ham94]

1.2 Evolution of Forecasting Methods

Forecasting has undergone a significant transformation over time, evolving from simple statistical approaches to more sophisticated ML techniques and lately transformer-based models. This section provides an overview of the key milestones in the evolution of forecasting methods.

1.2.1 Statistical Forecasting

The late 19th and early 20th centuries saw the emergence of statistical forecasting methods. These techniques, such as moving averages, exponential smoothing, and linear regression, used historical data to identify patterns and make predictions. They provided a more systematic and data-driven approach compared to intuition-based methods. Specially the ARIMA model is of interest, and will be described in more detail in the appendix 8.

1.2.2 Machine Learning-based Models

With the advent of machine learning, forecasting methods have undergone a significant transformation. Machine learning algorithms, such as artificial neural networks, long short-term memory (LSTM) networks, random forests, and support vector regression, have been applied to forecasting tasks. These methods can handle complex patterns, nonlinearities, and large datasets, often outperforming traditional statistical methods.

1.2.3 Transformer-based Models

In recent years, Transformer-based models have emerged as a popular architecture surrounding ANNs, garnering significant attention in machine learning communities due to their unique ability to capture long-range dependencies and complex patterns in sequential data. The main breakthrough of Transformer-based models is the multi-headed attention mechanism which updates the weights of the model based on all input steps. This allows it to retain more information about how each step in the sequence affects each other. One of the major pitfalls of the traditional self-attention mechanism, however, is the $O(L^2)$ (L = input length) time and memory complexity when training models.

One of the first papers to use Transformer architecture on time series data was Informer [Zho+21]. Informer's main focus is improving the speed of inference, training, and efficiently handling long inputs. Among other things, it introduces a new "ProbSparse" self-attention mechanism, which instead achieves $O(L \log L)$ and works by focusing on a subset of the most informative parts of the input sequence by selecting only the top u queries that have the highest sparsity measurements. Informer achieved an average of ~ 1.6 MSE and ~ 0.79 MAE across all datasets.

Half a year later, Autoformer comes out [Wu+22a], claiming state-of-the-art accuracy with an average of ~ 0.86 MSE (48% lower) and ~ 0.53 MAE (33% lower) compared to Informer. Using a new technique dubbed Auto-Correlation, they instead operate at the series level while ProbSparse operates at the point/query level, and also uniquely uses self-attention distilling in the encoder by halving layer inputs. Furthermore, Autoformer introduces a series decomposition block which progressively extracts and aggregates the long-term stationary trend from predicted intermediate results, allowing it to also capture general patterns in long sequences.

FEDformer [Zho+22] hopes to also take the overall trend into account by combining the Transformer, capturing more detailed structures, with their seasonal-trend decomposition method and Frequency enhanced blocks, both capturing a more global view of the time series. Applying a Fourier transform to map the time series into the frequency domain had already been done in the 80s but this is the first

time we see it in combination with a transformer – and yielding superior results. FEDformer claims to further reduce the prediction error from Autoformer by 14.8 % and 22.6 % for both multivariate and univariate forecasting.

1.2.4 New frontiers

Using just a basic one-layer linear model directly mapping the input sequence to the output sequence, DLinear [Zen+22] claims Transformers are ineffective for time series forecasting. The linear models in this paper are simply linear layers (i.e., fully connected layers without non-linear activations) that map the input sequence directly to the output sequence. Linear is a vanilla linear model, NLinear first normalizes the input by subtracting the last value before applying a linear layer and adding the subtracted value back to handle distribution shift, and DLinear decomposes the input into trend and remainder components which are processed separately by linear layers before being summed for the final prediction.

However, in 2022, PatchTST [Nie+23] challenges this claim by slightly outperforming DLinear on most datasets. PatchTST introduces: (1) segmentation of time series into subseries-level patches as input tokens to the Transformer, and (2) channel-independence where each univariate time series is processed separately but shares the same embedding and Transformer weights. These modifications make PatchTST quite superior to other transformers and slightly better than using simple linear layers.

FITS (Frequency Interpolation Time Series Analysis Baseline) is a recently (2023) proposed model that leverages Fourier transforms in conjunction with a single complex-valued linear layer for time series forecasting. Despite this simplicity, it achieves comparable performance to state of the art models with significantly fewer parameters. In another effort to reduce parameter size, FITS uses a low-pass filter in the frequency domain to capture relevant patterns without with minimal impact on test performance. By training in the complex frequency domain, the neural network layer efficiently learns amplitude scaling and phase shifting to interpolate time series. This is a very innovative approach compared to anything seen before and is the only one (of the aforementioned) to introduce any filters on the data. FITS reports to outperform DLinear by 8.80% on average.

2 Research Questions

2.1 Deep FITS

FITS uses only a single linear layer without any activation functions and can be considered a form of linear regression on the frequency-time data produced using FFT. This simplicity facilitates its small profile in terms of parameters with fast inference and training time but it might also be a potential area of improvement. To explore whether enhancing this model could improve performance, we experiment with multiple deep variant of FITS. These variants consists of a configurable number of linear layers, ReLU-like activation functions, and dropout layers either in the upscaling part of the network, after the reverse Fast Fourier Transform or parallel to the frequency part of the network.

2.2 Combining FITS with DLinear

In the evolution of forecasting methods, there is a clear tendency for model improvements to stem from processing the data in different ways before making predictions, rather than simply scaling up the models or complexifying the model architecture. DLinear decomposes its input into trend and seasonal components, patterns we notice after some preliminary testing that FITS fails to capture, and models them separately with linear layers before being summed for the final prediction.

FITS takes a different approach by first transforming the time series data into the frequency domain before that modeling with a complex-valued linear layer, and achieves superior results on some datasets.

We hypothesize an integrated approach of the two models will lead to improved performance by leveraging the strengths of both methods.

2.3 Measure of randomness and unpredictability in datasets [→ 3.8]

While reviewing other literature, we were astounded to find that for one of the datasets in [Zen+22], the Repeat column, i.e $x_t = x_{t-1}$, beat all other models in spite their complexity or simplicity on short forecasting horizons. This could imply that if some datasets are sufficiently random, models will incorrectly capture noise as patterns and have worse prediction power.

In [Nie+23], it is argued (with citation to [Ros13]) that the toughest benchmark for forecasting is a random walk without drift, which closely resembles and describes the exchange rate dataset which had the aforementioned quirk. Apparently, the prevailing consensus among statisticians and economists is that if a market is "efficient", meaning all public information is already priced in and new information that affects stock prices are assumed to be unpredictable and random, the asset price becomes impossible to predict, and the best prediction for x_t on average will just be x_{t-1} as published by the acclaimed Eugene Fama in [Fam70]. Intuitively, this also makes sense given that the mean of n random walks as $n \rightarrow \infty$ will be equal to the drift (a straight line) [ref. fig. 26] as it is equally likely to be up or down.

We want to investigate whether we can find a general numerical measure of this unpredictability in other datasets as well. For that, we introduce the Hurst exponent.

3 Methods

3.1 Complex Frequency Domain

In time-series data, the signal is a series of observations in a chronological order, often with equally spaced points representing an amplitude corresponding to some quantity at a fixed sampling rate. This domain focuses on understanding how the signal evolves over time, identifying trends, seasonal patterns, and any anomalies or outliers that may exist. However, while the time-domain analysis provides valuable insights into how the signal changes over time, it often fails to reveal the underlying periodic structures and frequency content of the data.

This is where frequency analysis becomes incredibly useful. Frequency analysis allows us to decompose a time-domain signal into its sinusoidal components, each characterized by a specific frequency, amplitude, and phase. By transforming the signal from the time domain to the frequency domain, we can uncover its inherent periodicities and identify dominant frequencies.

In frequency analysis the frequency component of signals can be represented as a complex number, which captures both the amplitude and phase of the component. The exponential notation for such a number is:

$$z = Ae^{i\theta}$$

Where A is the amplitude and θ is the phase of the number. This form is useful in the context of Fourier transforms where signals are decomposed into their sinusoidal components expressed by the complex exponential.

To calculate the real and imaginary components of the number the rectangular form can be used:

$$z = A \cos(\theta) + iA \sin(\theta)$$

Where $A \cos(\theta)$ represents the real component of the number and $iA \sin(\theta)$ the imaginary component.

3.1.1 Fourier Transform

A fourier transform is a broadly used method for decomposing a function of time, such as time series data, into the frequencies the function consists of. The Fourier transform essentially transforms data in the time domain to the frequency domain. The inverse function, a *synthesis*, for Fourier transformation also exists, and converts the frequencies back into time domain data, which is also referred to as the inverse Fourier transform. Because our data is in the time domain, Fourier transforms are a very powerful tool to disassemble the different attributes (frequencies) in our data, especially if there are any patterns that are prevalent through an extended period of time.

The formula for a continuous Fourier transform is

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

Where $F(\omega)$ is the frequency-domain representation, $f(t)$ is the time-domain signal, and $e^{-i\omega t}$ is the basis function for decomposing a signal into its frequency components. When you integrate the function over a specific amount of time you get a function that represents amplitude and phase of each frequency component in the signal provided. The inverse Fourier transformation from frequency components to time components is given by:

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

A continuous signal is problematic for computers to effectively work with and as such they are sampled with various rates. The length of your sampled signal will allow for greater representations in the frequency domain, and as such a faster sampling rate will give a greater representation at the cost of memory and storage. A critical concept for determining the sampling rate for a signal is the Nyquist-Shannon sampling theorem, which states that to accurately reconstruct a continuous signal from the sampled version the sampling rate must be at least twice the highest frequency of the signal:

$$f_s \geq 2f_{max}$$

Where f_s is the sampling rate and f_{max} is the highest frequency present in the signal.

In many applications related with computer science, it is natural to sample a continuous signal to discretize it. Naturally, as the time series data provided is already in a computer-readable format, we need to perform discrete Fourier transforms (DFT). The formula for a discrete fourier transform is seen in equation 2.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i \frac{2\pi}{N} nk}, \quad k = 0, 1, \dots, N-1 \quad (2)$$

Using the above definition directly to calculate the DFT of a signal would be costly procedure. The reason is clear from the formula - For each data point, for which there are N of, a sum of length N is performed. As such, the time complexity of the processing is proportional to the squared length of the signal we wish to transform i.e., in big O notation it has the complexity of $O(N^2)$.

3.1.2 Fast Fourier Transform

To address this inefficiency, we use the Fast Fourier Transform (FFT), which is an optimized algorithm for computing the Fourier transform. The FFT significantly reduces the computational cost by exploiting the symmetry and periodicity properties of the DFT. The FFT algorithm reduces the time complexity to $O(N \log N)$, making it much more feasible for practical applications.

The FFT often used is the Cooley-Tukey algorithm, which operates by recursively breaking down a DFT of any composite size $N = N_1 N_2$ into many smaller DFTs of sizes N_1 and N_2 , simplifying the computation. This divide-and-conquer approach leverages the efficiency of smaller DFTs to compute the larger one more quickly. The steps of the Cooley-Tukey algorithm are described in further detail in the appendix 8.

Real Valued Fast Fourier Transform

A simple increase in computational efficiency is possible when working with real valued data, $X \in \mathbb{R}$. In such case, only approximately half the computations are required to perform the FFT, as the output will be complex conjugates of one-another:

$$X_k = \overline{X_{N-k}}, \quad \forall k \in \left[1, \left\lfloor \frac{N}{2} \right\rfloor\right]$$

3.2 FITS – Frequency Interpolation (for) Time Series

In the project, the FITS model is written and reimplemented by the team, in order to assess and evaluate it on the original paper's datasets as well as some additional ones. The FITS model as described in the original paper does a series of simple data manipulations and transformations. The pipeline of the FITS model can be seen in the figure 1 below.

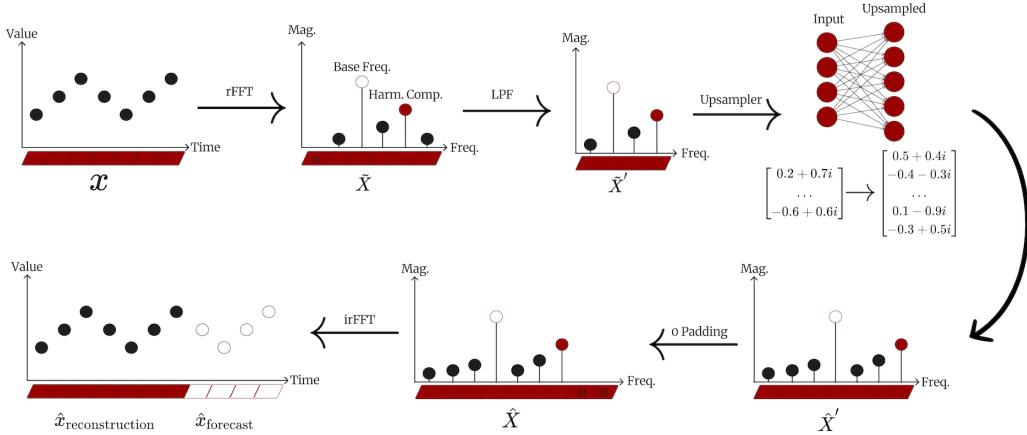


Figure 1: FITS pipeline for reconstruction task

During training and a forward pass through the model, the time series data batch is normalized to zero-mean and unit variance $\hat{x} = \frac{x - \mu}{\sigma}$ followed by a discrete Fourier transform calculated using the real valued Fast Fourier transform variation of equation 2.

One of the methods realized by the authors of the paper, is that for a time series projected into the equivalent frequency domain, the high frequencies in the model usually only represent the noisy terms in the time series, which are not important to the general trends in the series. By using a low-pass filter, the effective size of the model from that point onward is reduced with a marginal error increase. The challenge is choosing a cut-off frequency such that the frequencies that remain still represent and maintain the time series' trends without cutting away important information that could be important for the forecasting or reconstruction tasks.

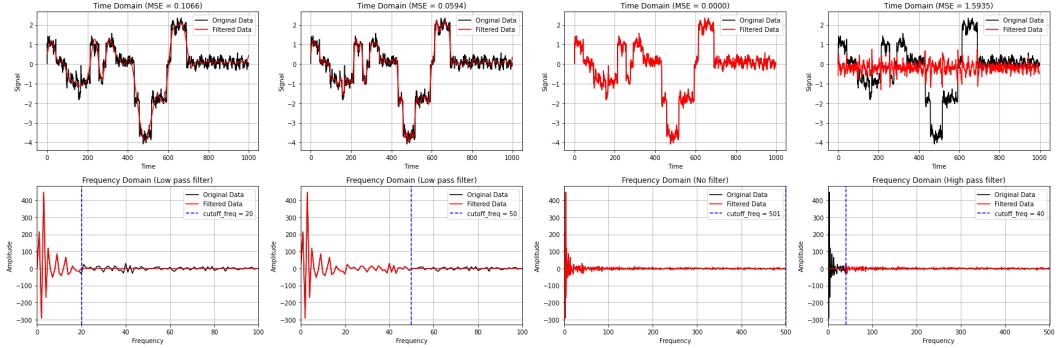


Figure 2: The effect of different cutoff frequencies using a low pass and high pass filter

After the low pass filter on the data in the frequency domain has been applied, the data is run through a complex linear layer with input dimensions corresponding to the cut-off frequency outputting complex frequencies based on a interpolation rate η which is calculated based on the model's output length L_o and corresponding input length L_i .

$$\eta_{freq} = \frac{L_o}{L_i}$$

After which the inverse fast Fourier transform (irFFT) is performed to project the new frequencies back into the time-domain:

$$\hat{x}_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i2\pi \frac{k}{N} n} \quad \text{for } n = 0, 1, \dots, N-1,$$

where \hat{x}_n are the reconstructed samples in the time domain where inverse normalization is then performed to revert the data back to its original scale: $x = \hat{x}\sigma + \mu$

Projecting the signal back into the time domain allows the model to learn using standard loss functions such as the MSE Loss. Depending on the task, a look-back window and a horizon is generated for fore-casting, whereas a down-sampling of the original time series is done, and a reconstruction loss is computed for reconstruction tasks.

When the signal is multichannel the model can be configured in two ways - with the linear layer shared between the channels or with each channel having an independent linear layer. In many time-series datasets the channels often share a base frequency as the data is captured from the same system, when this is the case sharing the linear layer is sufficient and cuts down on total parameters. When this doesn't hold and the signals don't share a common base frequency having individual layers can be a better option at the cost of an increase in parameters.

Another task that this projects aim to evaluate the FITS model on is anomaly detection. Anomaly detection for time series can happen with regards to signal gathering in the form of artifacts or simply data points that are of special interest. In both regards, identifying such points can be very beneficial before working with the data further.

In the reconstruction task, the original data is down-sampled first according to, and then run through a similar pipeline as the forecasting task, but here up-sampled to the same length as the segment of the timeline one wish to compare with. The loss is then a reconstruction loss with a comparison between the up-sampled time series data and the output.

In the project, the FITS model have been implemented with inspiration from the original code. The code and model is annotated with better naming for variables and functions, to help make it easier to work with should modifications be added, as well as a solid testing framework to ensure the model in the project trains exactly the same as the model that is introduced in the paper - when subject to the same hyper parameters and testing data.

Furthermore, the training scripts and data loaders are rewritten, much like how the model is rewritten, to allow for easier readability and extension. The dataset and dataloading classes have been made more general and customizable making it easy to use custom datasets, add time series augmentations, while features used in transformer-based models such as datetime-specific features have been discarded. This allows us to experiment faster with more datasets.

To evaluate the FITS model and observe if we successfully reproduced the results from the paper, we use scripts that mimic the training properties of the original authors, and run it on the same datasets that they describe in the paper, such that the comparison and evaluation between the models are subject to the same constraints and can be compared fairly.[[XZX24](#)]

3.3 Interpreting FITS

At its core FITS interpolates from the frequency components of a given input to that of a longer signal using a single weight matrix. To understand this process it makes sense to look at simple signals to clearly visualize how it works with the frequency domain.

When training FITS on a signal consisting of a combination of a base sine wave and multiple sine waves of higher frequency based on the higher order harmonics of the base signal it's possible to reconstruct the signal up to the cutoff frequency corresponding to the harmonic order of the signal chosen. In figure 3 this is visualized up to the 5th harmonic where everything above is discarded as it is considered "noise" due to the choice of cutoff. As the signal is deconstructed to its frequency components by the FFT, the network is able to reconstruct and predict on signals consisting of arbitrary combinations of these component frequencies up to the cutoff as seen in figure 4. This allows FITS to interpolate between the seen frequencies of signal and makes it robust in making predictions on time series data where the signal has inherent periodicity. Even though this example is very simple it illustrates the major mechanisms of FITS with weight values corresponding the frequencies of the harmonics of the input signal to the harmonics extrapolated in the reconstruction which can be seen for a complex multichannel dataset in the appendix: [11](#)

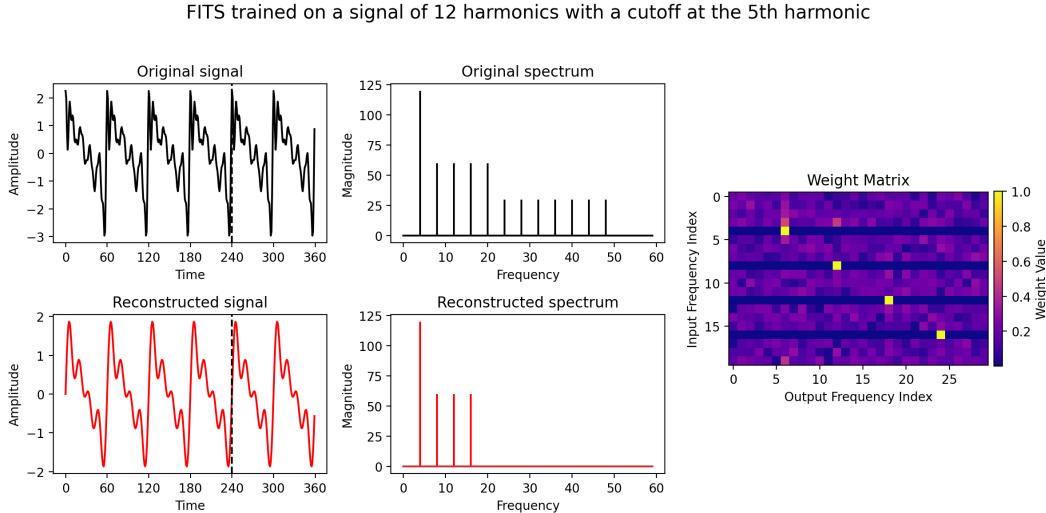


Figure 3: FITS trained with a cutoff at the 5th harmonic of a signal of mixed sines at higher harmonics with a dominant period of 60. The left most plots show the original and reconstructed/extrapolated signal with the vertical line visualising the split between input and prediction. Middle plots are the frequency representations of the signal and the reconstruction. Right plot is the absolute value of the complex weight matrix of FITS for this data showing high activations corresponding to the harmonics.

Prediction of individual harmonics using FITS trained on 12 harmonics with a cutoff at the 5th harmonic

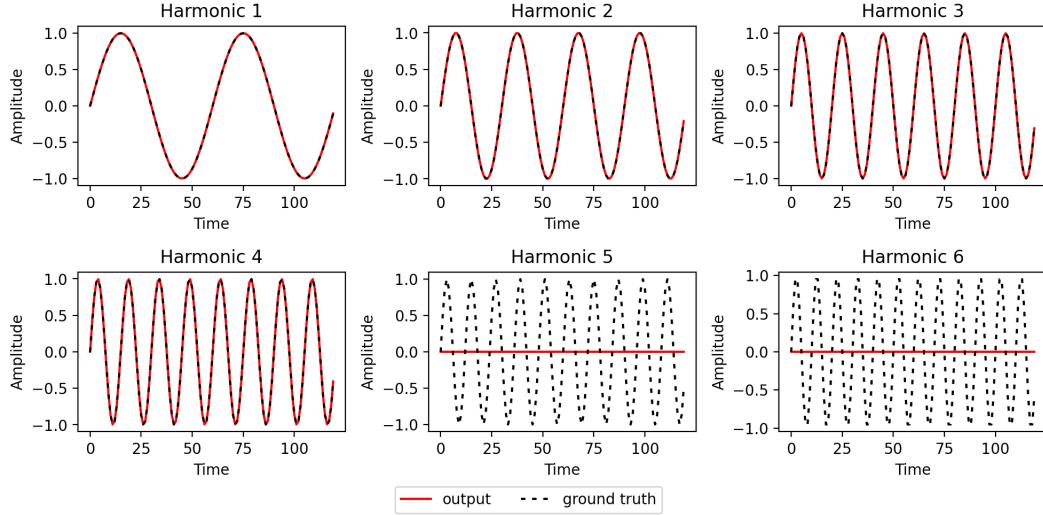


Figure 4: Inference of FITS on the harmonic base components illustrating the inherent separation of the frequency components of the signal.

Where FITS struggles is with signals where a major period is greater than the total sequence length of the output. Here, it's not possible to capture that signal as the dynamics of the period do not fit into the window. This failure mode can be seen most clearly at the beginning and end of the reconstruction, where specific artifacts appear due to the inclusion of higher frequency components than the true signal. This might occur because of spectral leakage, which happens when the signal period does not match the observation window. Leakage causes energy to spread across multiple frequency bins, leading to inaccuracies in the reconstructed signal. Plotting this, it becomes clear that the reconstructed signal is limited by the mismatch between the signal period and the window length, causing the observed artifacts. This artifact becomes more severe the further the period of the signal is from the output length, the effect of which can be seen in figure 5. Here the weights for each of the models are also plotted in which one can see that there is spectral leakage as there are components contributing to the output in the higher frequencies of the output dimension. The models have been initialized with 0 in the weight matrix to clearly visualize the higher frequency components erroneously learned.

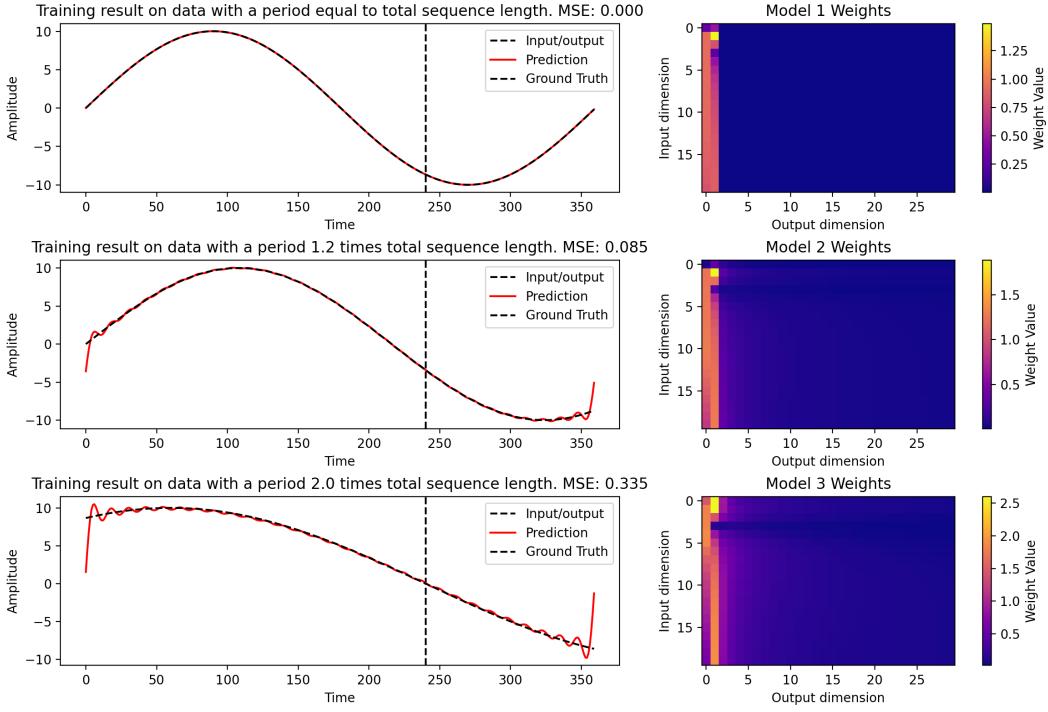


Figure 5: Increasing the length of the period of the signal that is being trained on to longer than the output allows. Output and original signal to the left and weights for each of the models to the right illustrating spectral leakage.

When training on a multichannel signal FITS can either have n number of independent linear layers corresponding to the number of channels in the signal or have one linear layer with shared weights between the n channels. Using the individual configuration provides better reconstruction quality measured in MSE if the channels do not share the same frequency components.

3.4 Deep FITS

To explore adding layers to the complex domain in FITS, modifications to the ReLU activation function had to be introduced since ReLU is incompatible with complex numbers. We implemented the two variants ModReLU and CReLU as described in [Tra+18] to translate ReLU to the complex domain.

CReLU extends ReLU by considering the real and imaginary component for the ReLU operation as separate components:

$$\text{CReLU}(z) = \text{ReLU}(\Re(z)) + i\text{ReLU}(\Im(z))$$

Note that the phase of the number z is modified due to the separate application of the ReLU function on the real and imaginary parts. Specifically, the magnitude of the complex number can change depending on whether the real or imaginary part (or both) are non-negative. If both parts are non-negative, the magnitude increases. If either part is negative, that part is set to zero, potentially reducing the overall magnitude. The original angle (or phase) of the complex number z can be altered significantly. For instance, if z lies in the third or fourth quadrant (where both the real and imaginary parts are negative), applying the ComplexReLU function will result in $\text{CReLU}(z) = 0$, thus collapsing the phase information to zero.

ModReLU works by only considering the magnitude $|z|$ of the complex number z in ReLU together with a learnable parameter b and scaling z by it:

$$\text{modReLU}(z) = \text{ReLU}(|z|+b)e^{i\theta_z} \begin{cases} \left(|z|+b \right) \frac{z}{|z|} & \text{if } |z|+b \geq 0, \\ 0 & \text{otherwise,} \end{cases}$$

The parameter b is introduced as the magnitude is complex number is never negative and ReLU would therefore not have any effect. This introduces a learnable dead-zone below which the complex number will be set to 0.

ModReLU adjusts the activation by shifting the magnitude of the complex number by the parameter b . This shift allows the network to learn a threshold below which the activations are set to zero, effectively creating a dead-zone. When $|z|+b \geq 0$, the magnitude is shifted positively and the complex number retains its phase θ_z . If the condition $|z|+b < 0$ is met, the output is set to zero, meaning both the real and imaginary parts are zeroed out, thereby affecting both the magnitude and phase.

Based on experiments in [Tra+18] CReLU seems to perform better than modReLU in image classification tasks used in a convolutional neural network but as our model is quite different and uses linear layers we intend to test both.

We experimented with adding linear layers with ModReLU and complex dropout in the upsampler and only placing the activation function between the deep layers.

Another variant of the network we tried utilizing linear layers after the reverse Fast Fourier Transform ie after the upscaling. The numbers are now in the real domain and we thus don't need to utilize the complex neural network modifications.

Finally, an experiment was conducted by adding a linear layer that simply maps input to output and sums up the output with the standard FITS output. The idea is that any general drift or linear trend in the data might be better captured by a linear mapping from input to output. As such, a trainable parameter was introduced as well as a linear layer. When reconstructing and forecasting the signal, a sum of both of frequency interpolated signal and the linear layer would happen:

$$xy = (1 - \beta) \cdot \bar{xy} + \beta \cdot xy_{linear}$$

3.5 Hybrid Models

3.5.1 DLinear

The original DLinear model first decomposes the input time series into a trend component and a remainder (seasonal) component using a moving average filter with a kernel size of 25. The trend component captures the long-term progression of the time series, while the remainder represents the seasonal or periodic fluctuations. It then uses two separate linear layers to learn the patterns of each component and sums them to produce the final prediction.

3.5.2 DLinear + FITS

In DLinear + FITS, we extend DLinear by incorporating FITS on the residual signal. After applying the standard DLinear decomposition and training the respective linear layers to produce predictions for the trend and seasonal parts, we compute the residual. The residual in this context is the difference between the original input time series and the sum of the predictions from DLinear during training. It represents the part of the signal that DLinear's trend and seasonal components couldn't explain or predict. This residual signal is then passed through and trained in a standard implementation of FITS to hopefully capture any remaining patterns or fluctuations that were not captured by the previous components. Finally, the predicted trend, seasonal components, and the forecast portion of FITS output are summed to produce the final prediction.

3.5.3 FITS + DLinear

In the FITS + DLinear model, we reverse the order of operations compared to DLinear + FITS. First, the input time series is processed through FITS, which as we know, normally returns its predictions for the entire sequence and forecast horizon. Its output then serves as training data for DLinear which produces the final prediction. This approach differs from DLinear + FITS in that FITS is applied to the entire input signal rather than just the residual of DLinear's predictions. By applying FITS first, we allow the model to capture the complex frequency patterns, ideally the main cyclical components in the data, before going through the seasonal and trend decomposition steps (and subsequent training). This may help in situations where the input signal contains periodic patterns that are not as easily separated by the moving average filter. The subsequent DLinear model can then focus on refining its predictions based on the "denoised signal" provided by FITS rather than the raw "noisy" input data. Although in hindsight, we could maybe do away with the moving average filter completely, as FITS

essentially serves as an enhanced version of that (see figure 6 below) or not train FITS, as the low pass filter might be enough by itself.

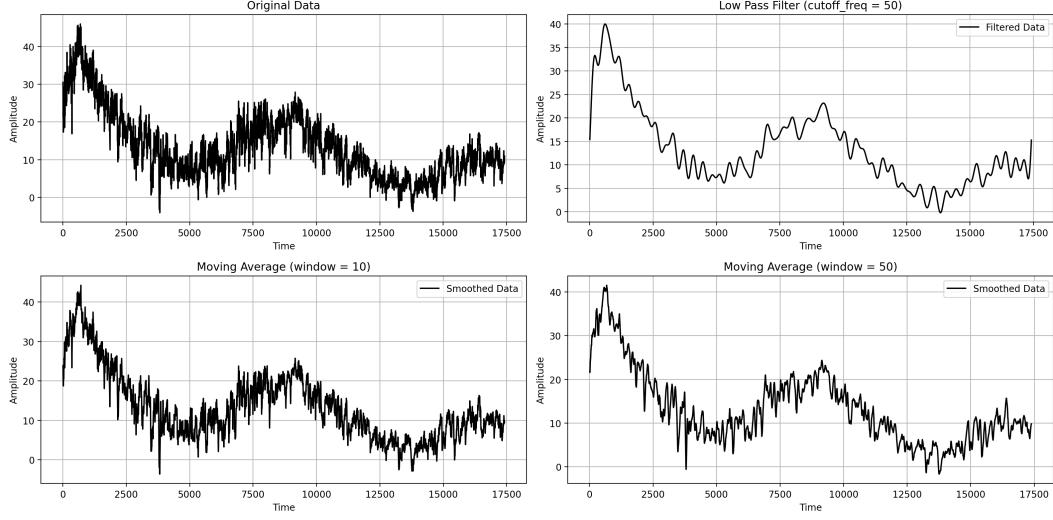


Figure 6: The effects of a moving average filter vs a low pass filter

The code for the three models can be found here:

https://github.com/thorhojhus/ssl_fts/tree/DLinear/models

3.6 Hurst exponent

The Hurst exponent H seeks to provide insight into the behavior or predictability of a time series by quantifying the tendency of a dataset to exhibit trending, mean-reverting, or random walk behaviour. H always takes a value between 0 and 1 and is formally defined as^{1,2}:

$$\mathbb{E} \left[\frac{R(n)}{S(n)} \right] = Cn^H \text{ as } n \rightarrow \infty$$

Where

$$R(n) = \max(Z_1, Z_2, \dots, Z_n) - \min(Z_1, Z_2, \dots, Z_n) \quad (\text{Calculate the range } R)$$

$$Z_t = \sum_{t=1}^t Y_t \quad \text{for } t = 1, 2, \dots, n \quad (\text{Cumulative deviation from the mean-adjusted series})$$

$$Y_t = X_t - \mu \quad \text{for } t = 1, 2, \dots, n \quad (\text{Remove trend, keep fluctuations})$$

$$\mu = \frac{1}{n} \sum_{t=1}^n X_t \quad (\text{Mean of the series } X)$$

And

$$S(n) = \sqrt{\frac{1}{n} \sum_{i=1}^n (X_i - \mu)^2} \quad (\text{Standard deviation } S \text{ of the original series } X)$$

H is then fitted as a straight line using least-squares regression on $\log[R(n)/S(n)]$ as function of $\log n$ where the slope gives H (ref. figure 7).

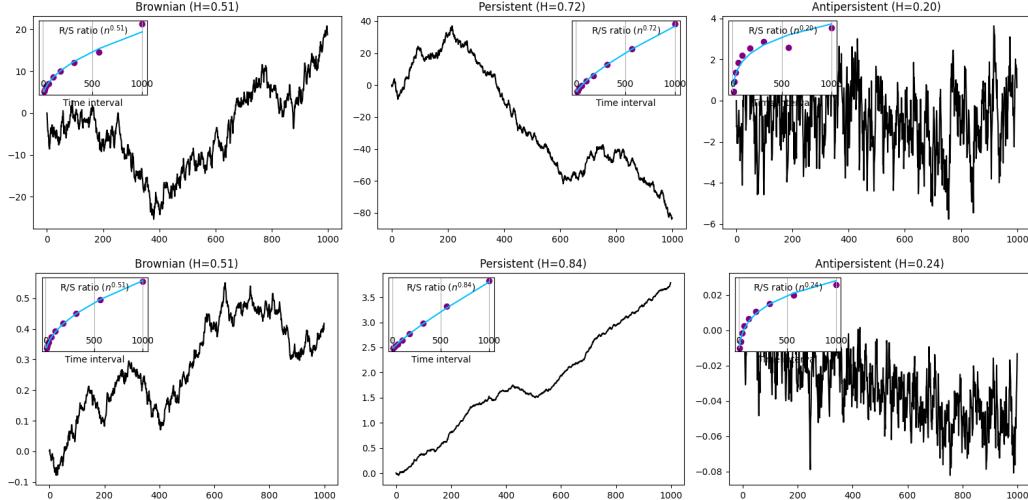


Figure 7: 1 simulation (top row) & Mean of 10000 simulations (bottom row) and their mean R/S

The rescaled range $R(n)/S(n)$ is really just a measure of the variability of a time series relative to its standard deviation. This means that it captures the idea of how much the cumulative deviations from the mean ($R(n)$) vary compared to the typical size of fluctuations in the original series ($S(n)$).

If the time series is purely random (e.g., brown noise or random walk), the cumulative deviations from the mean ($R(n)$) will grow roughly proportionally to the square root of the series length ($n^{H=0.5}$). This is because random fluctuations tend to cancel each other out over time. This indicates short-memory, meaning each value is independent of each other and there is no correlation.

However, if the time series has long-term memory or persistence (no noise), meaning trending behaviour, i.e. high values to be followed by higher value and lower values tend to be followed by lower values, the cumulative deviations from the mean ($R(n)$) will grow faster than the square root of the series length ($n^{H>0.5}$). This is because the positive (or negative) deviations tend to maintain their direction (positive or negative deviations) for longer periods than would be expected by chance, leading to larger cumulative deviations.

Conversely, if the rescaled range $R(n)/S(n)$ grows slower than $H = 0.5$, so $n^{(H<0.5)}$, it suggests that the time series is anti-persistent, meaning high values are more likely to be followed by low values, and low values are more likely to be followed by high values, with a tendency to revert to the mean value over time (white noise / quickly fluctuating series).

All Hurst Exponents used in plots have been calculated using this python library:

<https://pypi.org/project/hurst/>.

3.7 Autocorrelation function (ACF)

The autocorrelation function (ACF) is a statistical tool used to measure the correlation between a time series and a lagged version of itself. It quantifies the degree of similarity between observations as a function of the time lag between them. The ACF is also useful for identifying patterns, such as persistence or mean reversion, in a time series.

The ACF is defined as³:

$$\rho(k) = \frac{\text{Cov}(X_t, X_{t+k})}{\text{Var}(X_t)}$$

Where

$$\begin{aligned} \text{Cov}(X_t, X_{t+k}) &= \frac{1}{n-k} \sum_{t=1}^{n-k} (X_t - \mu)(X_{t+k} - \mu) && \text{(Autocovariance at lag } k\text{)} \\ \text{Var}(X_t) &= \frac{1}{n} \sum_{t=1}^n (X_t - \mu)^2 && \text{(Variance)} \\ \mu &= \frac{1}{n} \sum_{t=1}^n X_t && \text{(Mean)} \end{aligned}$$

The ACF is related to the Hurst exponent in that they both provide information about the memory or persistence of a time series. A time series with long-term memory or persistence a (H close to 1) will typically exhibit slowly decaying autocorrelations and remains positive for many lags, because it mostly correlates with itself. Conversely, a time series anti-persistent behaviour (H close to 0) will show rapidly decaying and quickly oscillating autocorrelations because of its erratic nature. If the time series is completely uncorrelated exhibiting no memory ($H = 0$), meaning all steps are i.i.d., stationary, and uncorrelated, the ACF should be close to zero for all lags $k > 0$. Note that this is different from a random walk as every next time step becomes its new stationarity, therefore exhibiting some autocorrelation that decays slow due to the dependence of each step on the previous step.

In figure 8 below, the ACF plots for 10000 simulated random walks show similarly decaying autocorrelations to those descriptions.

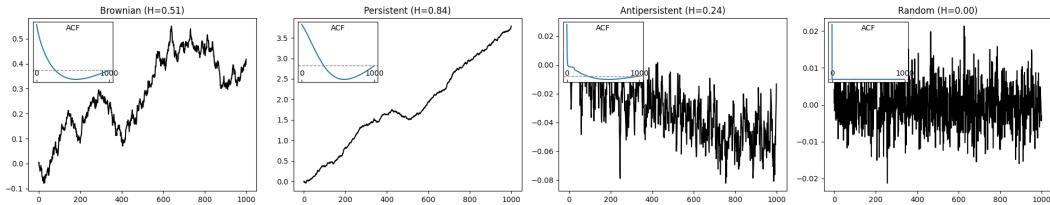


Figure 8: Mean of 10000 simulated random walks and their ACF

However, an ACF plot for a single example is much more likely to resemble something like figure 9 below as taking the mean smooths the otherwise common distinguishable characteristics of the three variants.

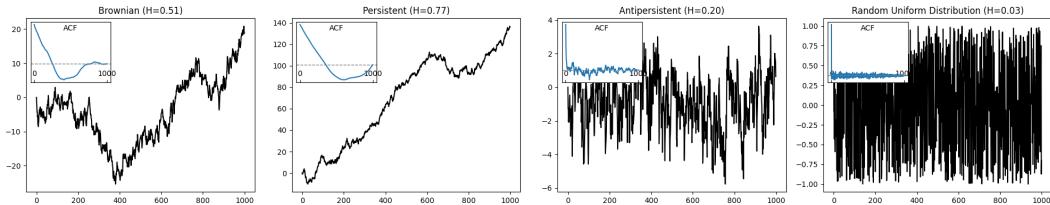


Figure 9: 1 simulated random walk and its ACF

3.8 Baseline Models in forecasting

As forecasting can be a difficult task, some of the simplest models can perform quite well. As we recall from section 2.3 , if we assume a random walk without drift, meaning each data point is independent and random, no model will predict better than the simple yet surprisingly effective forecasting model **Naive Forecasting Model**. The idea is simple; predict the next L time steps to be equal to the last observed value:

$$\hat{z}_{T+l|T} = z_T, \quad \forall l \in [1 \cdots L]$$

This provides a solid baseline for which to benchmark more sophisticated models against. If this baseline model reliably predicts better than the rest, it strongly hints that the dataset is fundamentally unpredictable.

Another straightforward and simple baseline for time series data is simply to forecast the next L values as the mean of the observed time series, a **Mean Valued Forecast**:

$$\hat{z}_{T+l|T} = \mu_z, \quad \forall l \in [1 \dots L]$$

A simple **linear regression** on the time series data could also serve as a baseline but for reasons will be expanded on later, has shown to be an inferior baseline and will not be utilized.

These are the the most simple forecasting models, and we include the Naïve forecasting as a baseline comparison in the results section on page 19.

3.9 Autoregressive Integrated Moving Average Models (ARIMA)

Moved to appendix. 8

4 Data

4.1 Real data

Make descriptions of how the data looks plotted and asses any potential for forecasting on the data such as any patterns that are present throughout the data (or parts of it)

We use 9 common datasets used to benchmark time-series based models [Wu+22a]. This is used as a baseline to easily reproduce previous papers' results and compare new modifications. Their descriptions are outlined in the tables below. Summary statistics have been omitted from the tables below because of the vast number of independent variables.

`electricity.csv` contains the hourly electricity consumption of 321 different clients in kWh from 2016-07-01 to 2019-07-02.¹

`ETTh1.csv`, `ETTh2.csv`, `ETTm1.csv`, and `ETTm2.csv` are all datasets of Electricity Transformer Temperature where each column measures different loads and their usefulness and Oil Temperature (OT) from 2016-07-01 to 2018-06-26.²

`traffic.csv` contains hourly recordings of road occupancy rates from 862 different sensors on San Francisco freeways between 2015-01-01 and 2015-04-27 from California Department of Transportation.³

`national_illness.csv` contains weekly recorded influenza-like illness (ILI) from CDC in United States from 2002-01-01 and 2022-06-30.⁴

`weather.csv` is a collection of German weather data with 21 different meteorological indicators.⁵

`exchange_rate.csv` is a collection of daily exchange rates of eight foreign countries including Australia, British, Canada, Switzerland, China, Japan, New Zealand and Singapore ranging from 1990 to 2010.⁶

\$GD and \$MRO stock data collected from the python library `yfinance`⁷. Comprises of stock data changes over 15723 days, or approximately 43 years, of stock data. The columns consist of "Open", "High", "Low", "Close", "Volume", "Adj Close".

Furthermore, some of the data has been cleaned of invalid or extreme outlier values, like -9999 for certain variables in `weather.csv`. We have replaced those values with interpolation between the previous and next value of the corresponding variable. Other than that, no processing has been made to any of the other datasets.

Most common benchmarking datasets

Dataset	Electricity (ECL)	ETTh1	ETTh2	ETTm1	ETTm2
Prediction (y)	Last col.	OT	OT	OT	Channels
Description of y	kWh	Oil Temp.	Oil Temp.	Oil Temp.	Oil Temp.
Sampling Rate	1 hour	1 hour	1 hour	15 min	15 min
Channels (columns) ×	321	7	7	7	7
Total Timesteps (rows)	26,304	17,420	17,420	69,680	69,680
Hurst Exp. (y)	0.2449	0.3191	0.2322	0.3471	0.2319

Note: Oil Temp.: Oil temperature. Last Col.: Last column in the dataset (index -1)

Expanded table 7 from [XZX24]

In each dataset, the last column has been renamed Original Target (OT) by other papers with no mention of it what it was before. Presumably, this was done in order to fit with the data loader. Therefore, some of the column descriptions are unknown.

In the appendix plots of all the channels in the datasets are provided and their autocorrelation over time. From those we see clearly daily periods in all the datasets but illness, where there instead is a yearly period.

We have decided to include two more datasets for benchmarking. The first one, \$GD, was collected using the `yfinance` python library to download all stocks in the S&P500⁶ for all available intervals (1m, 2m, 5m, 15m, 30m, 1h, 1d, 5d, 1wk, 1mo, 3mo) and then sorted in descending order by its

number timesteps and afterwards its deviation from $H = 0.5$. The second one, \$MRO, was collected the same way except the data was sent through a low pass filter with cutoff frequency 200, mimicking the FITS pipeline and removing "noise", before being sorted. The rationale for this method is that theoretically, these two stocks should resemble random walks the least out of the 500 and therefore have the least "unpredictability".

Dataset	Traffic	Illness	Weather	Exchange rate	\$GD	\$MRO
Prediction (y)	Last col.	OT	OT	OT	Adj Close	Adj Close
Description of y	Road occ.			FX rate	Price	Price
Sampling Rate	1 hour	1 week	10 min	1 day	1 day	1 day
Channels (columns) \times	861	6	20	7	6	6
Total Timesteps (rows)	17544	966	52696	7588	15723	15723
Hurst Exp. (y)	0.1637	0.3410	0.3249	0.5002	0.5792	0.4982
Hurst Exp. ($y_{\text{low_pass}}$)					0.3828	0.3086

Note: Road occ.: Road occupancy rates. Descriptions left blank are unknown.

A Hurst Exponent of 0.5 indicates random walk behaviour

It is possible to train on these datasets in three different ways:

1. Univariate input and univariate target, i.e. predicting y based on only the previous rows of the same column (univariate regression)
2. Multivariate input and univariate target, i.e. predicting y based on all other variables (multivariable regression)
3. Multivariate input and multivariate target, i.e. predicting all columns besides the `datetime` column based on every other column. (multiple auto regression)

For benchmarking FITS against the previous other state-of-the-art models, we rely on the results from a previous 2023 paper [SWW23] as retraining all the models would be infeasible and train only on multivariate input with multivariate prediction.

4.2 Synthetic data

In order to help identify some of the models weaknesses and strengths, we developed a framework for generating synthetic data to allow us to manipulate trends, noise, and other characteristics. The purpose of this is to help us see how the nature of the data influences FITS' predictive power.⁷

We have not used any synthetic datasets for official benchmarking but it has been used for interpreting and experimenting with FITS.

4.3 Training Setup

Our training process begins by calculating the training loss on the combined input and output of the signal, providing the model with the most comprehensive information. Afterwards, we enter a fine-tuning stage, where the loss is computed between the prediction part of the network and the unseen part of the signal. In summary, we first consider both x and y , and then focus solely on y .

We use a training, validation, and test split of the datasets to train the models effectively. To prevent overfitting on the training set, we use the validation set to adjust the learning rate and implement early stopping if no improvement is observed within a specified number of epochs. At the end of each stage, we calculate the test loss in terms of Mean Squared Error (MSE) exclusively on the prediction part of the output.

The dataset is split into training, validation, and test sets in a 70:10:20 ratio respectively. For ETT we split 60:20:20 as that is what others report on.

5 Results

Below we report the results of our various models and baselines on all the different dataset trained and test with multivariate input and multivariate output similar to the reported

5.1 Common benchmark datasets

To reproduce the results reported in [XZX24] we run with exactly the same model parameters and splits on the datasets as reported and read from their source code. All the runs are done multivariate and we report the MSE over all the channels in the dataset

The parameters for the model are as follows:

For the ETTh1 dataset, the lookback window (seq_len) is set to 360, the base period (base_T) is 24 hours, and the harmonic order (H_order) is 6. In contrast, the ETTh2 dataset uses a lookback window of 720, with the remaining parameters unchanged. Both ETTm1 and ETTm2 share the same parameters: a lookback window of 720, a base period of 96 (representing 15-minute intervals per day), and a harmonic order of 14. The Electricity dataset features a lookback window of 720, a base period of 24 hours, and a harmonic order of 10. Similarly, the weather dataset has a lookback window of 720, a base period of 144 (equivalent to 10-minute intervals per day), and a harmonic order of 12 but notably is the only dataset that is trained on in individual mode - having independent layers for all channels. Traffic has a base period of 24 hours and harmonic order of 10.

The base period (base_T) for each dataset reflects the number of particular time intervals within a day at which the dataset was sampled, while the harmonic order denotes the n-th harmonic derived from base_T, utilized to determine the cutoff frequency. Note that for national illness (ILI) the base period is number of weeks in a year.

The primary results are provided in table 1, where we also include exchange rate and national illness (ILI). We include comparable models and repeat of the last value into the forecast horizon as a baseline. ARIMA (Auto-ARIMA) was only tested on 100 samples and on forecast horizon of 96 due to the computational complexity and time required.

Our reproduced scores have some minor discrepancies compared to the reported, which might be caused by the slightly different data handling or different seeds. We also notice from the results provided in the log files in the original FITS repo, that the numbers are not rounded but floored to the significant digit (e.g., 0.34857 is floored to 0.348), while ours have been rounded often causing slight differences in the reported scores. A particularly notable difference is in the ETTh2 dataset at the 192 forecast horizon, where there is a discrepancy of 0.01, for which we lack a clear explanation. None of these differences are however large enough to change the rankings compared to other models.

Methods		IMP.	FITS (ours)		FITS*	DLinear*		NLinear*		PatchTST/64		PatchTST/42		Repeat*		ARIMA
Metric		MSE	MSE	MAE	MSE	MSE	MAE	MSE								
Weather	96	4.20%	0.143	0.195	0.143	0.176	0.237	0.182	0.232	0.149	0.198	0.152	0.199	0.259	0.254	0.239
	192	4.30%	0.186	0.236	0.186	0.220	0.282	0.225	0.269	<u>0.194</u>	0.241	0.197	0.243	0.309	0.292	
	336	3.81%	0.236	0.277	0.236	0.265	0.319	0.271	0.301	<u>0.245</u>	<u>0.282</u>	0.249	0.283	0.377	0.338	
	720	1.95%	0.308	0.330	0.307	0.323	0.362	0.338	0.348	<u>0.314</u>	0.334	0.320	0.335	0.465	0.394	
Traffic	96	-6.74%	0.386	0.266	0.385	0.410	0.282	0.410	0.279	0.360	0.249	<u>0.367</u>	<u>0.251</u>	2.723	1.079	1.448
	192	-4.77%	0.398	0.273	0.397	0.423	0.287	0.423	0.284	<u>0.379</u>	0.256	<u>0.385</u>	<u>0.259</u>	2.756	1.087	
	336	-5.08%	0.413	0.277	0.410	0.436	0.296	0.435	0.290	<u>0.392</u>	0.264	<u>0.398</u>	<u>0.265</u>	2.791	1.095	
	720	-3.79%	0.449	0.297	0.448	0.466	0.315	0.464	0.307	<u>0.432</u>	0.286	<u>0.434</u>	0.287	2.811	1.097	
Electricity	96	-3.73%	0.134	0.229	0.134	0.140	0.237	0.141	0.237	<u>0.129</u>	0.222	<u>0.130</u>	0.222	1.588	0.946	
	192	-3.29%	0.152	0.243	0.149	0.153	0.249	0.154	0.248	<u>0.147</u>	0.240	<u>0.148</u>	0.240	1.595	0.950	
	336	-1.21%	<u>0.165</u>	0.256	<u>0.165</u>	0.169	0.267	0.171	0.265	<u>0.163</u>	<u>0.259</u>	0.167	0.261	1.617	0.961	
	720	-3.43%	0.204	0.292	0.203	0.203	0.301	0.210	0.297	<u>0.197</u>	0.290	<u>0.202</u>	0.291	1.647	0.975	
ILJ	24	-47.18%	2.497	0.988	-	2.215	1.081	1.683	0.858	1.319	0.754	<u>1.522</u>	<u>0.814</u>	6.587	1.701	9.249
	36	-40.59%	2.407	0.995	-	1.963	0.963	1.703	0.859	<u>1.579</u>	0.870	1.430	0.834	7.130	1.884	
	48	-37.99%	2.472	1.038	-	2.130	1.024	1.719	0.884	1.553	0.815	1.673	0.854	6.575	1.798	
	60	-39.28%	2.421	1.033	-	2.368	1.096	1.819	0.917	1.470	0.788	<u>1.529</u>	<u>0.862</u>	5.893	1.677	
ETTh1	96	5.63%	0.373	0.395	<u>0.372</u>	0.375	0.399	0.374	0.394	0.370	0.400	0.375	0.399	1.295	0.713	0.839
	192	1.97%	0.407	0.415	0.404	<u>0.405</u>	0.416	0.408	0.415	0.413	0.429	0.414	0.421	1.325	0.733	
	336	-0.23%	0.429	0.429	0.427	0.439	0.443	0.429	0.427	0.422	0.440	0.431	0.436	1.323	0.744	
	720	5.84%	0.428	0.448	0.424	0.472	0.490	0.440	<u>0.453</u>	0.447	0.468	0.449	0.466	1.339	0.756	
ETTh2	96	1.48%	0.270	0.322	0.271	0.289	0.353	0.277	0.338	0.274	0.337	0.274	0.336	0.432	0.422	0.444
	192	-21.35%	0.341	0.365	0.331	0.383	0.418	0.344	0.381	0.341	0.382	<u>0.339</u>	<u>0.379</u>	0.534	0.473	
	336	-8.36%	0.359	0.439	0.354	0.448	0.465	0.357	0.400	0.329	<u>0.384</u>	<u>0.331</u>	0.380	0.591	0.508	
	720	-0.26%	0.380	0.457	0.377	0.605	0.551	0.394	<u>0.436</u>	<u>0.379</u>	0.422	<u>0.379</u>	0.422	0.588	0.517	
ETTm1	96	-5.84%	0.308	0.345	0.303	0.299	0.343	0.306	0.348	<u>0.293</u>	0.346	0.290	0.342	1.214	0.665	0.910
	192	-1.48%	0.337	0.363	0.337	0.335	<u>0.365</u>	0.349	0.375	<u>0.333</u>	0.370	0.332	0.369	1.261	0.690	
	336	0.00%	0.366	0.381	0.366	<u>0.369</u>	0.386	0.375	0.388	<u>0.369</u>	0.392	0.366	0.392	1.283	0.707	
	720	1.20%	0.415	0.408	0.415	0.425	0.421	0.433	0.422	<u>0.416</u>	0.420	0.420	0.424	1.319	0.729	
ETTm2	96	1.85%	0.162	0.242	0.162	0.167	0.262	0.167	0.255	0.166	0.260	0.165	0.255	0.266	0.328	0.224
	192	1.85%	0.216	0.276	0.216	0.224	0.303	0.221	0.293	0.223	0.303	<u>0.220</u>	<u>0.292</u>	0.340	0.371	
	336	2.24%	0.268	0.309	0.268	0.281	0.342	<u>0.274</u>	0.327	<u>0.274</u>	0.342	0.278	0.329	0.412	0.410	
	720	3.72%	0.349	0.360	0.348	0.397	0.421	0.368	0.384	0.362	0.421	0.367	0.385	0.521	0.465	
Exchange	96	-14.94%	0.087	0.204	-	0.081	<u>0.203</u>	0.089	0.208	-	-	-	-	<u>0.081</u>	0.196	0.074
	192	-12.78%	0.180	0.295	-	0.157	<u>0.293</u>	0.180	0.300	-	-	-	-	<u>0.167</u>	0.289	
	336	-8.41%	0.333	0.405	-	0.305	0.414	<u>0.331</u>	0.415	-	-	-	-	0.305	0.396	
	720	-32.39%	0.951	0.713	-	0.643	0.601	1.033	0.780	-	-	-	-	0.823	0.681	

*Results borrowed from [ZXX24], [Zen+22], and [Nie+23].

Table 1: Multivariate long-term forecasting errors in terms of MSE and MAE, the lower the better. Note that the ILI dataset has prediction length $T \in \{24, 36, 48, 60\}$ whereas the others have prediction length $T \in \{96, 192, 336, 720\}$. Repeat repeats the last value in the look-back window and serves as our baseline. The best results are highlighted in bold and the second best results are highlighted with underline. IMP denotes the relative improvement or downgrade from FITS (ours) and the second best or best model.

5.2 Results with variants of FITS

We present our findings from our research on enhancing the FITS model. In Table 2, we examined the effects of different lookback window lengths [96, 192, 336, 720] on various models using the ETTm1 dataset. The real-valued deep FITS consistently underperformed, while the other models showed similar results. The bypass layer model performed slightly better with shorter lookback windows [96, 192], whereas deep FITS showed marginal improvements with longer lookback windows [336, 720].

	ETTm1							
Lookback	96		192		336		720	
Model	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
real_deep_FITS	0.5771	0.5104	0.5311	0.4899	0.4727	0.4602	0.4261	0.4287
FITS_bypass_layer	0.4861	0.4485	0.4405	0.4229	0.4275	0.4165	0.4155	0.4113
deep_FITS (ModReLU)	0.4871	0.4514	0.4416	0.4263	0.4272	0.4199	0.4138	0.4144
deep_FITS_after_upscaler	0.4866	0.4488	0.4407	0.4227	0.4274	0.4161	0.4138	0.4113

Table 2: Long-term forecasting results in MSE and MAE with different lookback lengths on ETTm1 with forecast horizon of 720 steps.

Table 3 presents the models' performance across all ETT datasets. The models were configured with harmonic order 6 and, where applicable, 2 hidden layers with 128 neurons per layer. Despite the significant increase in parameters compared to the original FITS, none of the variants demonstrated substantial improvements.

Dataset	ETTh1				ETTh2				ETTm1				ETTm2			
	96	192	336	720	96	192	336	720	96	192	336	720	96	192	336	720
FITS	0.3804	0.4173	0.4426	0.4363	0.2707	0.3418	0.3588	0.3796	0.3076	0.3372	0.3655	0.4148	0.1617	0.2157	0.2687	0.3494
deep_FITS (ModReLU)	0.3692	0.4050	0.4333	0.4440	0.2687	0.3414	0.3593	0.3830	0.3100	0.3371	0.3643	0.4137	0.1722	0.2155	0.2682	0.3501
real_deep_FITS	0.3792	0.4158	0.4494	0.5164	0.2877	0.3619	0.3783	0.4251	0.3117	0.3437	0.3672	0.4248	0.1719	0.2299	0.2825	0.3586
FITS_bypass_layer	0.3761	0.4153	0.4462	0.4635	0.2685	0.3398	0.3573	0.3826	0.3062	0.3359	0.3655	0.4155	0.1617	0.2155	0.2674	0.3495
deep_FITS_after_upscaler	0.3804	0.4178	0.4427	0.4365	0.2703	0.3415	0.3586	0.3797	0.3079	0.3373	0.3657	0.4149	0.1618	0.2161	0.2680	0.3494

Table 3: Long-term forecasting results on the ETT datasets in MSE with a look-back window of 720 and different forecast horizons. real_deep_FITS, deep_FITS and deep_FITS_after_upscaler use 3 layers and 128 neurons per layer.

To evaluate whether deep FITS with ModReLU actually performed statistically better than FITS on ETTh1 at a forecast horizon of 96, as indicated by the results in Table 3, we trained each model 10 times with different random seeds to calculate the mean and standard deviation of the MSE. We then conducted a paired t-test to assess the statistical significance of the difference between the two models' performance.

Model	Mean MSE	Std Dev	p-value
FITS	0.3806	0.0004	3.42×10^{-13}
deep_FITS (ModReLU)	0.3692	0.0002	

Table 4: Comparison of FITS and deep_FITS models on ETTh1 dataset (lookback: 720, horizon: 96).

The results in table 4 show that deep FITS with ModReLU achieved a lower mean MSE (0.3692) compared to FITS (0.3806), with a smaller standard deviation as well. A paired t-test yielded a p-value of 3.42×10^{-13} , indicating a statistically significant difference between the two models' performance.

We also present an analysis of the performance of three Deep FITS variants: CReLU, ModReLU, and Deep FITS After Upscaler, on the ETTh1 dataset in table 5. Our investigation explores the effect of varying the number of layers and hidden neurons, utilizing a consistent lookback and forecast horizon of 720 time steps. The baseline '0 layers' configuration represents the original FITS model, with its linear layer embedded within a sequential module for consistency.

Our findings show that the Deep FITS After Upscaler variant achieves marginally superior performance, resulting in the lowest MSE when configured with 3 layers and 512 hidden units per layer, and with the lowest deterioration of MSE overall. CReLU consistently performs the worst among

Layers	CReLU Deep FITS				ModReLU Deep FITS				Deep FITS After Upscaler			
Hidden Units	64	128	256	512	64	128	256	512	64	128	256	512
0	0.4363	0.4363	0.4363	0.4363	0.4363	0.4363	0.4363	0.4363	0.4363	0.4363	0.4363	0.4363
1	0.4929	0.4870	0.5198	0.5315	0.4379	0.4498	0.4413	0.4494	0.4367	0.4366	0.4364	0.4364
2	0.4759	0.4816	0.5010	0.5214	0.4526	0.4440	0.4478	0.4492	0.4362	0.4363	0.4366	0.4364
3	0.4797	0.4522	0.4626	0.4857	0.4607	0.4515	0.4506	0.4470	0.4365	0.4360	0.4364	0.4357
4	0.4972	0.4561	0.4565	0.4822	0.4736	0.4588	0.4530	0.4477	0.4366	0.4361	0.4365	0.4361

Table 5: MSE results for CReLU Deep FITS, ModReLU Deep FITS, and Deep FITS after upscaler on ETTh1 (forecast horizon: 720, lookback window: 720, harmonic order: 6).

the three variants across different configurations. For instance, with 1 layer and 512 hidden units, CReLU records an MSE of 0.5315, significantly higher than the other variants. ModReLU shows slightly better performance than CReLU but still lags behind the Deep FITS After Upscaler, and the unmodified FITS.

5.3 Price datasets

Price datasets with various error metrics across 381, 369, 351, and 303 test samples for horizons 96, 192, 336, and 720 respectively with a set lookback-window of 336. The standard FITS models have their cutoff frequency noted in parentheses (10 & 100), whereas the hybrid FITS models have a cutoff frequency of 49 for DLinear + FITS and 168 for FITS + DLinear.

Mean Squared Error (MSE):

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

Mean Absolute Error (MAE):

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

Squared Error (SE):

$$\text{SE} = (y_T - \hat{y}_T)^2$$

Relative Root Mean Squared Error (RRMSE):

$$\text{RRMSE} = \frac{\sqrt{\text{MSE}}}{\frac{1}{n} \sum_{i=1}^n |y_i|}$$

Where n is the length of the forecasted horizon (96, 192, 336, and 720), y_i represents the true value, \hat{y}_i represents the predicted value, and \hat{y}_T & y_T represents the last value in lookback window.

The training and test logs for these results can be found here:

https://github.com/thorhojhus/ssl_fts/tree/DLinear/results

Dataset		Exchange Rate				General Dynamics <i>GD</i>				Marathon Oil Corp <i>MRO</i>				Best
Model	Horizon	MSE	MAE	SE	RRMSE	MSE	MAE	SE	RRMSE	MSE	MAE	SE	RRMSE	48
Repeat	96	0.081	0.196	0.164	19.53%	1.196	0.763	2.283	10.98%	4.465	1.246	6.730	36.19%	17/48
	192	0.167	0.289	0.342	28.41%	2.250	1.077	4.923	15.07%	6.562	1.653	10.663	44.26%	
	336	0.306	0.397	0.626	38.85%	3.899	1.444	8.558	19.78%	10.411	2.190	18.627	56.68%	
	720	0.813	0.677	1.781	63.52%	9.859	2.366	21.451	31.22%	20.380	3.233	33.096	85.56%	
DLinear	96	0.081	0.203	0.184	19.51%	1.659	0.948	2.356	12.93%	3.826	1.270	5.770	33.50%	9/48
	192	0.157	0.293	0.321	27.51%	2.430	1.153	4.825	15.66%	6.203	1.765	10.376	43.03%	
	336	0.299	0.417	0.492	38.44%	4.189	1.551	8.089	20.50%	9.121	2.178	15.831	53.05%	
	720	1.040	0.767	1.804	71.86%	8.923	2.341	15.157	29.70%	16.783	2.968	19.187	77.64%	
DLinear + FITS	96	0.089	0.214	0.152	20.48%	1.660	0.933	3.268	12.93%	3.838	1.276	5.890	33.55%	6/48
	192	0.173	0.308	0.295	28.93%	2.382	1.128	4.686	15.51%	5.824	1.696	9.691	41.70%	
	336	0.300	0.416	0.702	38.51%	5.116	1.707	8.590	22.66%	9.322	2.203	17.687	53.63%	
	720	0.962	0.736	1.485	69.12%	8.889	2.323	15.236	29.65%	16.773	3.021	17.534	77.62%	
FITS + DLinear	96	0.079	0.200	0.156	19.31%	1.775	0.946	3.212	13.37%	3.823	1.272	5.774	33.48%	16/48
	192	0.157	0.294	0.302	27.52%	2.374	1.136	4.733	15.48%	5.852	1.691	9.671	41.80%	
	336	0.276	0.396	0.544	36.94%	5.887	1.851	10.428	24.30%	9.376	2.218	16.519	53.78%	
	720	0.961	0.736	1.728	69.06%	8.814	2.301	15.031	29.52%	15.462	2.882	17.467	74.52%	
FITS	96	0.089	0.212	0.192	20.46%	1.539	0.899	2.634	12.45%	3.975	1.304	6.195	34.14%	0/48
	192	0.184	0.310	0.378	29.81%	2.691	1.208	5.281	16.48%	6.062	1.716	10.265	42.54%	
	336	0.331	0.421	0.717	40.40%	4.795	1.633	9.223	21.93%	9.977	2.259	17.624	55.48%	
	720	0.913	0.732	1.878	67.32%	11.150	2.599	25.127	33.21%	18.319	3.179	26.543	81.12%	
FITS100	96	0.088	0.212	0.196	20.43%	1.531	0.895	2.509	12.42%	3.963	1.302	6.695	34.09%	0/48
	192	0.184	0.310	0.371	29.79%	2.692	1.208	5.098	16.49%	6.053	1.716	10.786	42.51%	
	336	0.331	0.421	0.695	40.44%	4.787	1.633	8.921	21.92%	9.973	2.260	17.936	55.47%	
	720	0.920	0.735	1.937	67.59%	11.161	2.602	25.952	33.22%	18.325	3.181	27.608	81.13%	
FITS10	96	0.101	0.228	0.211	21.78%	1.620	0.929	2.670	12.78%	4.101	1.347	6.437	34.68%	0/48
	192	0.199	0.325	0.408	30.99%	2.787	1.236	5.564	16.77%	6.168	1.747	10.849	42.91%	
	336	0.352	0.436	0.750	41.69%	4.900	1.662	10.084	22.17%	10.095	2.287	17.827	55.81%	
	720	0.993	0.764	2.144	70.22%	11.310	2.630	21.824	33.44%	18.343	3.192	25.342	81.17%	

Table 6: Multivariate input on multivariate columns

Dataset		Exchange Rate				General Dynamics <i>GD</i>				Marathon Oil Corp <i>MRO</i>				Best
Model	Horizon	MSE	MAE	SE	RRMSE	MSE	MAE	SE	RRMSE	MSE	MAE	SE	RRMSE	48
Repeat	96	0.088	0.221	0.179	17.68%	2.180	1.045	4.423	9.57%	2.870	1.261	5.347	20.17%	26/48
	192	0.188	0.333	0.386	26.03%	4.301	1.531	9.898	13.48%	5.362	1.761	10.657	27.86%	
	336	0.370	0.468	0.809	36.49%	7.579	2.093	17.440	17.87%	10.261	2.421	21.428	39.26%	
	720	1.002	0.765	1.831	58.32%	20.163	3.575	46.714	29.08%	23.185	3.680	39.829	63.92%	
DLinear	96	0.099	0.242	0.192	18.82%	2.459	1.170	4.054	10.16%	2.921	1.310	5.254	20.35%	8/48
	192	0.197	0.355	0.404	26.65%	4.335	1.542	9.311	13.53%	5.714	1.828	12.139	28.76%	
	336	0.317	0.458	0.657	33.77%	7.986	2.160	15.729	18.35%	10.830	2.488	24.803	40.33%	
	720	0.981	0.783	0.928	57.69%	17.167	3.397	30.434	26.83%	20.396	3.536	36.740	59.95%	
DLinear + FITS	96	0.107	0.254	0.201	19.52%	2.840	1.247	3.979	10.92%	2.997	1.321	5.574	20.62%	6/48
	192	0.255	0.397	0.360	30.32%	4.944	1.709	9.626	14.45%	5.714	1.825	12.242	28.76%	
	336	0.404	0.506	0.759	38.14%	9.204	2.395	17.318	19.70%	10.638	2.504	19.348	39.97%	
	720	1.057	0.803	1.072	59.89%	17.583	3.280	37.847	27.15%	19.986	3.499	34.045	59.35%	
FITS + DLinear	96	0.106	0.249	0.210	19.42%	2.604	1.161	5.208	10.46%	3.024	1.309	5.617	20.71%	7/48
	192	0.197	0.351	0.333	26.62%	4.419	1.571	9.304	13.66%	5.562	1.819	11.092	28.37%	
	336	0.337	0.460	0.575	34.83%	10.053	2.544	17.222	20.58%	10.474	2.466	23.334	39.66%	
	720	0.873	0.732	1.043	54.42%	17.726	3.311	29.015	27.26%	21.393	3.610	40.060	61.40%	
FITS	96	0.120	0.277	0.264	20.64%	2.392	1.125	4.278	10.03%	3.163	1.385	6.150	21.18%	0/48
	192	0.243	0.393	0.472	29.58%	4.723	1.637	9.919	14.12%	5.701	1.880	11.051	28.72%	
	336	0.405	0.485	0.851	38.17%	8.658	2.267	16.092	19.10%	10.886	2.541	20.806	40.43%	
	720	1.066	0.808	2.415	60.14%	23.621	3.900	59.070	31.47%	22.865	3.720	36.404	63.48%	
FITS100	96	0.119	0.273	0.247	20.54%	2.459	1.153	4.802	10.17%	3.174	1.391	6.469	21.22%	0/48
	192	0.239	0.389	0.443	29.36%	4.738	1.642	10.412	14.15%	5.701	1.881	11.448	28.73%	
	336	0.404	0.487	0.817	38.13%	8.704	2.278	17.656	19.15%	10.892	2.545	20.770	40.44%	
	720	1.048	0.802	2.064	59.64%	23.623	3.919	64.024	31.47%	22.869	3.723	35.469	63.48%	
FITS10	96	0.143	0.303	0.276	22.57%	2.704	1.239	5.193	10.66%	3.457	1.482	5.547	22.14%	1/48
	192	0.244	0.391	0.507	29.61%	5.167	1.749	12.032	14.77%	6.021	1.957	11.073	29.52%	
	336	0.422	0.505	0.852	38.97%	9.303	2.394	21.951	19.80%	11.191	2.598	20.616	40.99%	
	720	1.065	0.822	1.739	60.12%	24.351	4.081	48.666	31.95%	22.852	3.737	33.523	63.46%	

Table 7: Multivariate input on univariate column

Dataset		Exchange Rate				General Dynamics <i>GD</i>				Marathon Oil Corp <i>MRO</i>				Best
Model	Horizon	MSE	MAE	SE	RRMSE	MSE	MAE	SE	RRMSE	MSE	MAE	SE	RRMSE	48
Repeat	96	0.088	0.221	0.179	17.68%	2.180	1.045	4.423	9.57%	2.870	1.261	5.347	20.17%	26/48
	192	0.188	0.333	0.386	26.03%	4.301	1.531	9.898	13.48%	5.362	1.761	10.657	27.86%	
	336	0.370	0.468	0.809	36.49%	7.579	2.093	17.440	17.87%	10.261	2.421	21.428	39.26%	
	720	1.002	0.765	1.831	58.32%	20.163	3.575	46.714	29.08%	23.185	3.680	39.829	63.92%	
DLinear	96	0.099	0.242	0.192	18.82%	2.459	1.170	4.054	10.16%	2.921	1.310	5.254	20.35%	8/48
	192	0.197	0.355	0.404	26.65%	4.335	1.542	9.311	13.53%	5.714	1.828	12.139	28.76%	
	336	0.317	0.458	0.657	33.77%	7.986	2.160	15.729	18.35%	10.830	2.488	24.803	40.33%	
	720	0.981	0.783	0.928	57.69%	17.167	3.397	30.434	26.83%	20.396	3.536	36.740	59.95%	
DLinear + FITS	96	0.107	0.254	0.201	19.52%	2.840	1.247	3.979	10.92%	2.997	1.321	5.574	20.62%	6/48
	192	0.255	0.397	0.360	30.32%	4.944	1.709	9.626	14.45%	5.714	1.825	12.242	28.76%	
	336	0.404	0.506	0.759	38.14%	9.204	2.395	17.318	19.70%	10.638	2.504	19.348	39.97%	
	720	1.057	0.803	1.072	59.89%	17.583	3.280	37.847	27.15%	19.986	3.499	34.045	59.35%	
FITS + DLinear	96	0.106	0.249	0.210	19.42%	2.604	1.161	5.208	10.46%	3.024	1.309	5.617	20.71%	7/48
	192	0.197	0.351	0.333	26.62%	4.419	1.571	9.304	13.66%	5.562	1.819	11.092	28.37%	
	336	0.337	0.460	0.575	34.83%	10.053	2.544	17.222	20.58%	10.474	2.466	23.334	39.66%	
	720	0.873	0.732	1.043	54.42%	17.726	3.311	29.015	27.26%	21.393	3.610	40.060	61.40%	
FITS	96	0.120	0.277	0.264	20.64%	2.392	1.125	4.278	10.03%	3.163	1.385	6.150	21.18%	0/48
	192	0.243	0.393	0.472	29.58%	4.723	1.637	9.919	14.12%	5.701	1.880	11.051	28.72%	
	336	0.405	0.485	0.851	38.17%	8.658	2.267	16.092	19.10%	10.886	2.541	20.806	40.43%	
	720	1.066	0.808	2.415	60.14%	23.621	3.900	59.070	31.47%	22.865	3.720	36.404	63.48%	
FITS100	96	0.119	0.273	0.247	20.54%	2.459	1.153	4.802	10.17%	3.174	1.391	6.469	21.22%	0/48
	192	0.239	0.389	0.443	29.36%	4.738	1.642	10.412	14.15%	5.701	1.881	11.448	28.73%	
	336	0.404	0.487	0.817	38.13%	8.704	2.278	17.656	19.15%	10.892	2.545	20.770	40.44%	
	720	1.048	0.802	2.064	59.64%	23.623	3.919	64.024	31.47%	22.869	3.723	35.469	63.48%	
FITS10	96	0.143	0.303	0.276	22.57%	2.704	1.239	5.193	10.66%	3.457	1.482	5.547	22.14%	1/48
	192	0.244	0.391	0.507	29.61%	5.167	1.749	12.032	14.77%	6.021	1.957	11.073	29.52%	
	336	0.422	0.505	0.852	38.97%	9.303	2.394	21.951	19.80%	11.191	2.598	20.616	40.99%	
	720	1.065	0.822	1.739	60.12%	24.351	4.081	48.666	31.95%	22.852	3.737	33.523	63.46%	

Table 8: Univariate input on univariate column

Note: While method 7 has access to other variables during training, both methods only optimize for the target column since their loss functions ignore non-target predictions. This explains why their tables show such similar results – they’re effectively doing the same optimization. Attempting to make method 7 mode predict or share weights from all columns resulted in substantially worse performance, probably because it diverts the model’s focus from the actual target variable.

6 Discussion

6.1 What does the Hurst Exponent really measure?

In theory, a random walk should yield a Hurst Exponent of approximately $H = 0.5$. However, as can be seen from figure 10 below, data which clearly has a non-random underlying signal, can also yield a $H \approx 0.5$ when non-stationary and noisy enough.

Although one of the steps when estimating H is detrending the (entire) signal, we notice that if a noisy signal has a, or more, linear trends, the Hurst exponent also seems to become an unreliable measure of unpredictability. Similarly, given that the H value for simple sine waves or combinations of sine waves without noise indicates "anti-persistent behavior" and is also lower than that of an actual (approximated) random walk, one should exercise caution before using it as a definitive measure.

In retrospect, it seems that the Hurst Exponent seems less useful for quantifying predictability in terms of trends or random walk behaviour but instead serves as a decent measure of the amount of noise in a signal (ref. figure 9), with its simple to understand value between 0 (white noise) and 1 (no noise).

In any case, the Hurst exponent is a quick, inexpensive, and effective calculation, and while inconclusive by itself, it can serve in conjunction with other indicators like the autocorrelation function (ACF) or Naïve Forecast (NF) baseline benchmark to paint a broader picture.

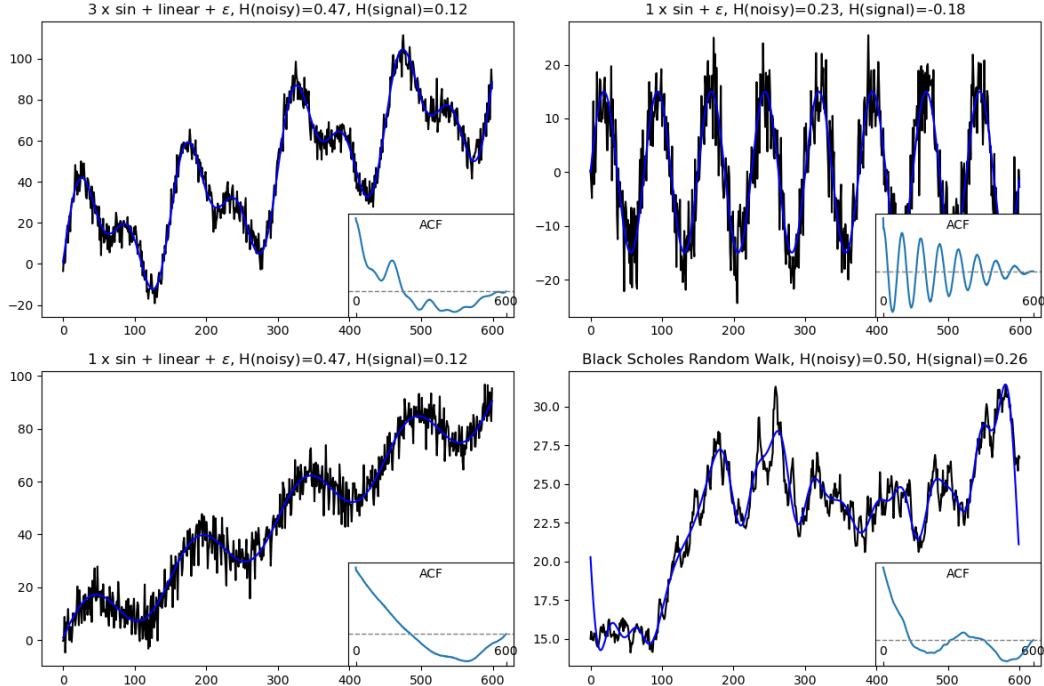


Figure 10: Hurst Exponent for different types of synthetically generated data. The ACF plot is of the noisy (black) graph. The "Signal" in the random walk has had a simple low pass filter applied.

6.2 Autocorrelation Functions (ACF) – an insight into why FITS performs differently

As explained earlier, FITS upscales a signal to a longer length using the frequency domain, relying on the fast Fourier transform for the translation back and forth between time and frequency domain, with the weights learning the proper interpolation between the input and output frequencies.

While a Fourier transformation can decompose arbitrary signals into frequency components, not all signals are necessarily modeled appropriately in the frequency domain. Fourier transformation excels in analyzing stationary signals—those whose statistical properties do not change over time. In such cases, the signal can be accurately represented as a sum of sinusoids with different frequencies, amplitudes, and phases. But when data has little to no periodic attributes, either globally or locally,

FITS will not be able to model the data when, as the decomposed frequency components have just been used to fit to non-periodic attributes and they therefore hold no forecasting value. Most of the datasets we've tested are modelled well in the frequency domain, whereas exchange rate and the stock data in particular are unsuitable for FITS with naive baselines consistently performing better.

Even though the channels in a dataset might be heavily correlated in a non periodic manner, FITS will not be able to model these correlations. This limits the utility of FITS in time series applications where the data lacks these periodic or seasonal attributes, such as systems where the input is random such as randomly applied speeder or brake pressure in relation to the acceleration of an electric car, where those attributes are directly correlated, but non-periodic.

Autocorrelation functions, have shown to be incredibly informative (ref. figure 16) in understanding why FITS performs better on datasets like (ETTh1, ETTh2, ETTm1, ETTm2, Weather, Electricity, & Traffic) compared to other datasets which were omitted in the original paper. These datasets show clear periodicity, often coinciding with 24h cycles or compared to the more uninformative ACF plot of the Exchange Rate dataset where the correlation seems non periodic and converges towards zero correlation. And the datasets where there is clear periodicity from the ACF plots we can also see that FITS is able to perform really well.

6.3 Evaluation of ARIMA

In the results section, specifically table 1, ARIMA is included due to it being a standard model for statistical forecasting. The results of ARIMA do not look particularly favorable, however, compared to the other models, but we assert that it is due to the fact that ARIMA models do not perform well on the long-range forecasting horizons, the uncertainty grows at larger time steps. Depending on the values of p and q , ARIMA achieves best performs on short horizons, especially if no seasonality is in the data.

Along with that fact, we do not report ARIMA results for the forecast horizons of over 96, equally much due to the performance of ARIMA at such forecast lengths, but also due to the fact that the computation time for it is abnormally high compared to FITS - especially when the data contains a large amount of channels.

The FITS model we do a lot better for the aforementioned cases, as it represents the data in the frequency domain after a Fourier transformation of the data, and then learns a linear interpolation of the frequencies; the ARIMA model is fitted only on the observed part of the test data, and trained on the forecasting part. This is unlike the FITS model, which as a training set on which it trains both reconstruction and forecasting.

6.4 FITS and the various attempts to improving it

All of the models incorporating additional layers in the FITS interpolation step performed worse at the longest forecast horizon 2 with a far higher parameter count. CReLU deteriorated performance to a higher degree than ModReLU most likely due to implicit change in the phase of the complex numbers 5.

The real valued complex upscaler also performed quite poorly on everything but the shorter forecast horizons on ETTh1, likely due to similar problems as the CReLU model but now causing problems throughout the entire network as the complex numbers no longer had proper representation. At shorter forecast horizons [96, 192, 336] the results are somewhat inconclusive as a minor statistically significant improvement was achieved 4 indicating that the ModReLU variant of deep FITS is able to learn more features in the frequency representation of the signal compared to a single weight matrix. Some of the reasons for the small improvements might simply be because the lowest MSE possible on the ETT datasets is around the previous FITS scores due to the noise inherent in the datasets and much further improvement might not be possible. On the longer forecast horizon [720] none of the models convincingly beat the default FITS implementation could be due to several factors. The increased model complexity of the deep FITS variants may not be capturing meaningful long-term patterns that generalize well to the test set for extended forecast horizons. This suggests that the added complexity might not be necessary or beneficial for longer-term predictions in these datasets. Additionally, the default FITS implementation may already be capturing the most relevant frequency components for long-term forecasting, with the additional complexity introduced by the deep variants not adding substantial value in terms of identifying more meaningful long-term patterns in the frequency domain.

The use of the Fourier transformation in the pipeline seems to be generally an excellent way to help forecast better, especially on data that shows seasonality and/or periodic patterns. This is the case for most of the datasets that the performance is evaluated on. More on this can be read in the data assessment section of Appendix 8. The addition of extra layers for processing in our experiments showed that these were unnecessary and that the default FITS implementation may already be capturing the most relevant frequency components for long-term forecasting. The additional complexity introduced by the deep variants might not be adding substantial value in terms of identifying more meaningful long-term patterns in the frequency domain.

6.5 DLinear vs FITS and hybrid model performance

DLinear and FITS represent two relatively simple yet very effective approaches to time series forecasting compared to prior models. While both utilize linear layers, they differ significantly in how they transform and process the input data.

In table 6 and 1, we compared the results of DLinear vs FITS, which reveal some interesting patterns in the performance of these models.

On datasets with clear periodic patterns (e.g., ETT datasets, Weather, Electricity, Traffic), FITS often outperforms DLinear, especially at longer forecast horizons. This aligns with FITS' strength in capturing frequency-domain patterns.

For datasets with less obvious periodicity or more random-resembling behavior (like the pricing datasets in table 6), DLinear tends to perform better than FITS, though often only marginally better than the naive baseline.

It would seem from the SE scores, that if one prioritizes what a stock price looks like in 192, 336, or 720 days, irregardless of the fluctuations along the way, the DLinear + FITS and FITS + DLinear hybrid models seem like the better option for this purpose. However, for capturing non-periodic and prices without drift like the Exchange Rate dataset, DLinear slightly outperforms the Naive Repeat baseline, compared to the rest.

The hybrid models also tend to outperform each individual model at longer forecast horizons and for multivariate targets. This suggests that the two approaches combined can complement each other, with FITS capturing periodic patterns and DLinear handling non-periodic trends and seasonality. It also suggests that having more cross-column context when predicting each column can yield better results, assuming the columns are uniquely informative about each other.

This improvement, although minor, suggests that there is potential in combining frequency-domain, time-domain, and perhaps exploring other mathematical representations of the data for gain new insights into the patterns of the time series. Future research could explore more sophisticated ways of integrating these complementary techniques to create more robust and versatile forecasting models.

In conclusion, it seems that architecturally simpler models beat bigger and more advanced models on time series forecasting tasks when properly decomposed into components that provide signals about the data's trend, fluctuations, and periodicity.

7 Conclusion

In this report, we have explored the performance of the model Frequency Interpolation for Time Series Analysis (FITS), testing its performance on common datasets as well as \$GD and \$MRO. The FITS model leverages the Fourier transform to project time-domain data into the frequency domain, capturing periodic patterns that can be highly informative in accurate forecasting on certain types of datasets. Through various experiments and comparisons with other models, including the autoregressive integrated moving average (ARIMA) model and hybrid models like DLinear, we have explored the strengths and limitations of FITS.

We have found that FITS performs very well on datasets with clear periodicity, such as the ETT, Electricity and Weather datasets, where it often outperforms other larger models especially at longer forecast horizons. The frequency representation inherent in FITS allows it to easily decompose and reconstruct signals which results in it being able to easily capture patterns other models might miss. However, FITS struggles with datasets that exhibit random or non-periodic behavior such as stock prices, where simpler models or naive baselines perform better. Likewise if the base period of a signal is too long compared to the window used for the Fourier transform, FITS is unable to fully capture the underlying signal instead learning higher frequency representations.

We also experimented with enhancements to the FITS model, such as deep FITS with ModReLU or CReLU, deep FITS after upscaler, and linear layer bypass FITS. Some of these showed promise at shorter forecast horizons, but the improvements were often marginal and came at the cost of increased model complexity. Which indicates that the original FITS implementation is already close to optimal for its intended purpose and when operating only in the frequency domain.

Additionally, we evaluated the use of hybrid models, combining the strengths of FITS and DLinear, which showed improved performance in some cases. These models effectively captured both periodic and non-periodic components of the data, making them robust for a wider range of time series forecasting tasks.

In conclusion, FITS stands out as an efficient model for time series which can be modelled appropriately using the frequency representation. It is particularly useful for longer forecasting horizons compared to previous models while being exceedingly simple and efficient.

References

- [Fam70] Eugene Fama. “Efficient Capital Markets: A Review of Theory and Empirical Work”. In: *Journal of Finance* 25.2 (1970), pp. 383–417. URL: <https://EconPapers.repec.org/RePEc:bla:jfinan:v:25:y:1970:i:2:p:383-417> (cit. on p. 5).
- [BJ76] George E.P. Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, 1976 (cit. on pp. 3, 39).
- [Ham94] James Douglas Hamilton. *Time series analysis*. en. Princeton, NJ: Princeton University Press, Jan. 1994 (cit. on pp. 3, 42–44).
- [Ros13] Barbara Rossi. “Exchange Rate Predictability”. In: *Journal of Economic Literature* 51.4 (Dec. 2013), pp. 1063–1119. DOI: [10.1257/jel.51.4.1063](https://doi.org/10.1257/jel.51.4.1063). URL: <https://www.aeaweb.org/articles?id=10.1257/jel.51.4.1063> (cit. on p. 5).
- [Tra+18] Chiheb Trabelsi et al. *Deep Complex Networks*. 2018. arXiv: [1705.09792](https://arxiv.org/abs/1705.09792). URL: <https://arxiv.org/abs/1705.09792> (cit. on pp. 11, 12).
- [Zho+21] Haoyi Zhou et al. *Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting*. 2021. arXiv: [2012.07436](https://arxiv.org/abs/2012.07436). URL: <https://arxiv.org/abs/2012.07436> (cit. on p. 3).
- [Wu+22a] Haixu Wu et al. *Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting*. 2022. arXiv: [2106.13008 \[cs.LG\]](https://arxiv.org/abs/2106.13008) (cit. on pp. 3, 17).
- [Wu+22b] Haixu Wu et al. “Timesnet: Temporal 2d-variation modeling for general time series analysis”. In: *The eleventh international conference on learning representations*. 2022 (cit. on p. 2).
- [Zen+22] Ailing Zeng et al. *Are Transformers Effective for Time Series Forecasting?* 2022. arXiv: [2205.13504 \[cs.AI\]](https://arxiv.org/abs/2205.13504) (cit. on pp. 4, 5, 20).
- [Zho+22] Tian Zhou et al. *FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting*. 2022. arXiv: [2201.12740 \[cs.LG\]](https://arxiv.org/abs/2201.12740) (cit. on p. 3).
- [Nie+23] Yuqi Nie et al. *A Time Series is Worth 64 Words: Long-term Forecasting with Transformers*. 2023. arXiv: [2211.14730](https://arxiv.org/abs/2211.14730). URL: <https://arxiv.org/abs/2211.14730> (cit. on pp. 4, 5, 20).
- [SWW23] Li Shen, Yunling Wei, and Yangzhu Wang. “GBT: Two-stage transformer framework for non-stationary time series forecasting”. In: *Neural Networks* 165 (2023), pp. 953–970. ISSN: 0893-6080. URL: <https://arxiv.org/abs/2307.08302> (cit. on p. 18).
- [ZXZ24] Zhijian Xu, Ailing Zeng, and Qiang Xu. *FITS: Modeling Time Series with 10k Parameters*. 2024. arXiv: [2307.03756](https://arxiv.org/abs/2307.03756). URL: <https://arxiv.org/abs/2307.03756> (cit. on pp. 2, 9, 17, 19, 20).

8 Appendix

Weights

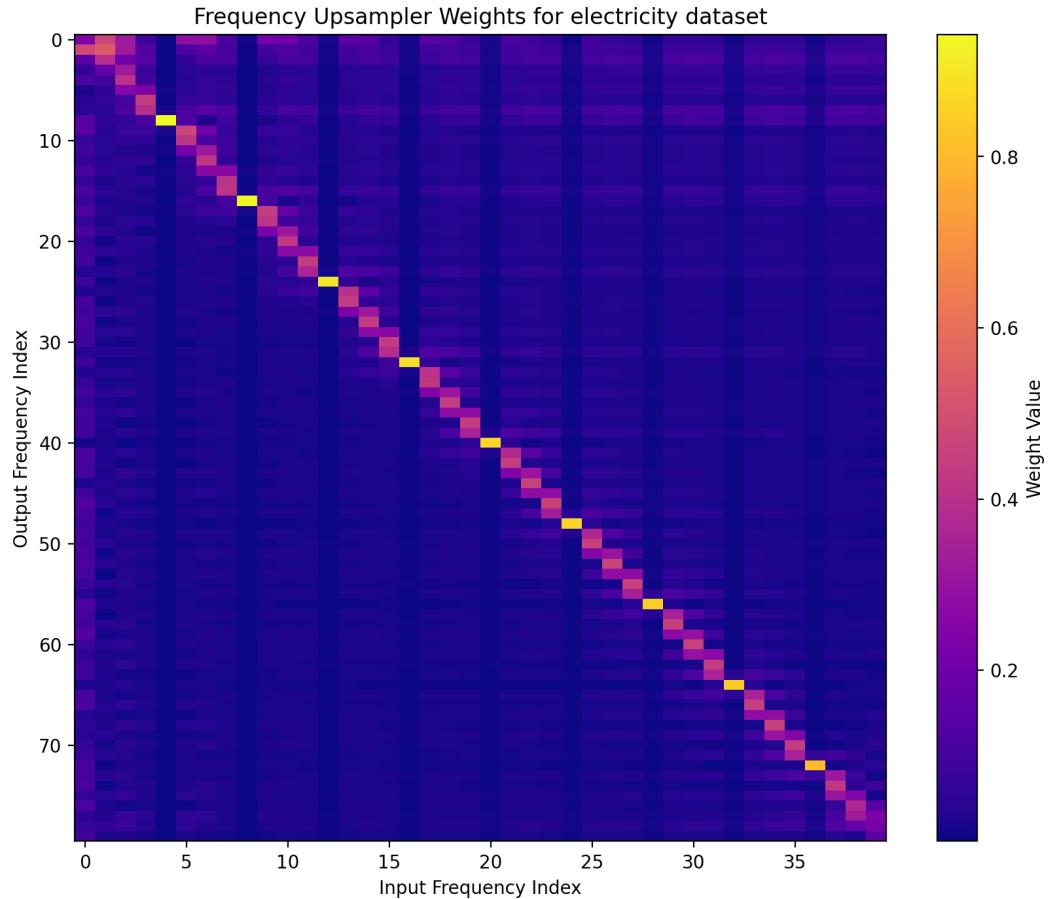


Figure 11: Weights for the electricity dataset [→ 3.3]

Data assessment

A problem that goes throughout most of the datasets, is their documentation, which is rather lacking. The lack of proper documentation makes it hard to make assumptions about the nature of the datasets, in the sense of how they will behave in the future. But even though the datasets are not documented properly, it is still possible to use them for evaluating our time series models as it is their performance in relation to each other they get evaluated on.

ETT datasets

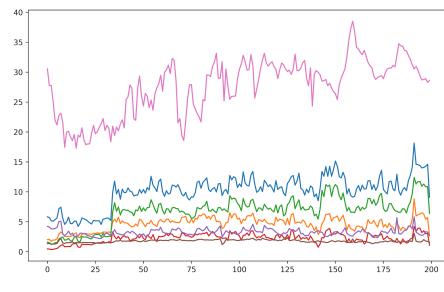


Figure 12: Plot of the channels in ETTh1

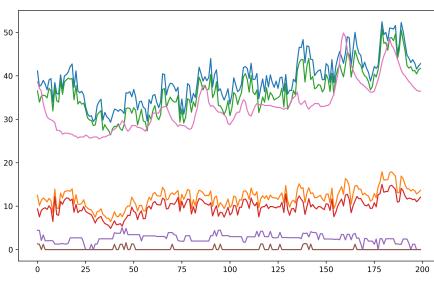


Figure 13: Plot of the channels in ETTh2

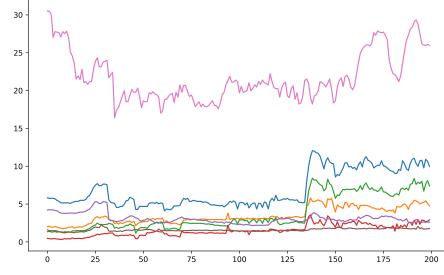


Figure 14: Plot of the channels in ETTm1

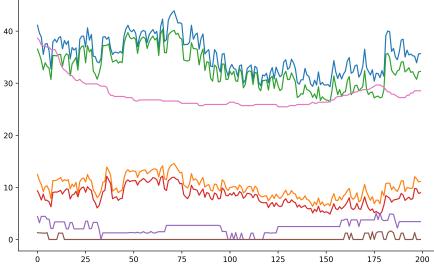


Figure 15: Plot of the channels in ETTm2

These plots are all clippings of 200 time steps from each of their respective whole datasets. ETThX and ETTmX are from the same datasets, where ETTmX samples each 15th minute, where ETThX samples once every hour, the X indicates the source of the data sampling.

The column OT is the column we forecast on since it denotes the oil temperature of the transformer station it is recorded at. It is quite apparent that all of the channels are quite correlated apart from the OT channel from doing a visual assessment alone.

Price datasets

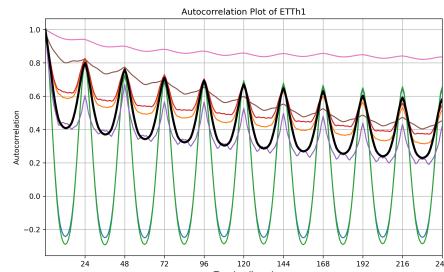
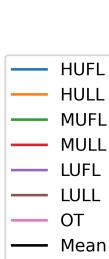


Figure 16: Time lag correlation, ETTh1

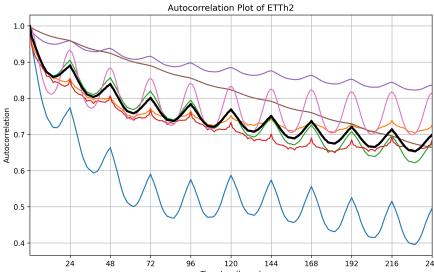


Figure 17: Time lag correlation, ETTh2

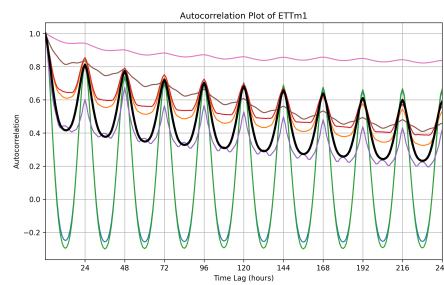


Figure 18: Time lag correlation, ETTm1

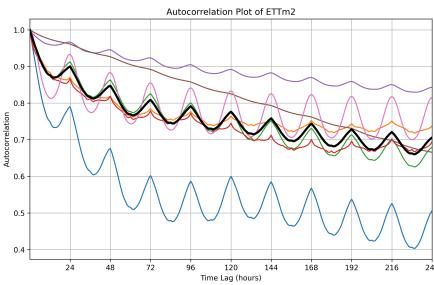
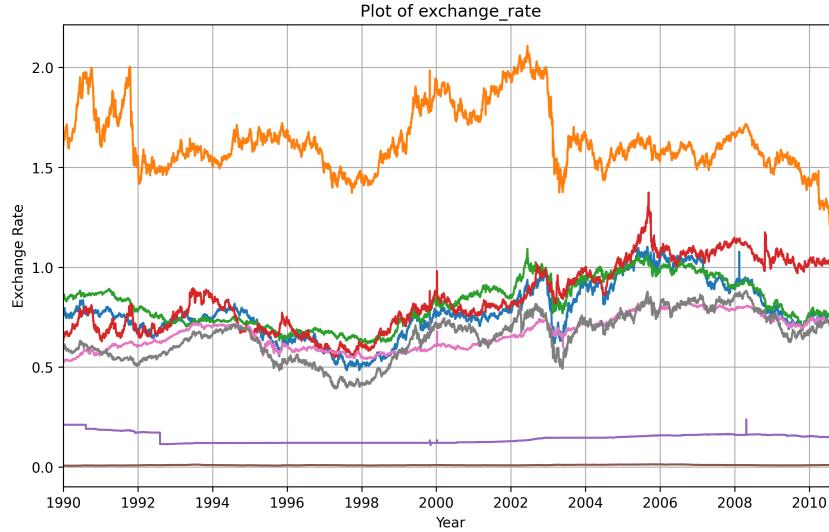
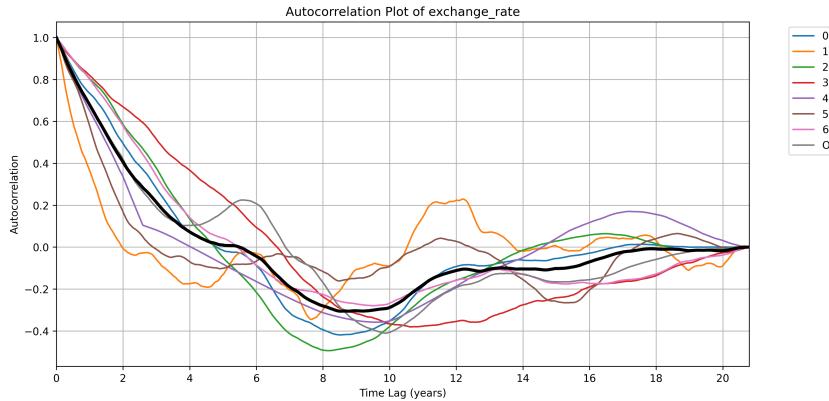


Figure 19: Time lag correlation, ETTm2

Afterwards we performed auto correlation on the data and saw that peaks between different channels happened at similar positions in the data and at high correlation values. This indicates strong correlation between the channels, which we also expected from our first visual assessment of the dataset. We can also observe that OT is highly correlated with its earlier and slightly positively correlated with the other channels as it shows slight peaking at the same positions. Because of the periodic peaking with high time lag correlation this indicates high periodicity in the data, which in theory should mean that there are frequencies our models should be able to pick up.



This dataset consists of the exchange rates between different currencies. It isn't known which channel corresponds to which currency. On this small cutout of the data, there seems to be no discernible signs of correlation or periodicity between the channels.



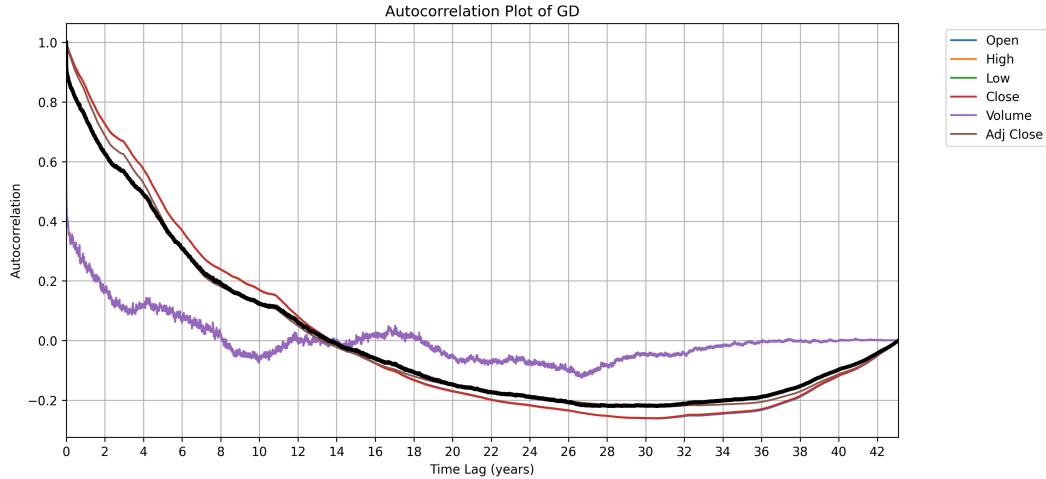


Figure 20: Autocorrelation plot of \$GD

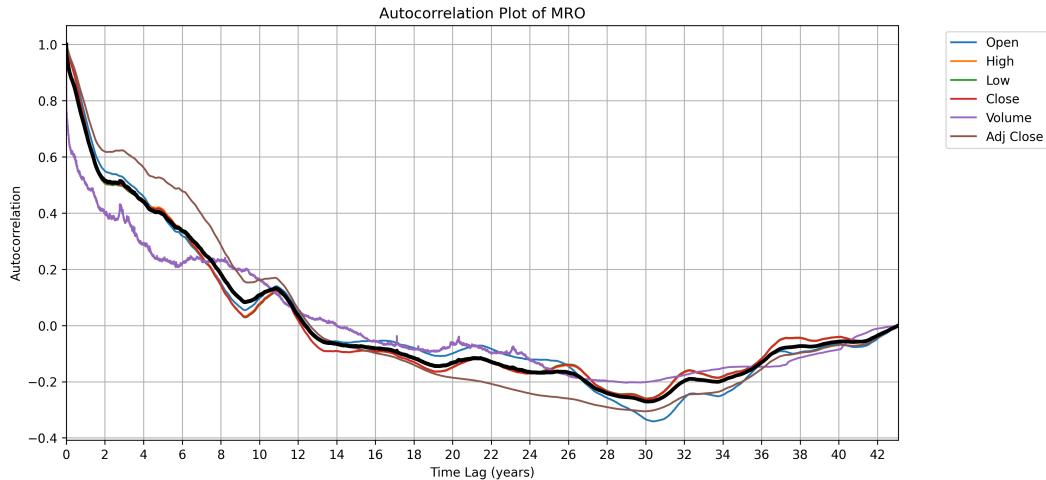
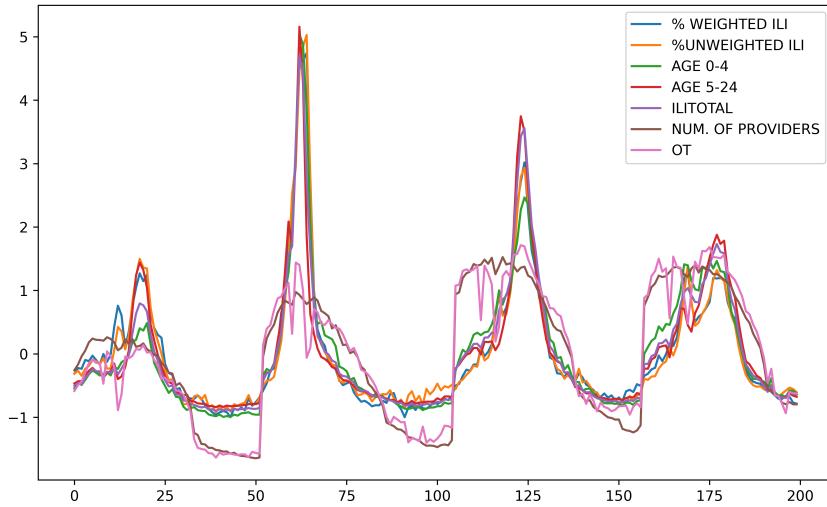
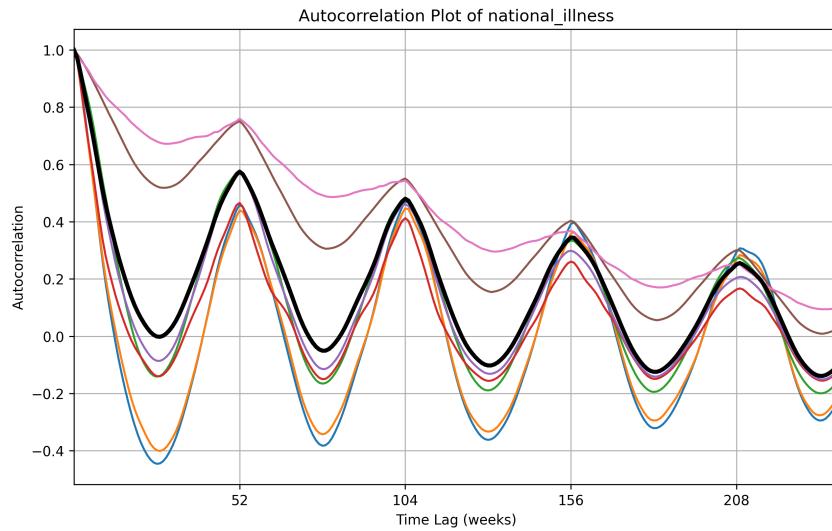


Figure 21: Autocorrelation plot of \$MRO

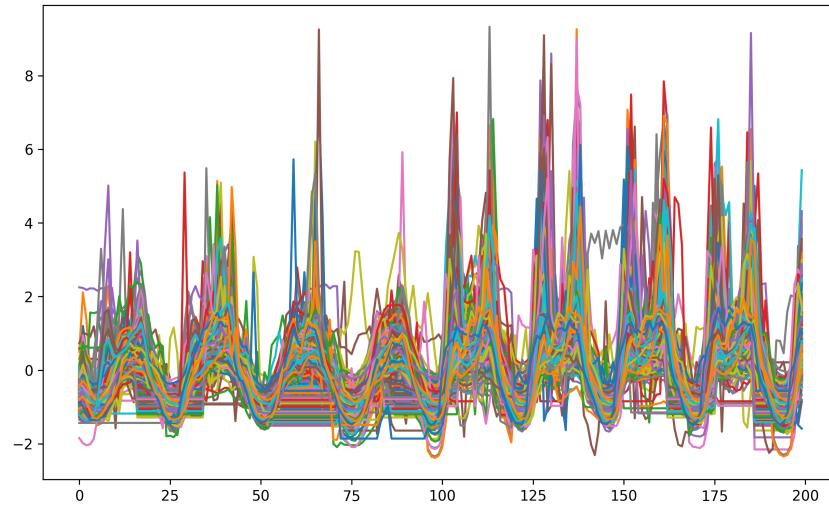
When performing auto correlation on the dataset we observe that there is no periodicity, but quite high correlation between previous values in most channels, except for channel 1, but the correlation is slowly diverging, which could indicate that it would be hard to make predictions based on frequencies.



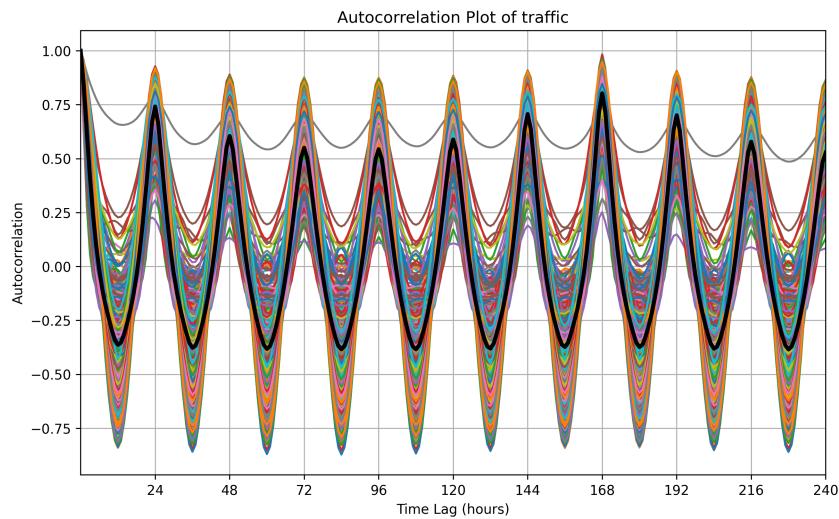
It is quite apparent that there are infection patterns present as the peaks of the dataset seem to peak around the same time every year, as the x-axis represents weeks and there is a heavy increase in value of the OT at the same time of year. There is also peaking patterns in the rest of the dataset



We chose to plot the time lag in weeks because of the assumption that infection numbers are seasonal. The autocorrelation plot shows peaking around the same time of year (52 weeks) each time, which indicates that the dataset is periodic. OT (pink) also shows periodicity, though its prediction value lessens over time which makes sense as infection patterns change from each year.



The traffic dataset shows quite a bit of periodicity every 24 hours. This also makes sense as traffic patterns are quite affected by the time of day.



Performing time correlation on the dataset we can observe heavy periodicity with peaks sitting around the same time of day each day. Again this would indicate that there are predictable frequencies in the data which FITS will be able to effectively forecast on.

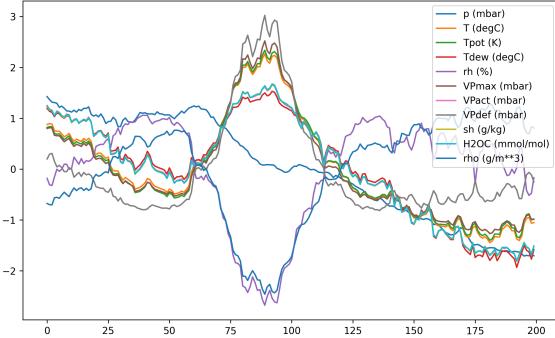


Figure 22: Plot, which is part of the weather dataset, the X-axis is Time, where each time step is 10 minutes

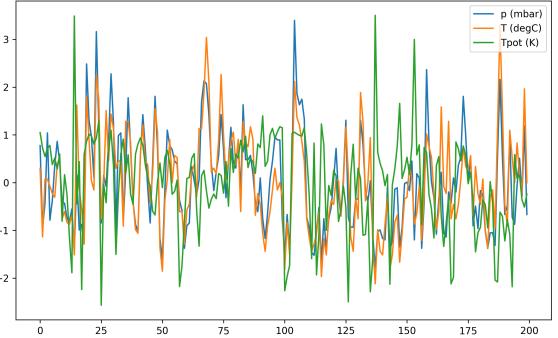


Figure 23: Plot, which is part of the weather dataset, the X-axis is Time, where each time step is 10 minutes

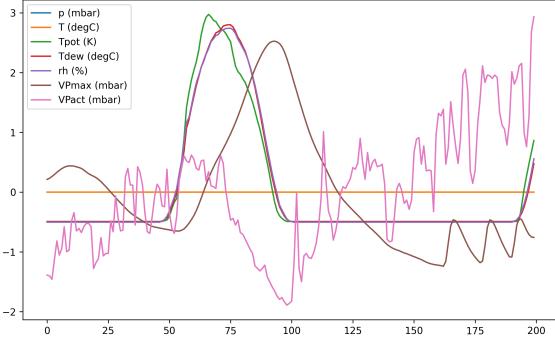


Figure 24: Plot, which is part of the weather dataset, the X-axis is Time, where each time step is 10 minutes

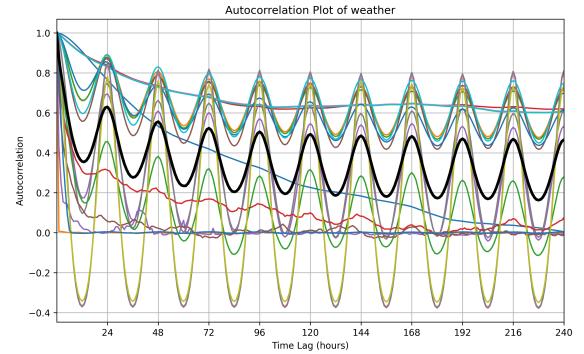


Figure 25: Time lag correlation of all channels

We split the weather data in several plots for better readability. We can also observe from the plots that there seems to be quite high correlation between some of the channels. This makes perfectly sense as weather channels are heavily correlated such as water content and dew point. From the auto correlation plot we can observe heavy correlation and periodicity between most of the channels, where some of the channels converge to a correlation of 0.

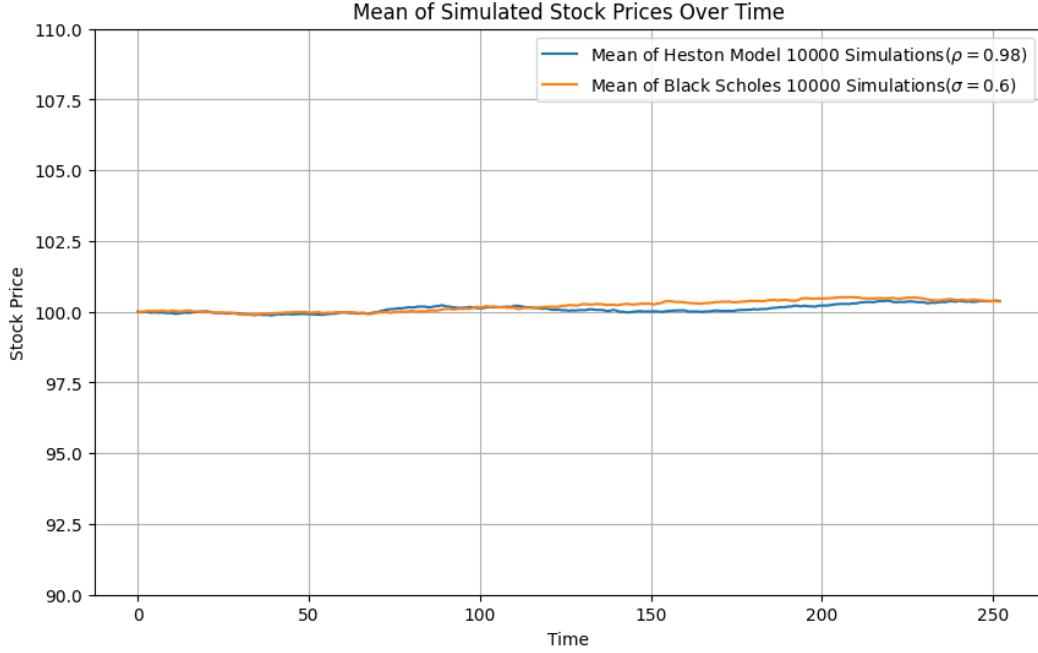


Figure 26: Infinite random walks \implies naive repeat ([compare_random_walk.ipynb](#)) [$\rightarrow 2.3$]

Fast Fourier Transform algorithm [$\rightarrow 3.1.2$]

Implementation based on [The Fast Fourier Transform \(FFT\): Most Ingenious Algorithm Ever?](#).

- Bit-Reversal Permutation:** Start by reordering the input array using a bit-reversal permutation where the binary representation of the indices are reversed resulting in a new index. This step rearranges the original data points so that they align with their final positions early in the process, facilitating the efficient combination in later steps.

$$x'_{\text{bit-reversed}(k)} = x_k, \quad \text{for all } k \in [0, N - 1] \quad (3)$$

- Divide and Conquer:** Recursively divide the reordered array into smaller arrays. the lenght of the current data input is larger than N , this step splits the data into two halves, each of size $N/2$, and applies the FFT recursively.
- Butterfly Combinations:** For each level of recursion, combine the DFTs of smaller arrays using the butterfly operations, which are simple add-and-subtract operations with twiddle factor multiplications which are constant trigonometric coefficients corresponding to the N roots of unity for the given input:

$$\begin{aligned} X_k &= E_k + O_k \cdot e^{-2\pi ik/N}, \quad \text{for } k = 0, \dots, \frac{N}{2} - 1 \\ X_{k+\frac{N}{2}} &= E_k - O_k \cdot e^{-2\pi ik/N}, \quad \text{for } k = 0, \dots, \frac{N}{2} - 1 \end{aligned} \quad (4)$$

where E_k and O_k are the DFTs of the even and odd indexed parts of the array, respectively.

- Recursive Computation:** Continue the recursive process until the full DFT is computed. When the base case ($N = 1$) is reached, the DFT of a single element is the element itself.
- Combine Results:** Combine all small DFT results back into the final DFT of the original array using the relationships established in the butterfly operations.

In pure Python the FFT and the inverse FFT it can be expressed as follows:

```

1 def FFT(x):
2     # Must to be a power of 2
3     n = len(x)
4
5     # Base case
6     if n == 1:
7         return x
8     # Twiddle factor
9     w = exp(-2j * pi/n)
10    # Even indexed coefficients
11    xe = x[0::2]
12    # Odd indexed coefficients
13    xo = x[1::2]
14
15    # Recursive calls
16    ye = FFT(xe)
17    yo = FFT(xo)
18
19    # Initialize the result
20    y = [0] * n
21    for k in range(n//2):
22        # Compute twiddle
23        factor
24        wk = w ** k
25        # Butterfly
26        combinations
27        y[k] = ye[k]+yo[k]*wk
28        y[k+n//2] = ye[k]-yo[k]
29        ]*wk
30    return y
31
32 def IFFT(x):
33     # Must to be a power of 2
34     n = len(x)
35
36     # Base case
37     if n == 1:
38         return x
39     # Twiddle factor
40     w = exp(2j * pi/n)
41     # Even indexed coefficients
42     xe = x[0::2]
43     # Odd indexed coefficients
44     xo = x[1::2]
45
46     # Recursive calls
47     ye = IFFT(xe)
48     yo = IFFT(xo)
49
50     # Initialize the result
51     y = [0] * n
52     for k in range(n // 2):
53         # Compute twiddle
54         factor
55         wk = w ** k
56         # Butterfly
57         combinations
58         y[k] = ye[k]+yo[k]*wk
59         y[k+n//2] = ye[k]-yo[k]
60     return y

```

Note that this example doesn't normalize the data from the inverse FFT nor does it implement the bit-reversal permutation. The inverse FFT can be expressed similarly to FFT just with the opposite sign in the twiddle factor and afterward should be scaled by $\frac{x}{n}$ for $x \in \mathbf{x}$.

Autoregressive Integrated Moving Average Models (ARIMA) [[→ 1.2.1](#)] [[→ 3.9](#)]

In this chapter, an overview in short detail the individual components of the complete ARIMA model, as well as estimation of the parameters using a maximum likelihood framework, will be described.

Autoregressive Model (AR)

The autoregressive model is a type of linear model in which the current value is expressed as a finite, linear combination of the previous values of the model \tilde{z} and a error term of white noise ϵ . The term \tilde{z}_t is defined as the deviation from the mean of the data $\tilde{z}_t = z_t - \mu$. The autoregressive model with weights ϕ can be expressed as:

$$\tilde{z}_t = \sum_{i=1}^p \phi_i \tilde{z}_{t-i} + \epsilon_t \quad (5)$$

Where the order p is the number of lagged values of the time series included in the model and indicates how many previous time steps the current value of the series is regressed upon.

In contrast to a linear regression model can be stated as:

$$\tilde{z} = \sum_{i=1}^p \phi_i \tilde{x}_i + \epsilon$$

Where the dependent variable \tilde{z} is regressed on the independent variables \mathbf{x} with the error ϵ .

The primary difference between the autoregressive model and the linear regression model, is that the variable z is regressed on previous values of itself – this is why it is called an *autoregressive* model.

Using the backshift operator defined as $\mathbf{B}^k x_t = x_{t-k}$, we can express the autoregressive operator of order p in a more concise format as:

$$\phi(\mathbf{B}) = 1 - \sum_{i=1}^p \phi_i \mathbf{B}^i$$

This allows us to express the autoregressive model as:

$$\phi(\mathbf{B}) \tilde{z}_t = \epsilon_t \quad (6)$$

Which is equivalent to:

$$\tilde{z}_t = \phi^{-1}(\mathbf{B}) \epsilon_t$$

This gives us the model parameters $\mu, \phi_1, \phi_2, \dots, \phi_p, \sigma_\epsilon^2$ which is a total of $p + 2$ parameters.

The model is stationary, meaning constant mean μ and variance σ_ϵ^2 , if the roots to the characteristic polynomial of the autoregressive equation $\phi(\mathbf{B}) = 1 - \sum_i^p \phi_i \mathbf{B}^i = 0$, are greater than 1 [BJ76]. For the AR(1) process, this imposes the condition of $\phi_1 < 1$, for example.

Moving Average Model (MA)

Another important model for time series deviation representation is the moving average model which is defined as:

$$\begin{aligned} \tilde{z}_t &= \epsilon_t - \theta_1 \epsilon_{t-1} - \dots - \theta_q \epsilon_{t-q} \\ &= \epsilon_t - \sum_{i=1}^q \theta_i \epsilon_{t-i} \end{aligned}$$

In this model, \tilde{z}_t represents the deviation of the time series from its mean value at time t . The term ϵ_t denotes the white noise error at time t , while $\theta_1, \theta_2, \dots, \theta_q$ are parameters that quantify the influence of the past q error terms on the current value. The moving average model employs a linear combination of these q past error terms to describe the current deviation. This approach captures dependencies between consecutive errors, thereby enhancing forecasting accuracy.

Using the backshift operator \mathbf{B} , it can concisely be written as:

$$\tilde{z}_t = \theta(\mathbf{B}) \epsilon_t$$

where the notation for the backshift operator is the same as in equation (6) with ϕ replaced by θ . In the moving average model, the parameters $\theta_1, \dots, \theta_q$ are all unknown, as well as the mean μ and the variance σ_t^2 , and have to be estimated.

Mixed Autoregressive-Moving Average Models (ARMA)

The ARMA model consists of an autoregressive component and a moving average component mixed together to form the model form of

$$\tilde{z}_t = \phi_1 \tilde{z}_{t-1} + \dots + \phi_p \tilde{z}_{t-p} + \epsilon_t - \theta_1 \epsilon_{t-1} - \dots - \theta_q \epsilon_{t-q} \quad (7)$$

$$= \epsilon_t + \sum_{i=1}^p \phi_i \tilde{z}_{t-i} - \sum_{j=1}^q \theta_j \epsilon_{t-j} \quad (8)$$

As such, it has a total of $p + q + 2$ parameters that need to be estimated from the data, which are: $\mu, \epsilon_t^2, \phi_i, \theta_j$. [BJ76]

ARMA parameter estimation

AR(p)

Now consider the general AR model:

$$z_t = \mu + \phi_1 z_{t-1} + \dots + \phi_p z_{t-p} + \epsilon_t \quad (9)$$

If we consider the same conditional maximum likelihood framework as before, we now assert that the p first observations are deterministic. The likelihood function, $\Theta = (\mu, \sigma^2, \phi_1, \dots, \phi_p)$ is now:

$$\mathcal{L}(\Theta) = \prod_{t=p+1}^T \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(z_t - \mu - \phi_1 z_{t-1} - \phi_2 z_{t-2} - \cdots - \phi_p z_{t-p})^2}{2\sigma^2}\right) \quad (10)$$

And the equivalent log likelihood functions, simplified directly and omitting intermediary steps:

$$\log \mathcal{L}(\Theta) = -\frac{T-p}{2} \cdot \log(2\pi) - \frac{T-p}{2} \cdot \log(\sigma^2) - \sum_{t=p+1}^T \frac{(z_t - \mu - \phi_1 z_{t-1} - \phi_2 z_{t-2} - \cdots - \phi_p z_{t-p})^2}{2\sigma^2} \quad (11)$$

Likewise with AR(1), the normal equations for the OLS regression can be set up, and the variance is similar to AR(1) simply:

$$\hat{\sigma}^2 = \frac{1}{T-p} \sum_{t=p+1}^T (z_t - \hat{\mu} - \hat{\phi}_1 z_{t-1} - \hat{\phi}_2 z_{t-2} - \cdots - \hat{\phi}_p z_{t-p})^2 \quad (12)$$

MA(q)

Now consider the general MA(q) process:

$$z_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q} \quad (13)$$

Using a similar approach, assume the first q errors are 0; $\epsilon_0 = \epsilon_{-1} = \cdots = \epsilon_{-q+1} = 0$. This again allows the iteration of:

$$\epsilon_t = z_t - \mu - \theta_1 \epsilon_{t-1} - \theta_2 \epsilon_{t-2} - \cdots - \theta_q \epsilon_{t-q} \quad (14)$$

The conditional log likelihood function, much like before can be seen in equation (15), which also requires numerical optimization:

$$\log \mathcal{L}(\Theta) = -\frac{T}{2} \log(2\pi) - \frac{T}{2} \log(\sigma^2) - \sum_{t=1}^T \frac{\epsilon_t^2}{2\sigma^2} \quad (15)$$

Autoregressive Integrated Moving Average (ARIMA)

Developed in the 1970s, ARIMA models combine autoregressive, integration, and moving average components to capture trends, seasonality, and other patterns in time series data. ARIMA and its variants, such as ARIMAX (ARIMA with explanatory variables), have been widely used for univariate forecasting tasks and are particularly useful for modeling non-stationary time series data by applying differencing to induce stationarity.

Motivation for Integration term in AR/MA models

One of the assumptions for AR and MA models to work is that the model is stationary. Often, time series will not be stationary, however, differencing (i.e., *integrating*) the time series can be a tool to potentially introduce stationarity.

Differencing a time series means taking the difference between the time series of t and that of lag 1, i.e., $t-1$. Introducing the *backwards difference* operator ∇ , we can now define it with $k > 0$ as:

$$\nabla^1 z_t = z_t - z_{t-1} \quad (16)$$

$$\nabla^k z_t = \nabla^{k-1} z_t - \nabla^{k-1} z_{t-1} \quad (17)$$

Backwards differencing a time series once will then yield the following:

$$\nabla^1 z_t = z_t - z_{t-1}$$

adding yet another difference term would yield:

$$\begin{aligned}
\nabla^2 z_t &= \nabla^1 z_t - \nabla^1 z_{t-1} \\
&= (z_t - z_{t-1}) - (z_{t-1} - z_{t-2}) \\
&= z_t - 2z_{t-1} + z_{t-2}
\end{aligned}$$

This can also be written using the backshift operator giving us the differencing operator $\nabla^d = (1 - \mathbf{B})^d$:

$$\nabla^1 z_t = (1 - \mathbf{B}) z_t = z_t - z_{t-1}$$

and for another term:

$$\nabla^2 z_t = (1 - \mathbf{B})^2 z_t = z_t - 2z_{t-1} + z_{t-2}$$

By continuing to add difference terms, we can further transform the series until stationarity is achieved. Each level of differencing removes a trend or seasonality in the data, simplifying the underlying structure and making it suitable for AR and MA modeling. For instance, applying a second level of differencing as shown transforms the series by removing linear trends.

In general, if a time series requires d levels of differencing to become stationary, it is said to be integrated of order d , denoted as $I(d)$, indicating the number of differencing steps required for stationarity.

Recall that if the roots to the characteristic polynomial of the autoregressive equation $\phi(\mathbf{B}) = 1 - \sum_i^p \phi_i B^i = 0$, are greater than 1, the time series was considered stationary. In case one, or more, of the roots are not, the time-series is considered non-stationary. In such cases, differentiating can be a tool to introduce it.

ARIMA model

An ARIMA model of order (p, d, q) can be written as:

$$\nabla^d z_t = \phi_1 \nabla^d z_{t-1} + \cdots + \phi_p \nabla^d z_{t-p} + \epsilon_t - \theta_1 \epsilon_{t-1} - \cdots - \theta_q \epsilon_{t-q}$$

Where p is the order of the autoregressive part, d is the degree of differencing, q is the order of the moving average part and ∇^d is the differencing operator applied d times.

Using the backshift operator \mathbf{B} and the differencing operator ∇^d , the ARIMA model can be compactly represented as:

$$\phi(\mathbf{B}) \nabla^d z_t = \theta(\mathbf{B}) \epsilon_t$$

where $\phi(\mathbf{B})$ is the autoregressive part, $(1 - \mathbf{B})^d$ is the integration part, and $\theta(\mathbf{B}) \epsilon_t$ is the moving average part.

An ARIMA(0,0,0) is equivalent to $z_t = Z_t$ and can be considered white noise, ARIMA(0,1,0) is equivalent to $z_t = Z_t - Z_{t-1}$ and is a random walk

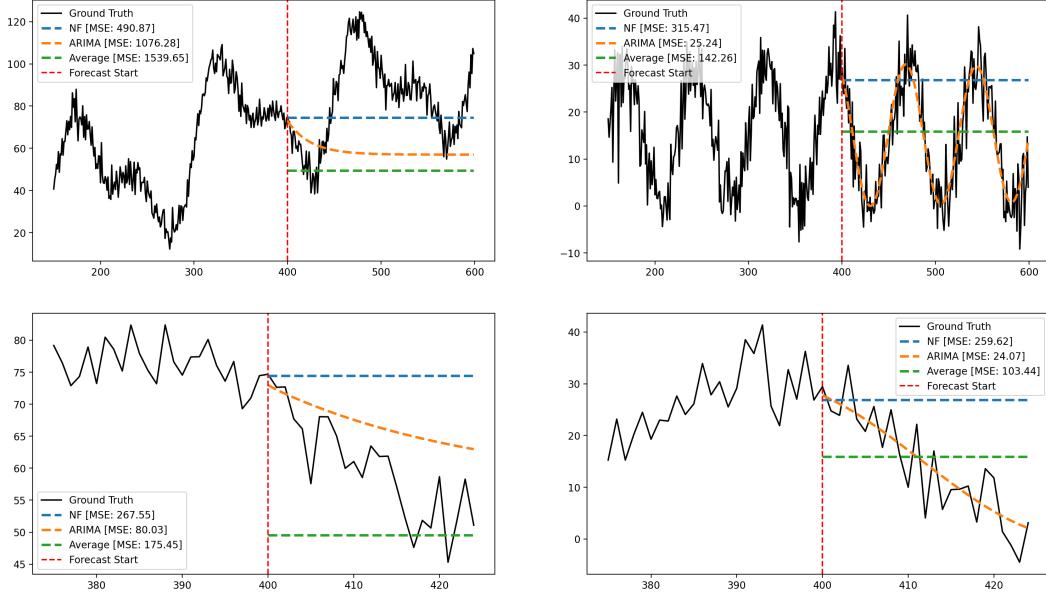


Figure 27: Examples of the baselines and ARIMA on two types of synthetic data.

Left: Synthetic data consists of 3 sinusoidal waves with normally distributed noise $\mathcal{N}(\mu = 15, \sigma^2 = 25)$ and a drift of $d = 0.15$. Here the Naïve forecast performs better on longer horizons, but ARIMA does best on short horizons.

Right: A simpler sinusoidal wave with noise $\mathcal{N}(\mu = 15, \sigma^2 = 25)$ and no drift. Here ARIMA almost perfectly captures the underlying signal, achieving superior performance, whereas the Naïve forecast does much worse.

Estimating parameters

Estimating the parameters of any of the previous models can be done using different methods including *Least Squares Estimation* (LSE), *Maximum Likelihood Estimation* (MLE), *Method of Moments* (MoM) etc.. We mention these, but we will only inspect the method *Maximum Likelihood Estimation* (MLE), more specifically the *conditional* version, and how it can be used to estimate the parameters $\hat{\mu}, \hat{\phi}_1, \hat{\theta}_1$ etc.

Conditional Maximum Likelihood Estimation

Maximum likelihood estimation is a method to estimate parameters in a probabilistic framework given observed data. The idéa is a two-step process: first is to define a likelihood function \mathcal{L} based on the assumed or determined distribution of the observed data and the second step is maximize it with regards to the parameters and determine the values. This is usually done by differencing with regards to the parameters and setting it to 0.

$$\frac{\partial \mathcal{L}}{\partial x} = 0, \quad x \in \{\mu, \sigma, \theta_i, \phi_j\}$$

For most time series data, the error is assumed to be i.i.d., following a gaussian distribution around a zero mean $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$.

The likelihood function can turn out to be computationally tough to do, due to non-linearity. However, if one models it conditioned on a full stochastic process. If one instead assumes some deterministic behaviour, it simplifies the calculations without sacrificing too much precision in results, under assumptions of large sample size and stationarity. This is the approach we explore here. [Ham94]

AR(1)

Recall the AR(1) model²:

²Recall we assume *stationarity*, as such the μ is not time dependent.

$$z_t = \mu + \phi_1 z_{t-1} + \epsilon_t \quad (18)$$

The process for a more exact maximum likelihood function can be found for an AR(1) process, however, it will require numerical/iterative algorithms to find a result.

Instead we here condition on the fact that the first observation, z_1 is *deterministic*. This simplification will make it possible to derive a cleaner expression. The likelihood function $\mathcal{L}(\Theta) = (\mu, \sigma, \phi_1)$ will be conditioned on the first observation:

$$\mathcal{L}(\Theta) = f_{z_T, z_{T-1}, \dots, z_2 | z_1}(z_T, z_{T-1}, \dots, z_2 | z_1; \Theta) = \prod_{t=2}^T f_{z_t | z_{t-1}}(z_t | z_{t-1}, \Theta) \quad (19)$$

We assumed the noise follows a Gaussian distribution, it is fair to assume the same for the time series observations.³ With the assumption that the first observation is deterministic, from equation (18), the expected value and variance is trivially seen to be $(c + \phi_1 \cdot z_{t-1}, \sigma^2)$, and we can now model the conditional distribution as $P(z_t | z_{t-1}) \sim \mathcal{N}((\mu + \phi_1 z_t), \sigma^2)$:

$$f_{z_t | z_{t-1}}(z_t | z_{t-1}, \Theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(z_t - \mu - \phi_1 z_{t-1})^2}{2\sigma^2}\right) \quad (20)$$

Our likelihood functions is thus:

$$\mathcal{L}(\Theta) = \prod_{t=2}^T \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(z_t - \mu - \phi_1 z_{t-1})^2}{2\sigma^2}\right) \quad (21)$$

It is customary to take the logarithm of ones likelihood function. As the logarithm functions is monotone and increasing, the values that would maximize a likelihood function would also maximize a log likelihood function. When using computer software, it is also provides more correct results due to floating point arithmetic, as a product of many value is instead a sum of the same values.

$$\log \mathcal{L}(\Theta) = \sum_{t=2}^T \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(z_t - \mu - \phi_1 z_{t-1})^2}{2\sigma^2}\right)\right) \quad (22)$$

$$= \sum_{t=2}^T \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \log\left(\exp\left(\frac{-(z_t - \mu - \phi_1 z_{t-1})^2}{2\sigma^2}\right)\right) \quad (23)$$

$$= \sum_{t=2}^T \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{(z_t - \mu - \phi_1 z_{t-1})^2}{2\sigma^2} \quad (24)$$

$$= \sum_{t=2}^T -\frac{\log(2\pi)}{2} - \frac{\log(\sigma^2)}{2} - \frac{(z_t - \mu - \phi_1 z_{t-1})^2}{2\sigma^2} \quad (25)$$

$$= -\frac{(T-1)}{2} \cdot \log(2\pi) - \frac{(T-1)}{2} \cdot \log(\sigma^2) - \sum_{t=2}^T \frac{(z_t - \mu - \phi_1 z_{t-1})^2}{2\sigma^2} \quad (26)$$

From equation (26) a sharp eye can see that a maximization of the log likelihood with regards to ϕ_1 and μ would simply be the minimization of $\sum_{t=2}^T (z_t - \mu - \phi_1 z_{t-1})^2$, which can be achieved using Ordinary Least Squares (OLS) Regression. After writing out the normal equations, the computation simplifies to the following. [Ham94]

$$\begin{bmatrix} \hat{\mu} \\ \hat{\phi}_1 \end{bmatrix} = \begin{bmatrix} T-1 & \sum z_{t-1} \\ \sum z_{t-1} & \sum z_{t-1}^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum z_t \\ \sum z_{t-1} z_t \end{bmatrix}$$

³The maximum likelihood for non-gaussian time series requires some other approaches and methods, but can in some cases be modelled as if gaussian. This is known as *quasi-maximum likelihood estimation*. This will not be explored in this report.

The conditional maximum likelihood estimate of the variance, $\hat{\sigma}^2$ is done by differentiating equation (26) w.r.t. σ^2 and setting it equal to 0.

$$\frac{\partial \log \mathcal{L}(\Theta)}{\partial \sigma^2} = -\frac{T-1}{2\sigma^2} + \sum_{t=2}^T \left(\frac{(z_t - \mu - \phi_1 z_{t-1})^2}{2\sigma^4} \right) = 0 \quad (27)$$

$$\hat{\sigma}^2 = \frac{1}{T-1} \sum_{t=2}^T (z_t - \hat{\mu} - \hat{\phi}_1 z_{t-1})^2 \quad (28)$$

MA(1)

Unfortunately where AR(1)'s conditional maximum likelihood equations could be solved, MA does not impose such elegance - the exact likelihood function is also not trivial. Nonetheless, we'll derive the conditional likelihood function, and then refer to Numerical Optimization algorithms to solve them.

Recall the MA(1) model.

$$z_t = \mu + \epsilon_t + \theta \epsilon_{t-1} \quad (29)$$

Assume that we know for certain ϵ_{t-1} and that $\epsilon_0 = 0$ ⁴, then the probability of the first observation is simply $(z_1|\epsilon_0) \sim \mathcal{N}(\mu, \sigma^2)$

Consequently, one now knows $\epsilon_1 = z_1 - \mu$ allowing one to form the probability density function for z_2 given z_1 :

$$\begin{aligned} f_{z_2|z_1, \epsilon_0=0}(z_2|z_1, \epsilon_0 = 0; \Theta) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(z_2 - \mu - \theta_1 \epsilon_1)^2}{2\sigma^2}\right) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-\epsilon_2^2}{2\sigma^2}\right) \end{aligned}$$

Similar can be written for the proceeding z_t 's! This allows us to form the conditional log-likelihood function, omitting some of the intermediary steps:

$$\log \mathcal{L}(\Theta) = \log f_{z_T, z_{T-1}, \dots, z_1|\epsilon_0=0}(z_T, z_{T-1}, \dots, z_1|\epsilon_0 = 0; \Theta) = -\frac{T}{2} \log(2\pi) - \frac{T}{2} \log(\sigma^2) - \sum_{t=1}^T \frac{\epsilon_t^2}{2\sigma^2} \quad (30)$$

The numerical values of Θ allows one to compute the sequence of ϵ 's from the data, for which the log likelihood is a function of the sum of squares of these. It is therefore then a complicated non-linear function, which in practice requires numerical optimizations to calculate.

ARMA(p, q)

Assume now the general ARMA model:

$$z_t = \mu + \phi_1 z_{t-1} + \phi_2 z_{t-2} + \dots + \phi_p z_{t-p} + \epsilon_t - \theta_1 \epsilon_{t-1} - \theta_2 \epsilon_{t-2} - \dots - \theta_q \epsilon_{t-q} \quad (31)$$

A common approximation of a likelihood function will require conditions on both z 's and ϵ 's, just like the individual AR and MA processes.

As such, assume the the values of $(z_0, z_{-1}, \dots, z_{-p+1})$ as given, likewise with $(\epsilon_0, \epsilon_{-1}, \dots, \epsilon_{-q+1})$. This will again allow the sequence of noise following $t = 0$ to be calculated:

$$\epsilon_t = z_t - \mu - \phi_1 z_{t-1} - \phi_2 z_{t-2} - \dots - \phi_p z_{t-p} - \theta_1 \epsilon_{t-1} - \theta_2 \epsilon_{t-2} - \dots - \theta_q \epsilon_{t-q} \quad (32)$$

And with that the conditional log likelihood function is as seen before, albeit a bit larger expression:

⁴This imposes a condition on θ and it is that $|\theta| < 1$. The reason is, if ϵ_0 is *not* 0, but we model as if it is, the error will consequently accumulate over time, and the result must be discarded, and the reciprocal value of θ should be investigated as a starting point instead.[Ham94]

$$\log \mathcal{L}(\Theta) = -\frac{T}{2} \log(2\pi) - \frac{T}{2} \log(\sigma^2) - \sum_{t=1}^T \frac{\epsilon_t^2}{2\sigma^2} \quad (33)$$

Like the the MA process, one must assume the values of the given noise and z 's. Commonly one assumes the ϵ takes on the value 0, and z 's take the observed values, and iteration in equation (32) starts from $t = p + 1$, and the noise:

$$\epsilon_p = \epsilon_{p-1} = \dots = \epsilon_{p-q+1} = 0$$

Which now allows the conditional likelihood to be of the form:

$$\log \mathcal{L}(\Theta) = -\frac{T-p}{2} \log(2\pi) - \frac{T-p}{2} \log(\sigma^2) - \sum_{t=p+1}^T \frac{\epsilon_t^2}{2\sigma^2} \quad (34)$$

MLE of ARIMA

When dealing with the parameters of ARIMA, one now deals with a differentiated time series, this has a direct impact on the Maximum Likelihood Estimation, and depends on the amount of differentiating. Nonetheless, the general approach is the same as in ARMA.