Article

Not peer-reviewed version

Surface Temperature Prediction with Bayesian Structral Time Series

Madhavan Balasubramaniam *

Posted Date: 17 March 2025

doi: 10.20944/preprints202503.1175.v1

Keywords: Bayesian Time Series Forecasting; Auto-Regressive Models; Frequentist vs Bayesian Forecasting; Seasonality and Trend Analysis; PyMC

Article

Surface Temperature Prediction with Bayesian Structural Time Series

Madhavan Balasubramaniam

mbalasub33@gatech.edu

Abstract: This article develops a Bayesian Auto-Regressive forecasting model for predicting global surface temperatures and compares its performance with a frequentist approach. Using the NASA GISS Surface Temperature dataset, we first implement an AR(4) model and then incorporate trend and seasonality components. Results show that the Bayesian approach improves generalization and provides probabilistic parameter estimates, making it more robust for long-term forecasting.

Keywords: bayesian time series forecasting; auto-regressive models; frequentist vs bayesian forecasting; seasonality and trend analysis; PyMC

Introduction

Time series forecasting models aim to predict future values of a sequence based on its historical data. Prominent examples include ARIMA (AutoRegressive Integrated Moving Average), ES (Exponential Smoothing), and VAR (Vector AutoRegression) models. One of the characteristic of Time series model is the dependency on the previous time stamp. This is called Auto Correlation.

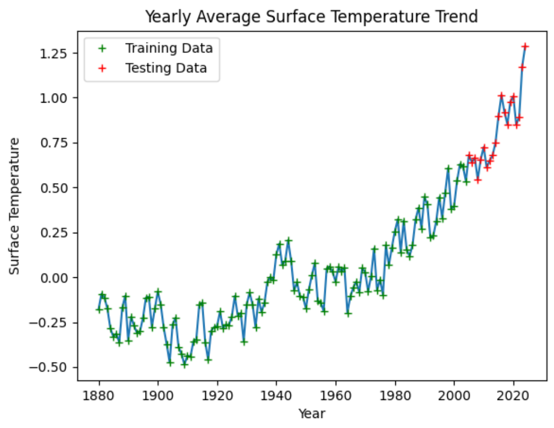
In this study, a Bayesian Auto-Regressive (AR) forecasting model is developed to predict future global surface temperature trends and is compared with its frequentist counterpart. The Bayesian AR model is first fitted to the data, followed by the incorporation of trend and seasonality components to enhance the accuracy of temperature predictions.

Literature Review

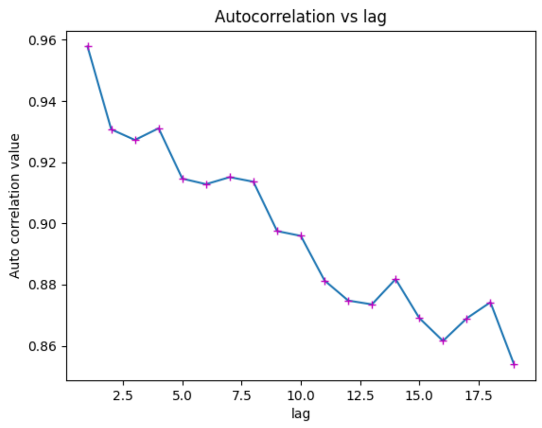
Bayesian Structural Time Series (BSTS) models have been widely applied across various industries for different forecasting and analytical applications. In [1], Brodersen et al. (2015) employed a BSTS state-space model to estimate the causal impact of market interventions, such as the effect of online advertising on search-related website visits. Their findings demonstrated that BSTS effectively quantifies causal impact in such scenarios. Poyser [2] utilized BSTS for forecasting Bitcoin's market price and analyzing influencing parameters through seasonality trend decomposition. This approach provided valuable insights into Bitcoin price fluctuations. Qiu et al. [3] extended BSTS to a multivariate setting (MSBTS) to improve inference and prediction for multiple correlated time series. Their study applied MSBTS to stock market price data and Google search trends, demonstrating superior forecasting performance compared to the ARIMAX model, particularly when strong correlations exist among the time series. Díaz Olariaga et al. [4] showcased the effectiveness of BSTS in forecasting air traffic cargo demand in Colombia, highlighting its capability to model complex trends and seasonality. Similarly, Hamza and Mohammad [5] applied BSTS to predict Turkish coal production, reporting that BSTS outperformed ARIMA in predictive accuracy. Given the demonstrated efficacy of BSTS across diverse fields, its application can be extended to surface temperature prediction. The ability of BSTS to incorporate seasonality and external predictors makes it well-suited for climate-related time series modeling.

Dataset

Goddard Institute for Space Studies Surface Temperature Analysis Dataset [7] provides global surface temperature estimates based on meteorological station data, sea surface temperature data, and polar temperature estimates. The surface temperatures values are available from 1880. The model is trained the yearly average values of Surface Temperature from 1880 to 2005 and the temperature values of the last 20 years is predicted.



Surface Temperature Time Series



AutoCorrelation vs Lag

The figures illustrate a clear upward trend in surface temperature values, indicating a long-term increase. Additionally, a multi-year cyclic pattern with a periodicity of approximately seven years is observed, highlighting the presence of seasonality. The autocorrelation values for the first four lags exceed 0.93, suggesting a strong temporal dependence in the data.

These observations justify the use of an Auto-Regressive (AR) model incorporating trend and seasonality components. Given the high autocorrelation in the first four lags, an AR(4) model is initially fitted to the data.

AR (4) Model

Frequentist Approach:

In AR(4) model, the current observation y_t depends linearly on the previous 4 observations.

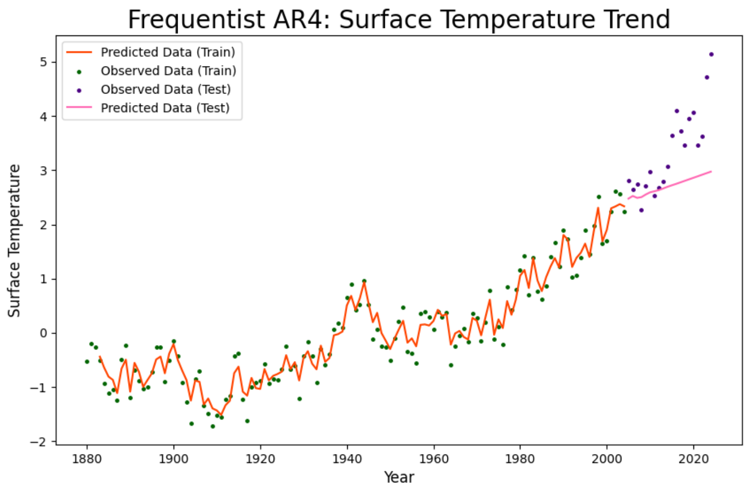
$$y_t = \rho_0 + \rho_1 y_{t-1} + \rho_2 y_{t-2} + \rho_3 y_{t-3} + \rho_4 y_{t-4} + \epsilon_t$$

where,

$$\begin{aligned} y_{t-1}, y_{t-2}, y_{t-3}, y_{t-4} &: \text{the previous 4 observations} \\ \rho_1, \rho_2, \rho_3, \rho_4 &: \text{auto-regressive coefficients} \\ \rho_0 &: \text{the constant term} \\ \epsilon_t &: \text{error term (white noise)} \end{aligned}$$

The dataset is first standardized to ensure comparability between the frequentist and Bayesian approaches. An Auto-Regressive (AR) model of order 4 is then implemented, with the lag parameter set to four. The model coefficients are estimated by optimizing the likelihood function, minimizing the sum of squared errors between the predicted values $\hat{y}_t$ and the observed values y_t. Once the model is fitted, it is used to generate future predictions based on the estimated parameters.

Frequentist AR(4) Model: Coefficients and Performance :



Frequentist AR(4) Model Predictions

Coefficient	Value
ρ_0	0.0446
ρ_1	0.6518
ρ_2	0.0059
ρ_3	0.1045
ρ_4	0.2413
$R^2_{\textbf{training}}$	0.9673
$R^2_{\textbf{testing}}$	-0.4265

Frequentist AR(4) Model Coefficients and Performance

The coefficient of determination (R^2) measures how well the independent variables in the model explains the variation in the dependent variable. A higher R^2 value indicates a better fit to the data. Although the Auto-Regressive (AR) model demonstrates a strong fit to the training dataset ($R^2_{\text{training}} = 0.9673$), it fails to generalize well to the testing data, resulting in a negative R^2 value ($R^2_{\text{testing}} = -0.4265$). This suggests that the model overfits the training data and is unable to capture the upward trend in the test dataset. This is evident from the model predictions illustrated above.

Bayesian Approach:

The Bayesian Auto Regressive Model of order 4 is described as follows:

$$\text{Prior on Coefficients:} \quad \rho_j \sim \mathcal{N}(\mu = 0, \sigma = 10), \quad j = 0, 1, 2, 3, 4,$$

$$\text{Prior on } \sigma_c\text{:} \quad \sigma_c \sim \mathcal{E}\text{xp}(\lambda = 1),$$

$$\text{Likelihood:} \quad y_t \mid \rho_j, y_{t-j} \sim \mathcal{N}\big(\mu = \rho_0 + \sum_{j=1}^{4} \rho_j y_{t-j},\, \sigma = \sigma_c\big).$$

Since prior information on the coefficients is not available, a non-informative prior is assigned. Specifically, a normal distribution with mean 0 and standard deviation 10 is used for the five coefficients, while an exponential distribution with a mean of 1 is assigned to σ. The exponential prior is weakly informative, favoring smaller values of σ.

Using Bayes' theorem, these priors are updated based on the observed time-series data to obtain the posterior distributions of the coefficients and σ_c. Unlike point estimates derived from the frequentist approach, these posterior distributions provide a comprehensive probabilistic characterization of the parameters.

For forecasting future values, samples are drawn from the posterior distributions of the coefficients and σ_c. These samples are then used to iteratively simulate future y_t values, thereby generating a posterior predictive distribution for the forecasted values.

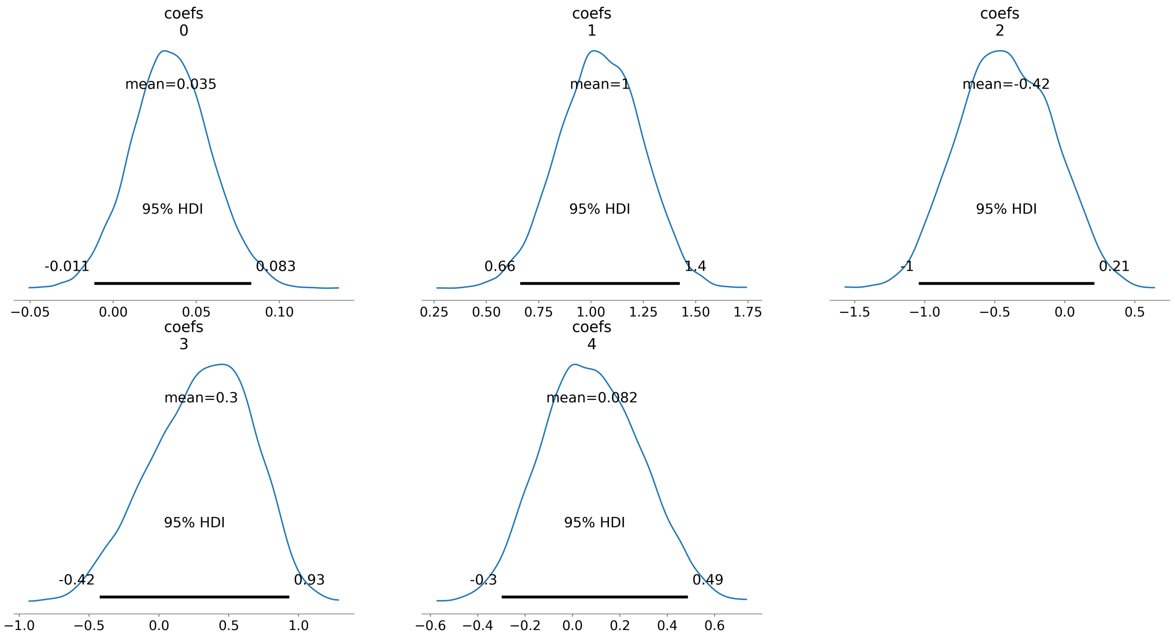
Posterior stats of coefficients:

Parameter	Mean	SD	HDI 2.5%	HDI 97.5%	$\hat{R}$
coefs[0]	0.035	0.024	-0.012	0.082	1.0
coefs[1]	1.035	0.195	0.657	1.413	1.0
coefs[2]	-0.392	0.339	-1.008	0.254	1.0
coefs[3]	0.268	0.376	-0.452	0.962	1.0
coefs[4]	0.100	0.212	-0.304	0.498	1.0
sigma	0.253	0.019	0.216	0.291	1.0

Forecasting with Bayesian AR4 model

During forecasting, the model predicts future values based on past observations. The number of steps for prediction must be specified. Since the autoregressive process of order 4 (AR(4)) requires an initial distribution, it is constrained to the last four observations of the training data. To enforce this constraint, a Dirac delta distribution of order 4 is employed, placing four point masses at the most recent observations. The posterior distribution of the coefficients is then utilized to derive the predictive distribution of future values.

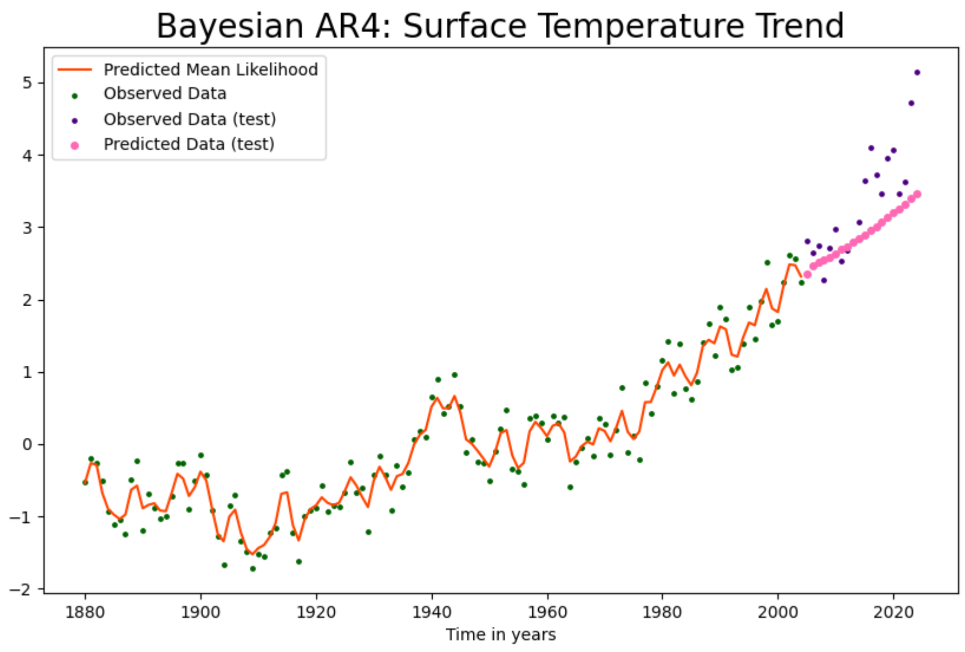
Posterior density plots of coefficients :



Posterior Density Plots (coefficients)

For the Bayesian AR(4) model, the coefficient of determination (R^2) is estimated as 0.88 for the training set and 0.171 for the testing set. In contrast, the frequentist approach yields R^2 values of 0.9673 and -0.4265 for training and testing, respectively. These results indicate that the Bayesian approach mitigates overfitting, providing more generalizable predictions. The predicted values are now compared with the actual observations from the testing dataset to evaluate the forecasting performance.

Forecasting with Bayesian AR4 model



The figure illustrates that the predicted values closely follow the observed values, demonstrating improved performance compared to the frequentist approach. However, the model fails to capture the steep trend present in the actual data. Additionally, the seasonal and cyclic variations are not well represented.

To address these limitations, seasonal and cyclic components are incorporated into the Bayesian AR(4) model to enhance its predictive accuracy.

AR (4) Model with trend and seasonality

Frequentist Approach :

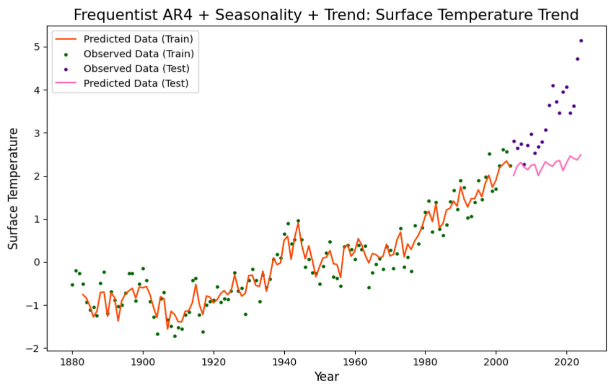
In AR(4) Trend Seasonality model, the current observation y_t depends linearly on the previous 4 observations, its sequence number t, and the seasonal terms. The seasonal terms consist of observations 'kp' steps before, where k ranges from 1 to m.

$$y_t = \rho_0 + \sum_{j=1}^{4} \rho_j y_{t-j} + \sum_{k=1}^{m} \phi_k y_{t-kp} + \alpha \cdot t + \epsilon_t$$

where,

$$\begin{aligned} \rho_1, \rho_2, \rho_3, \rho_4 &: \text{Auto regressive coefficients} \\ \phi_k &: \text{seasonality coefficients} \\ \alpha \cdot t &: \text{trend} \\ \text{p} &: \text{Period of seasonality} \end{aligned}$$

To model the seasonality and trend with AutoRegression (4), seasonality is incorporated by introducing dummy variables corresponding to each period within the seasonal cycle, while the trend is captured through a steady increase over time. By visual inspection, periodicity of the time series is estimated to be 7, corresponding to 7 dummy variables. The coefficients of autoregression, trend, and seasonal components are estimated by minimizing the sum of squared errors between the predicted values $\hat{y}_t$ and the observed values y_t, yielding maximum likelihood estimates. The fitted model can then be used to generate future predictions based on these estimated parameters.



AR4 frequentist model predictions

Coefficient	Value	Coefficient	Value
trend	0.0061	phi_4	-0.4152
phi_1	-0.6026	phi_5	-0.3742
phi_2	-0.2745	phi_6	-0.2802
phi_3	-0.2510	phi_7	0.3599
rho_1	0.5941	rho_2	-0.0004
rho_3	-0.0049	rho_4	0.2067
R^2_{training}	0.9452	R^2_{testing}	-1.9869

Frequentist AR(4) Model Coefficients and Performance

The model outperforms the AR(4) frequentist model, achieving a higher R^2 on the testing data and effectively capturing the cyclic patterns. However, it fails to forecast the upward trend in the testing data, as indicated by the very small trend coefficient of 0.0061. **Bayesian Approach:**
The Bayesian AR(4) model with trend and seasonality is described as follows.

$$\begin{aligned}
\text{Prior on AR coefficients:} \quad & \rho_j \sim \mathcal{N}(0.5, 0.5), \quad j = 0, 1, 2, 3, 4, \\
\text{Prior on slope coefficient:} \quad & \beta \sim \mathcal{N}(3, 0.5), \\
\text{Prior on intercept:} \quad & \alpha \sim \mathcal{N}(0.5, 0.5), \\
\text{Prior on } \sigma_c: \quad & \sigma_c \sim \text{Exp}(1), \\
\text{Prior on Fourier weights:} \quad & \text{fw} \sim \mathcal{N}(1, 0.5), \\
\text{Trend (deterministic):} \quad & \text{trend} = \alpha + \beta \cdot t, \\
\text{Seasonality (deterministic):} \quad & \text{seas} = \text{fw} \cdot \text{fourier\_features}, \\
\text{Likelihood:} \quad & y_t \mid \rho_j, y_{t-j} \sim \mathcal{N}\left(\rho_0 + \sum_{j=1}^{4} \rho_j y_{t-j} + \text{seas} + \text{trend},\ \sigma_c \right)
\end{aligned}$$

For effective MCMC sampling, when more degrees of freedom (such as trend and seasonality) are added, the Bayesian model requires a more informative prior. Since we know that the AR coefficients typically lie within [0,1], a prior of N(0.5,0.5) provides reasonable flexibility while discouraging values outside this range.

Moreover, as mentioned in [6], the Bayesian AR model with trend struggles to capture the trend line when using highly uninformative priors. Therefore, increasing the variability will never capture the directional pattern in the data. Hence, informative priors for the slope and intercept were chosen by iterating over multiple candidates using the method of prior predictive check.

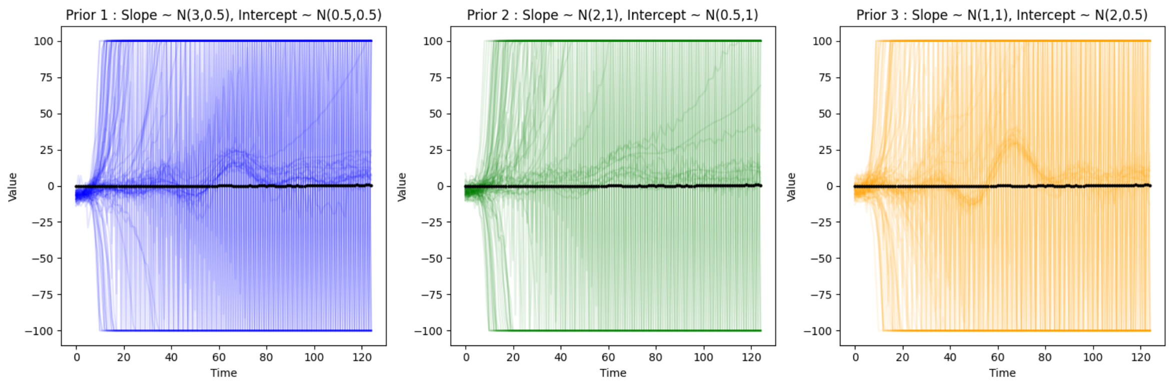
Trend: Bayesian AR(4) model exhibited a better upward trend compared to its frequentist counterpart. However, the trend eventually died out, and the predictions converged to the mean value when additional steps were forecasted. Adding a trend structure to the model would resolve this issue.

Seasonality: To account for seasonal cyclic patterns, synthetic Fourier features are generated and incorporated into the model. This approach represents the data as a weighted combination of sine and cosine functions. The cyclic components are treated as regular feature variables in the regression framework. Since these features are essential for capturing periodic trends, the model also requires Fourier features during the prediction phase

Prior Predictive check:
In a prior predictive check, the model computes the possible values of the observed variables (likelihood function). It converts the choices made in the parameter space to the observed values. Thinking in

terms of observations rather than coefficients makes the prior evaluation easier. Hence, the prior predictive check helps us choose reasonable priors.

Prior Predictive Check for AR4 + seasonality + trend model



Before performing prior predictive checks on various candidate priors for the slope, intercept, and Fourier coefficients, the remaining prior distributions were frozen.

Observations :

Most of the processes in three of the priors quickly diverge to large values. This is due to the recursive nature of the auto regressive MCMC sampling process.

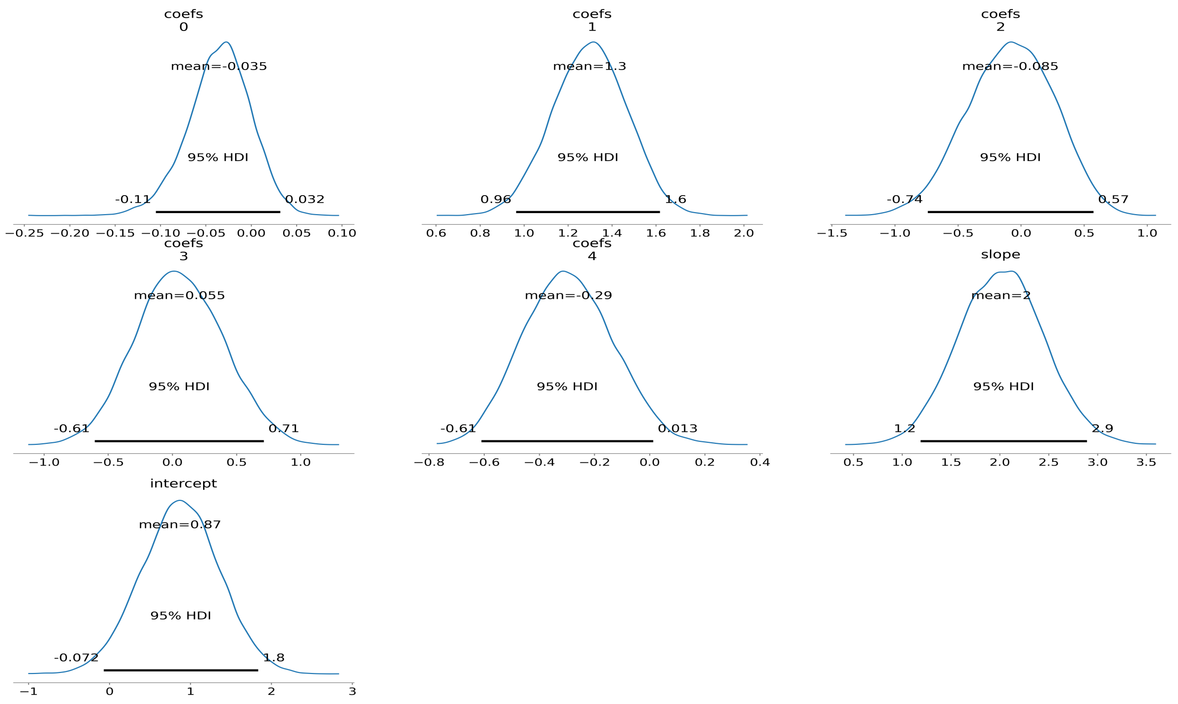
In the second prior, most samples quickly diverge to large values, and only weak cyclic patterns are visible. However, there is an upward trend, which is desirable.

The amplitude of the cyclic patterns are too large in the third prior. The first prior exhibits both a cyclic pattern and a weak upward trend. Therefore, it is a suitable candidate prior for Bayesian AR4 + seasonality + trend model.

Posterior stats of Bayesian AR(4)+ST model coefficients:

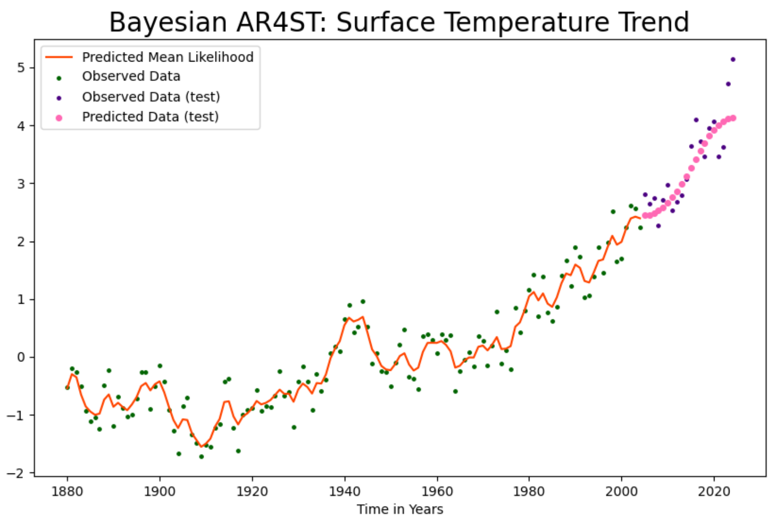
Parameter	Mean	SD	HDI 2.5%	HDI 97.5%	$\hat{R}$
coefs[0]	-0.035	0.035	-0.105	0.032	1.0
coefs[1]	1.299	0.169	0.964	1.618	1.0
coefs[2]	-0.085	0.343	-0.741	0.573	1.0
coefs[3]	0.055	0.339	-0.606	0.711	1.0
coefs[4]	-0.295	0.162	-0.611	0.013	1.0
sigma	0.289	0.022	0.248	0.332	1.0
slope	2.013	0.438	1.186	2.891	1.0
intercept	0.870	0.485	-0.072	1.834	1.0

Posterior density plots of coefficients, slope, trend:



R^2 score obtained on training data and testing data are 0.8634 and 0.7184, a significant improvement over the previous models.

Forecasting with Bayesian AR4 model with Seasonality and Trends:



We can observe that the model is able to capture both the seasonal variations and the trend in the data and able to forecast accurate values for the surface temperature

Model Comparison

Model	R^2_{train}	R^2_{test}
Frequentist AR(4)	0.9673	-0.4265
Bayesian AR(4)	0.8800	0.1710
Frequentist AR(4) + Seasonality + Trend	0.9452	-1.9870
Bayesian AR(4) + Seasonality + Trend	0.8634	0.7184

Conclusions

This study systematically analyzed the performance of Frequentist and Bayesian autoregressive models. Initially, under the assumption of uninformative priors, the Bayesian AR(4) model was expected to perform on par with its Frequentist counterpart. However, empirical results demonstrated that the Bayesian AR(4) model exhibited superior predictive accuracy.

Extending the analysis, the Frequentist AR(4)+ST model was employed to capture cyclic patterns. While this model effectively identified periodic variations, it failed to account for the underlying trend, highlighting a fundamental limitation in its formulation.

To address this, we incorporated weakly informative priors into the Bayesian AR(4)+ST model and conducted prior predictive checks to refine the selection of priors for slope, trend, and Fourier coefficients. This approach significantly enhanced the model's ability to capture both cyclical and trend components, leading to a marked improvement in predictive performance. The R^2 metric was used as a comparative measure, further validating the superiority of the Bayesian framework.

Beyond predictive accuracy, Bayesian AR models offer a distinct advantage by providing posterior distributions for both model coefficients and predicted variables, allowing for a more comprehensive uncertainty quantification. This probabilistic characterization enhances interpretability and robustness, making Bayesian methods a compelling choice for time series modeling in complex real-world scenarios.

Appendix A

Code Implementation Frequentist AR(4) Model

The `AutoReg` class from the `statsmodels` library is used to model AutoRegression (4) in Python with lags set to 4. The `.fit()` method estimates coefficients by maximizing the likelihood and minimizing the sum of squared errors. Future predictions are made using the `.predict()` method.

```python
import pandas as pd
from statsmodels.tsa.ar_model import AutoReg

gistemp_df = pd.read_csv("gisstemp.csv")
gistemp_df  = gistemp_df.apply(pd.to_numeric, errors='coerce')
gistemp_df['Yearly Average'] = gistemp_df.iloc[:, 1:13].mean(axis=1)
temp_trend = gistemp_df['Yearly Average'].values.reshape((-1,))
y_test = temp_trend[-20:]
y_train = temp_trend[:125]
y_train_std = (y_train - y_train.mean()) / y_train.std()
y_test_std = (y_test - y_train.mean()) / y_train.std()
ar_model = AutoReg(y_train_std, lags=4)
ar_results = ar_model.fit()
y_pred_std = ar_results.predict(start=1, end=145, dynamic=False)
```

Code Implementation Bayesian AR(4) Model

Modelling in PyMC:
To obtain the posterior estimates, the model is set up in PyMC using the `pm.AR()` module, which is defined as the mean of the likelihood function. Each lag term requires an initialization distribution, for which a normal distribution with a mean of 0 and a standard deviation of 10 is used.

```
import pymc as pm
priors = {
  "coefs": {"mu": [0, 0, 0, 0, 0], "sigma": [10, 10, 10, 10, 10], "size": 5},
  "lam": 1,
  "init": {"mu": 0, "sigma": 10, "size": 4}
with pm.Model(coords={"obs_id": y_train_std}) as AR4:
  yy = pm.Data(name="y", value=y_train_std, dims="obs_id")
  coefs = pm.Normal(name="coefs", mu=priors["coefs"]["mu"],
                    sigma=priors["coefs"]["sigma"])
  sigma = pm.Exponential(name="sigma", lam=priors["lam"])
  init = pm.Normal.dist(mu=priors["init"]["mu"], sigma=priors["init"]["sigma"],
                        size=priors["init"]["size"])
  ar4 = pm.AR(name="ar4",
              rho=coefs,
              sigma=sigma,
              init_dist=init,
              constant=True,
              steps=yy.shape[0] - (priors["coefs"]["size"] - 1),
              dims="obs_id")
  likelihood = pm.Normal("likelihood", mu=ar4, sigma=sigma, observed=yy, dims="obs_id")
  trace_ar4 = pm.sample(6000, tune=2000, nuts_sampler="numpyro", target_accept=0.95)
  posterior_predictive_ar4 = pm.sample_posterior_predictive(trace_ar4)
az.summary(trace_ar4, hdi_prob=0.95, var_names=["coefs", "sigma"])
```

Forecasting Step:
During forecasting, the model predicts future values based on trained past data, requiring a specified forecast horizon. The AR(4) process depends on an initial distribution constrained to the last four observations of y_{train}. A Dirac Delta distribution of order four, with point masses at these observations, is used to model this constraint.
Forecasting Step Python Code:

```
with AR4:
  AR4.add_coords({"obs_id_fut_4": range(len(y_train_std) - 4, prediction_length, 1)})
  AR4.add_coords({"obs_id_fut": range(len(y_train_std), prediction_length, 1)})
  init_dist = pm.DiracDelta.dist(ar4[..., -4:])  # Last 4 terms
  ar4_fut = pm.AR(
      "ar4_fut",
      init_dist=init_dist,
      rho=coefs,  # Coefficients: [rho_0, rho_1, rho_2]
      sigma=sigma,
      constant=True,
      dims="obs_id_fut_4",  # size of Lagged terms)
  yhat_fut = pm.Normal("yhat_fut", mu=ar4_fut[4:], sigma=sigma, dims="obs_id_fut")
  y_preds = pm.sample_posterior_predictive(
      trace_ar4, var_names=["likelihood", "yhat_fut"], predictions=True, random_seed=100)
az.summary(y_preds, var_names=["yhat_fut"], stat_funcs=None)['mean'].values
```

Code Implementation - Frequentist AR(4) Model with Trend and Seasonality

To model seasonality and trend with AutoRegression (4), the `seasonal` argument in the `AutoReg` class is set to true, `period` to 7, `trend` to 't' (time trend only), and lags to 4. Estimates for the autoregression, trend, and seasonality coefficients are obtained using the `.fit()` method, which employs an optimization algorithm to minimize the sum of squared errors between the predicted values $\hat{y}_t$ and the observed values y_t, producing maximum likelihood estimates (MLEs). Future predictions are generated using the `.predict()` method.

Python Implementation

```python
ar_model = AutoReg(y_train_std, lags=4, trend = 't', seasonal=True, period=7)
ar_results = ar_model.fit()
y_pred_std = ar_results.predict(start=1, end=145, dynamic=False)
def calculate_r_squared(y_true, y_pred):
    """Calculate R-squared between true and predicted values."""
    ss_total = ((y_true - y_true.mean()) ** 2).sum()
    ss_residual = ((y_true - y_pred) ** 2).sum()
    return 1 - (ss_residual / ss_total)

y_train_pred_std = y_pred_std[:len(y_train_std)]
y_test_pred_std = y_pred_std[-len(y_test_std):]
r_squared_train = calculate_r_squared(y_train_std[4:], y_train_pred_std [4:])
r_squared_test = calculate_r_squared(y_test_std, y_test_pred_std)
```

Code Implementation Bayesian AR(4) Model with Trend and Seasonality

Code to generate fourier features:

```python
n_order = 10
periods = np.array(year_train_std) / 7
fourier_features = pd.DataFrame(
    {f"{func}_order_{order}": getattr(np, func)(2 * np.pi * periods * order)
        for order in range(1, n_order + 1)
        for func in ("sin", "cos")
        })
```

Code for prior predictive check

```python
def ppc_AR_seasonal_model(y_train_std, y_test_std, fourier_features,
                          year_train_std, year_test_std, priors):
  fourier_features = fourier_features.to_numpy().T
  coords = {"obs_id": year_train_std,
            "fourier_features": np.arange(len(fourier_features))}

  with pm.Model(coords=coords) as AR4_plus_seasonal:
      yy = pm.Data(name="y", value=y_train_std, dims="obs_id")
      tt = pm.Data(name="t", value=year_train_std, dims="obs_id")
      fourier_terms = pm.Data(name="fourier_terms", value=fourier_features)
      coefs = pm.Normal(name="coefs", mu=priors["coefs"]["mu"],
                        sigma=priors["coefs"]["sigma"])
      sigma = pm.Exponential(name="sigma", lam=priors["lam"])
      init = pm.Normal.dist(mu=priors["init"]["mu"], sigma=priors["init"]["sigma"],
                          size=priors["init"]["size"])
      ar4 = pm.AR(name="ar4",
                  rho=coefs,
                  sigma=sigma,
                  init_dist=init,
                  constant=True,
                  steps=yy.shape[0] - (priors["coefs"]["size"] - 1),
                  dims="obs_id")

      slope = pm.Normal("slope", priors["slope"]["mu"], priors["slope"]["sigma"])
      intercept = pm.Normal("intercept", priors["intercept"]["mu"],
                                         priors["intercept"]["sigma"])
      trend = pm.Deterministic("trend", intercept + (slope * tt), dims="obs_id")

      fourier_weights = pm.Normal(
        "fourier_weights",
        mu=priors["fourier_weights"]["mu"],
        sigma=priors["fourier_weights"]["sigma"],
        dims="fourier_features")
      seasonality = pm.Deterministic(
            "seasonality", pm.math.dot(fourier_weights, fourier_terms), dims="obs_id")
      likelihood = pm.Normal("likelihood", mu=ar4+trend+seasonality, sigma=sigma,
                             observed=yy, dims="obs_id")
      prior_predictive = pm.sample_prior_predictive()
      return prior_predictive

n_order = 10
periods = np.array(year_train_std) / 7
fourier_features = pd.DataFrame(
    {
        f"{func}_order_{order}": getattr(np, func)(2 * np.pi * periods * order)
        for order in range(1, n_order + 1)
        for func in ("sin", "cos")
    }
)
priors_list = [
        {
        "coefs": {"mu": [0.5, 0.5, 0.5, 0.5, 0.5], "sigma": [0.5, 0.5, 0.5, 0.5, 0.5],
                                                   "size": 5},
        "lam": 1,
        "init": {"mu": 0, "sigma": 1, "size": 4},
        "slope": {"mu": 3, "sigma": 0.5},
        "intercept": {"mu": 0.5, "sigma": 0.5},
        "fourier_weights": {"mu": 1, "sigma": 0.5}
    },
         {
        "coefs": {"mu": [0.5, 0.5, 0.5, 0.5, 0.5], "sigma": [0.5, 0.5, 0.5, 0.5, 0.5],
                                                   "size": 5},
        "lam": 1,
        "init": {"mu": 0, "sigma": 1, "size": 4},
        "slope": {"mu": 2, "sigma": 1},
        "intercept": {"mu": 0.5, "sigma": 1},
        "fourier_weights": {"mu": 0.5, "sigma": 0.5}
    },
         {
        "coefs": {"mu": [0.5, 0.5, 0.5, 0.5, 0.5], "sigma": [0.5, 0.5, 0.5, 0.5, 0.5],
                                                   "size": 5},
        "lam": 1,
        "init": {"mu": 0, "sigma": 1, "size": 4},
        "slope": {"mu": 1, "sigma": 1},
        "intercept": {"mu": 2, "sigma": 0.5},
        "fourier_weights": {"mu": 2, "sigma": 1}
    },
]
```

Modelling in PyMC:
The priors for the slope, intercept, and Fourier weights are specified, followed by the computation of trend and seasonality as deterministic components. These components are integrated into the autoregressive process, with their sum serving as the mean of the likelihood function. Once the posterior distributions are estimated, forecasting is performed using Fourier features for the testing period.
Bayesian AR(4) Model with seasonality and trend:

```
year_info = np.array(list(range(1, len(temp_trend)+1)))
year_train = year_info[:125]
year_test = year_info[-20:]
year_train_std = (year_train - year_train.mean())/(year_train.std())
year_test_std = (year_test - year_train.mean())/(year_train.std())
priors = {"coefs": {"mu": [0.5, 0.5, 0.5, 0.5, 0.5], "sigma": [0.5, 0.5, 0.5, 0.5, 0.5],
          "size": 5},
    "lam": 1,
    "init": {"mu": 0, "sigma": 1, "size": 4},
    "slope": {"mu": 3, "sigma": 0.5},
    "intercept": {"mu": 0.5, "sigma": 0.5},
    "fourier_weights": {"mu": 1, "sigma": 0.5}}
fourier_features = fourier_features.to_numpy().T
coords = {"obs_id": year_train_std, "fourier_features": np.arange(len(fourier_features))}
with pm.Model(coords=coords) as AR4_plus_seasonal:
  yy = pm.Data(name="y", value=y_train_std, dims="obs_id")
  tt = pm.Data(name="t", value=year_train_std, dims="obs_id")
  fourier_terms = pm.Data(name="fourier_terms", value=fourier_features)
  coefs = pm.Normal(name="coefs", mu=priors["coefs"]["mu"],
                    sigma=priors["coefs"]["sigma"])
  sigma = pm.Exponential(name="sigma", lam=priors["lam"])
  init = pm.Normal.dist(mu=priors["init"]["mu"], sigma=priors["init"]["sigma"],
  size=priors["init"]["size"])
  ar4 = pm.AR(name="ar4",
            rho=coefs, sigma=sigma, init_dist=init, constant=True,
            steps=yy.shape[0] - (priors["coefs"]["size"] - 1),
            dims="obs_id")
  slope = pm.Normal("slope", priors["slope"]["mu"], priors["slope"]["sigma"])
  intercept = pm.Normal("intercept", priors["intercept"]["mu"], priors["intercept"]["sigma"])
  trend = pm.Deterministic("trend", intercept + (slope * tt), dims="obs_id")
  fourier_weights = pm.Normal(
    "fourier_weights",
    mu=priors["fourier_weights"]["mu"], sigma=priors["fourier_weights"]["sigma"],
    dims="fourier_features")
  seasonality = pm.Deterministic(
        "seasonality", pm.math.dot(fourier_weights, fourier_terms), dims="obs_id")
  likelihood = pm.Normal("likelihood", mu=ar4+trend+seasonality, sigma=sigma, observed=yy,
  dims="obs_id")
  trace= pm.sample(10000, tune=3000, nuts_sampler="numpyro", target_accept=0.99)
  posterior_predictive = pm.sample_posterior_predictive(trace)
```

Forecasting with Bayesian AR(4) Model with seasonality and trend:

```
prediction_length = len(y_train_std) + len(y_test_std)
order = 10
periods_fut = np.array(year_test_std) / 7
fourier_features_fut = pd.DataFrame(
  {
      f"{func}_order_{order}": getattr(np, func)(2 * np.pi * periods_fut * order)
      for order in range(1, n_order + 1)
      for func in ("sin", "cos")
  }
)
with AR4_plus_seasonal:
  AR4_plus_seasonal.add_coords({
      "obs_id_fut_4": range(len(y_train_std) - 4, prediction_length, 1)
  })
  AR4_plus_seasonal.add_coords({
      "obs_id_fut": range(len(y_train_std), prediction_length, 1)
  })
  t_fut = pm.Data(name="t_fut", value=year_test_std, dims="obs_id_fut")
  fourier_terms_fut = pm.Data("fourier_terms_fut", fourier_features_fut.to_numpy().T)
  ar4_fut = pm.AR(
      "ar4_fut",
      init_dist=pm.DiracDelta.dist(ar4[..., -4:]),
      rho=coefs,
      sigma=sigma,
      constant=True,
      dims="obs_id_fut_4"
  )
  trend_fut = pm.Deterministic("trend_fut", intercept + (slope * t_fut), dims="obs_id_fut")
  seasonality_fut = pm.Deterministic(
      "seasonality_fut", pm.math.dot(fourier_weights, fourier_terms_fut), dims="obs_id_fut"
  )
  yhat_fut = pm.Normal(
      "yhat_fut", mu=ar4_fut[4:] + trend_fut + seasonality_fut, sigma=sigma, dims="obs_id_fut"
  )
  y_preds = pm.sample_posterior_predictive(
      trace, var_names=["likelihood", "yhat_fut"], predictions=True, random_seed=100
  )
```

References

1. K. H. Brodersen, F. Gallusser, J. Koehler, N. Remy, and S. L. Scott, “Inferring causal impact using Bayesian structural time-series models,” *Annals of Applied Statistics*, vol. 9, no. 1, pp. 247-274, Mar. 2015. DOI: 10.1214/14-AOAS788.
2. O. Poyser, “Exploring the determinants of Bitcoin’s price: an application of Bayesian Structural Time Series,” 2017. DOI: 10.48550/arXiv.1706.01437.
3. J. Qiu, S. R. Jammalamadaka, and N. Ning, “Multivariate Bayesian structural time series model,” *Journal of Machine Learning Research*, vol. 19, no. 68, pp. 1-33, 2018.
4. Y. Rodríguez and O. Díaz Olariaga, “Air Traffic Demand Forecasting with a Bayesian Structural Time Series Approach,” *Periodica Polytechnica Transportation Engineering*, vol. 52, no. 1, pp. 75–85, 2024. DOI: 10.3311/PPtr.20973.
5. H. A. Hamza and M. J. Mohammad, “A Comparison of the Bayesian Structural Time Series Technique with the Autoregressive Integrated Moving Average Model for Forecasting,” *Migration Letters*, vol. 21, no. S4, pp. 528–538, 2024. DOI: 10.5281/zenodo.5654871.
6. N. Forde, “Forecasting with Structural AR Time Series,” in *PyMC Examples*, edited by PyMC Team. DOI: 10.5281/zenodo.5654871.

7. GISTEMP Team, "GISS Surface Temperature Analysis (GISTEMP), version 4," NASA Goddard Institute for Space Studies, Dataset accessed 2024-12-01. Available: https://data.giss.nasa.gov/gistemp/.
8. E. Herbst and C. Fonnesbeck, "Analysis of An Model in PyMC," in *PyMC Examples*, edited by PyMC Team. DOI: 10.5281/zenodo.5654871.