1 ARMA Models

Lab Objective: *ARMA(p,q) models combine autoregressive and moving-average models in order to forecast future observations using time-series. In this lab, we will build an ARMA(p,q) model to analyze and predict future weather data and then compare this model to statsmodels built-in ARMA package as well as the VARMAX package. Then we will forecast macroeconomic data as well as the future height of the Rio Negro.*

Time Series

A time series is any discrete-time stochastic process. In other words, it is a sequence of random variables, $\{Z_t\}_{t=1}^T$, that are determined by their time t. We let the realization of the time series $\{Z_t\}_{t=1}^T$ be denoted by $\{z_t\}_{t=1}^T$. Examples of time series include heart rate readings over time, pollution readings over time, stock prices at the closing of each day, and air temperature. Often when analyzing time series, we want to forecast future data, such as what will the stock price of a company will be in a week and what will the temperature be in 10 days.

ARMA(p,q) Models

One way to forecast a time series is using an ARMA model. The *Wold Theorem* says that any covariance-stationary time series can be well approximated with an ARMA model. An ARMA(p,q) model combines an autoregressive model of order p and a moving average model of order q on a time series $\{Z_t\}_{t=1}^T$. The model itself is a discrete-time stochastic process $(Z_t)_{t\in\mathbb{Z}}$ satisfying the equation

$$Z_t = \mathbf{c} + \underbrace{\left(\sum_{i=1}^{p} \Phi_i Z_{t-i}\right)}_{\text{AR(p)}} + \underbrace{\left(\sum_{j=1}^{q} \Theta_j \boldsymbol{\varepsilon_{t-j}}\right)}_{\text{MA(q)}} + \boldsymbol{\varepsilon_t} \tag{1.1}$$

where each $\boldsymbol{\varepsilon_t}$ is an identically-distributed Gaussian variable with mean 0 and constant covariance Σ, $\mathbf{c} \in \mathbb{R}^n$, and Φ_i and Θ_j are in $M_n(\mathbb{R})$.

AR(p) Models

An AR(p) model works similar to a weighted random walk. Recall that in a random walk, the current position depends on the immediate past position. In the autogregressive model, the current data point in the time series depends on the past p data points. However, the importance of each of the past p data points is not uniform. With an error term to represent white noise and a constant term to adjust the model along the y-axis, we can model the stochastic process with the following equation:

$$Z_t = \mathbf{c} + \sum_{i=1}^{p} \Phi_i Z_{t-i} + \boldsymbol{\varepsilon_t} \tag{1.2}$$

If there is a high correlation between the current and previous values of the time series, then the AR(p) model is a good representation of the data, and thus the ARMA(p, q) model will most likely be a good representation. The coefficients $\{\Phi_i\}_{i=1}^{p}$ are larger when the correlation is stronger.

In this lab, we will be using weather data from Provo, Utah[1]. To check that the data can be represented well, we need to look at the correlation between the current and previous values.

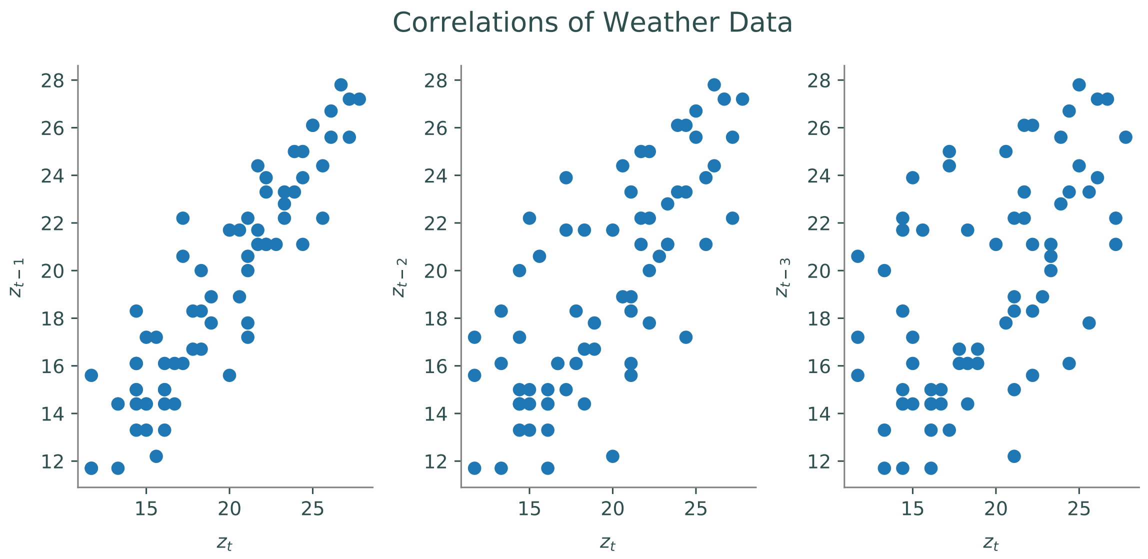

Figure 1.1: These graphs show that the weather data is correlated to its previous values. The correlation is weaker in each graph successively, showing that the further in the past the data is, the less correlated the data becomes.

MA(q) Models

A moving average model of order q is used to factor in the varying error of the time series. This model uses the error of the current data point and the previous data points to predict the next datapoint. Similar to an AR(p) model, this model uses a linear combination (which includes a constant term to adjust along the y-axis..

[1]This data was taken from https://forecast.weather.gov/data/obhistory/metric/KPVU.html

$$Z_t = \mathbf{c} + \boldsymbol{\varepsilon_t} + \sum_{i=1}^{q} \Theta_i \boldsymbol{\varepsilon_{t-i}} \tag{1.3}$$

This part of the model simulates shock effects in the time series. Examples of shock effects include volatility in the stock market or sudden cold fronts in the temperature.

Combining both the AR(p) and MA(q) models, we get an ARMA(p, q) model which forecasts based on previous observations and error trends in the data.

ARIMA(p, d, q) Models

Not all ARMA models are covariance stationary. However, many time series can be made covariance stationary by differencing. Let δZ_t represent the time series $z_t = Z_t - Z_{t-1}$ obtained by taking a difference of the terms. If the trend is linear a first difference is usually stationary. If the trend is quadratic a second difference may be necessary $\delta^2 Z_t = \delta(\delta Z_t)$. An ARIMA$(p, d, q)$ model is a discrete-time stochastic process $(Z_t)_{t\in\mathbb{Z}}$ satisfying the equation

$$\delta^d Z_t = \mathbf{c} + \underbrace{\left(\sum_{i=1}^{p} \Phi_i z_{t-i}\right)}_{\text{AR(p)}} + \underbrace{\left(\sum_{j=1}^{q} \Theta_j \boldsymbol{\varepsilon_{t-j}}\right)}_{\text{MA(q)}} + \boldsymbol{\varepsilon_t} \tag{1.4}$$

Finding Parameters

One of the most difficult parts of using an ARMA(p, q) model is identifying the proper parameters of the model. For simplicity, at the beginning of this lab we discuss univariate ARMA models with parameters $\{\phi_i\}_{i=1}^{p}$, $\{\theta_i\}_{i=1}^{q}$, μ, and σ, where μ and σ are the mean and standard deviation of the error. Note that $\{\phi_i\}_{i=1}^{p}$ and $\{\theta_i\}_{i=1}^{q}$ determine the order of the ARMA model. For this lab, we will let $\mathbf{c} = 0$.

A naive way to use an ARMA model is to choose p and q based on intuition. Figure 1.1 showed that there is a strong correlation between z_t and z_{t-1} and between z_t and z_{t-2}. The correlation is weaker between z_t and z_{t-3}. Intuition then suggests to choose $p = 2$. By looking at the correlations between the current noise with previous noise, similar to Figure 1.1, it can also be seen that there is a weak correlation between z_t and ε_t and between z_t and ε_{t-1}. Between z_t and ε_{t-2} there is no correlation. For more on how these error correlations were found, see Additional Materials. Intuition from these correlations suggests to choose $q = 1$. Thus, a naive choice for our model is an ARMA$(2, 1)$ model.

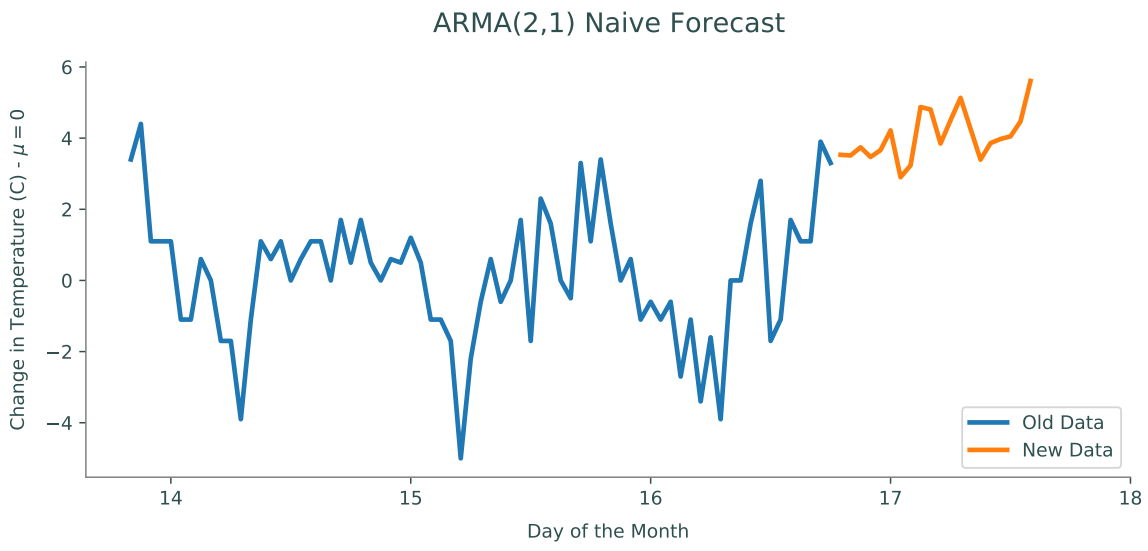

Figure 1.2: Naive forecast on `weather.npy`

Problem 1. Write a function `arma_forecast_naive()` that builds an ARMA(p,q) model. Your function should accept as parameters p, q, and n, where p is the order of the autoregressive model, q is the order of the moving average model, and n is the number of observations to predict. Assume `c`$= 0$, and let $\phi_i = .5$, $\theta_i = .1$, and $\varepsilon_i \sim \mathcal{N}(0, 1)$ for all i.

Use your function to predict the next n values of the time series found in `weather.npy`. This file contains data on the temperature in Provo, Utah from 7:56 PM May 13, 2019 to 6:56 PM May 16, 2019, taken every hour. This time series is NOT covariance stationary, so to make it covariance stationary, take its first difference (Hint: you might find `np.diff()` helpful). We denote the new covariance stationary time series as $\{z_t\}_{t=1}^T$. Predict the next n observations by iterating through equation 1.4.

Run your code on `weather.npy`, and plot the observed differences $\{z_t\}_{t=1}^T$ followed by your predicted observations of z_t. For $p = 2$, $q = 1$, and $n = 20$, your plot should look similar to Figure 1.2, however, due to the variance of the error ε_t, the plot will not look exactly like Figure 1.2. The predictions may be higher or lower on the y-axis.

Let $\Theta = \{\phi_i, \theta_j, \mu, \sigma\}$ be the set of parameters for an ARMA(p, q) model. Suppose we have a set of observations $\{z_t\}_{t=1}^n$. Our goal is to find the p, q, and Θ that maximize the likelihood of the ARMA model given the data. Using the chain rule, we can factorize the likelihood of the model given this data as

$$p(\{z_t\}|\Theta) = \prod_{t=1}^{n} p(z_t | z_{t-1}, \ldots, z_1, \Theta) \tag{1.5}$$

State Space Representation

In a general ARMA(p, q) model, the likelihood is a function of the unobserved error terms ε_t and is not trivial to compute. Simple approximations can be made, but these may be inaccurate under certain circumstances. Explicit derivations of the likelihood are possible, but tedious. However, when the ARMA model is placed in state-space, the Kalman filter affords a straightforward, recursive way to compute the likelihood.

We demonstrate one possible state-space representation of an ARMA(p, q) model. Let $r = \max(p, q + 1)$. Define

$$\hat{\mathbf{x}}_{t|t-1} = \begin{bmatrix} x_{t-1} & x_{t-2} & \cdots & x_{t-r} \end{bmatrix}^T \tag{1.6}$$

$$F = \begin{bmatrix} \phi_1 & \phi_2 & \cdots & \phi_{r-1} & \phi_r \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \tag{1.7}$$

$$H = \begin{bmatrix} 1 & \theta_1 & \theta_2 & \cdots & \theta_{r-1} \end{bmatrix} \tag{1.8}$$

$$Q = \begin{bmatrix} \sigma & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ldots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{1.9}$$

$$w_t \sim \text{MVN}(0, Q), \tag{1.10}$$

where $\phi_i = 0$ for $i > p$, and $\theta_j = 0$ for $j > q$. Note that Equation 1.2 gives

$$F\hat{\mathbf{x}}_{t-1|t-2} + w_t = \begin{bmatrix} \sum_{i=1}^{r} \phi_i x_{t-i} \\ x_{t-1} \\ x_{t-2} \\ \vdots \\ x_{t-(r-1)} \end{bmatrix} + \begin{bmatrix} \varepsilon_t \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{1.11}$$

$$= \begin{bmatrix} x_t & x_{t-1} & \cdots & x_{t-(r-1)} \end{bmatrix}^T \tag{1.12}$$

$$= \hat{\mathbf{x}}_{t|t-1} \tag{1.13}$$

We note that $z_{t|t-1} = H\hat{\mathbf{x}}_{t|t-1} + \mu$.[2]

Then the linear stochastic dynamical system

$$\hat{\mathbf{x}}_{t+1|t} = F\hat{\mathbf{x}}_{t|t-1} + w_t \tag{1.14}$$

$$z_{t|t-1} = H\hat{\mathbf{x}}_{t|t-1} + \mu \tag{1.15}$$

describes the same process as the original ARMA model.

Note

Equation 1.15 involves a deterministic component, namely μ. The Kalman filter theory developed in the previous lab, however, assumed $\mathbb{E}[\varepsilon_t] = 0$ for the observations $z_{t|t-1}$,. This means you should subtract off the mean μ of the error from the time series observations $z_{t|t-1}$ when using them in the predict and update steps.

[2]For a proof of this fact, see Additional Materials.

Likelihood via Kalman Filter

We assumed in Equation 1.10 that the error terms of the model are Gaussian. This means that each conditional distribution in 1.5 is also Gaussian, and is completely characterized by its mean and variance:

$$\text{mean} \quad H\hat{\mathbf{x}}_{t|t-1} + \mu \tag{1.16}$$

$$\text{variance} \quad HP_{t|t-1}H^T \tag{1.17}$$

where $\hat{\mathbf{x}}_{t|t-1}$ and $P_{t|t-1}$ are easily found via the Kalman filter, during the Predict step. Given that each conditional distribution is Gaussian, the likelihood can then be found as

$$p(\{z_t\}|\Theta) = \prod_{t=1}^{n} N(z_t \mid H\hat{\mathbf{x}}_{t|t-1} + \mu, HP_{t|t-1}H^T). \tag{1.18}$$

Problem 2. Write a function `arma_likelihood()` that returns the log-likelihood of an ARMA model, given a time series $\{z_t\}_{t=1}^{T}$. This function should accept `filename` which contains the observations, and it should accept as parameters each parameter in Θ. In this case, the time series should be the change in temperature of `weather.npy`, which is the first difference of the time series found in `weather.npy`, as was done in Problem 1. Adapt Equation 1.18 to calculate and return the log-likelihood of the ARMA(p, q) model as a **float**.

Use the provided `state_space_rep()` function to generate F, Q, and H. The function `kalman()` has also been provided to help calculate the means and covariances of each observation. Calling the function `kalman()` on a time series will return an array whose values are $\hat{\mathbf{x}}_{t|t-1}$ and an array whose values are $P_{t|t-1}$ for each $t \leq n$.

Hint: remember to subtract off the mean μ from the inputted observation when using `kalman()`.

When implemented correctly, your function should match the following output:

```
>>> arma_likelihood(filename='weather.npy', phis=np.array([0.9]),
                    thetas=np.array([0]), mu=17., std=0.4)
-1375.1805469978776
```

Model Identification

Now that we can compute the likelihood of a given ARMA model, we want to find the best choice of parameters given our time series. In this lab, we define the model with the "best" choice of parameters as the model which minimizes the AIC. The benefit of minimizing the AIC is that it rewards goodness of fit while penalizing overfitting. The AIC is expressed by

$$2k\left(1 + \frac{k+1}{n-k}\right) - 2\ell(\Theta) \tag{1.19}$$

where n is the sample size, $k = p + q + 2$ is the number of parameters in the model, and $\ell(\Theta)$ is the maximum likelihood for the model class.

To compute the maximum likelihood for a model class, we need to optimize 1.18 over the space of parameters Θ. We can do so by using an optimization routine such as `scipy.optimize.minimize` on the function `arma_likelihood()` from Problem 2. Use the following code to run this routine.

```
>>> from scipy.optimize import minimize

>>> # assume p, q, and time_series are defined
>>> def f(x): # x contains the phis, thetas, mu, and std
>>>     return -1*arma_likelihood(filename, phis=x[:p], thetas=x[p:p+q],
                                    mu=x[-2], std=x[-1])
>>> # create initial point
>>> x0 = np.zeros(p + q + 2)
>>> x0[-2] = time_series.mean()
>>> x0[-1] = time_series.std()
>>> sol = minimize(f,  x0, method = "SLSQP")
>>> sol = sol['x']
```

This routine will return a vector `sol` where the first p values are $\{\phi_i\}_{i=1}^{p}$, the next q values are $\{\theta_i\}_{i=1}^{q}$, and the last two values are μ and σ, respectively. Note the wrapper `f(x)` returns the **negative** log-likelihood. This is because `scipy.optimize.minimize` finds the *minimizer* of $f(x)$ and we are solving for the *maximum* likelihood.

To minimize the AIC, we perform *model identification*. This is choosing the order of our model, p and q, from some admissible set. The order of the model which minimizes the AIC is then the optimal model.

Problem 3. Write a function `model_identification()` that accepts `filename` containing the time series data and parameters `p_max` and `q_max` as integers. Determine which ARMA(i, j) model has the minimum AIC for all $1 \leq i \leq$ `p_max` and $1 \leq j \leq$ `q_max`. Then, return each parameter in Θ of that model.

Hint: when calculating the AIC using Equation 1.19, bear in mind that $-\ell(\Theta) =$ `f(sol)` where `sol` is found in the code above and explained in the following paragraph.

Your code should replicate the following output up to at least 4 decimal places.

```
>>> model_identification(filename='weather.npy', p_max=4, q_max=4)
(array([ 0.7213538]), array([-0.26246426]), 0.359785001944352, ↩
    1.5568374351425505)
```

Forecasting with Kalman Filter

We have now identified the optimal ARMA(p, q) model. We can use this model to predict future states. The Kalman filter provides a straightforward way to predict future states by giving the mean and variance of the conditional distribution of future observations. Observations can be found as follows

$$z_{t+k}|z_1, \cdots, z_t \sim N(z_{t+k};\ H\hat{x}_{t+k|t} + \mu,\ HP_{t+k|t}H^T) \tag{1.20}$$

To evolve the Kalman filter, recall the predict and update rules of a Kalman filter.

$$\begin{aligned}
\textbf{Predict} \qquad & \widehat{\mathbf{x}}_{k|k-1} = F\widehat{\mathbf{x}}_{k-1|k-1} + \mathbf{u} \\
& P_{k|k-1} = FP_{k-1|k-1}F^T + Q \\
\textbf{Update} \qquad & \tilde{\mathbf{y}}_k = \mathbf{z}_k - H\widehat{\mathbf{x}}_{k|k-1} \\
& S_k = HP_{k|k-1}H^T + R \\
& K_k = P_{k|k-1}H^T S_k^{-1} \\
& \widehat{\mathbf{x}}_{k|k} = \widehat{\mathbf{x}}_{k|k-1} + K_k\tilde{\mathbf{y}}_k \\
& P_{k|k} = (I - K_kH)P_{k|k-1}
\end{aligned}$$

With ARMA, we define observational noise covariance R and drift term $\mathbf{u}$ to both be 0.

Achtung!

Recall that the values returned by `kalman()` are conditional on the previous observation. To compute the mean and variance of future observations, the values $x_{n|n}$ and $P_{n|n}$ MUST be computed using the Update step. Once they are computed, only the Predict step is needed to find the future means and covariances.

Problem 4. Write a function `arma_forecast()` that accepts `filename` containing a time series, the parameters for an ARMA model, and the number n of observations to forecast. Calculate the mean and covariance of the future n observations using the Kalman filter.

To do this, use `state_space_rep()` to generate F, Q, and H. Then, use `kalman()` (with μ subtracted off from the covariance stationary time series $\mathbf{z}_k$) to calculate $\hat{\mathbf{x}}_{k|k-1}$ and $P_{k|k-1}$, respectively. Use the Update step on the last elements of $\mathbf{z}_k$ (with μ subtracted off), $\hat{\mathbf{x}}_{k|k-1}$, and $P_{k|k-1}$ to find $\hat{\mathbf{x}}_{k|k}$ and $P_{k|k}$. Then, iteratively use the Predict step to make future predictions of the mean and covariance. Remember that once you find a mean $\hat{\mathbf{x}}_{k|k-1}$ and covariance $P_{k|k-1}$, you must use equations 1.16 and 1.17 to transform them back into observation space.

Plot the original observations as well as the mean of each future observation. Plot a 95% confidence interval (2 standard deviations away from the mean) around the means of future observations. Hint: the standard deviation is the square root of the covariance calculated.

The following code should create a plot similar to Figure 1.3.

```
>>> # Get optimal model as found in the previous problem
>>> phis, thetas, mu, std = np.array([0.72135856]), np.array↩
    ([-0.26246788]), 0.35980339870105321, 1.5568331253098422

>>> # Forecast optimal mode
>>> arma_forecast(filename='weather.npy', phis=phis, thetas=thetas,
                  mu=mu, std=std, n=30)
```

How does this graph compare to the naive ARMA graph from Problem 1?

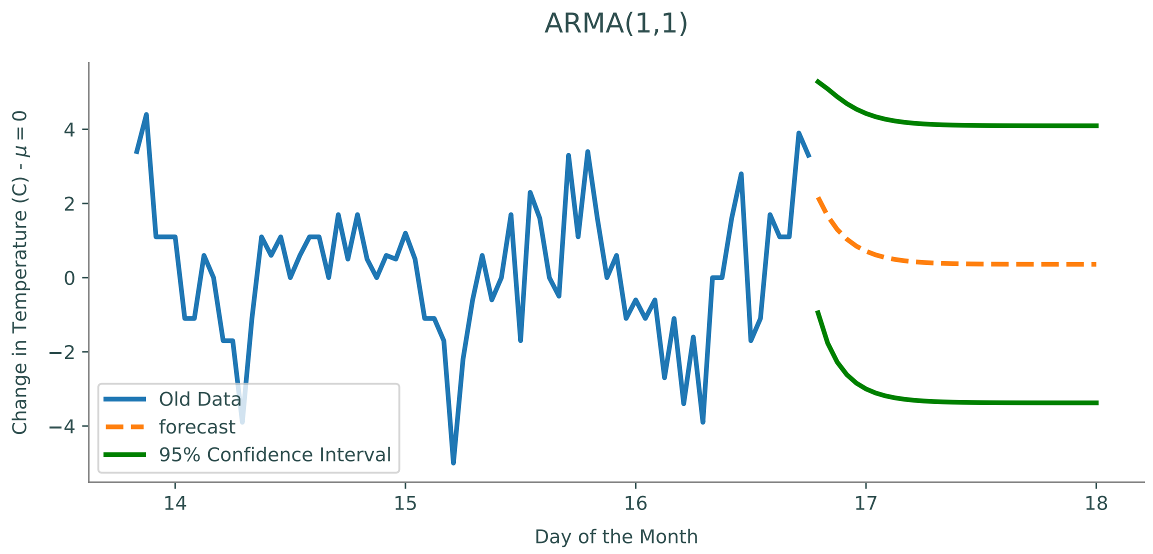

Figure 1.3: ARMA(1,1) forecast on `weather.npy`

Statsmodel ARMA

The module `statsmodels` contains a package that includes an ARMA model class. This is accessed through ARIMA model, which stands for Autoregressive Integrated Moving Average. This class also uses a Kalman Filter to calculate the MLE. When creating an ARIMA object, initialize the variables `endog` (the data) and `order` (the order of the model). The order is of the form (p, d, q) where d is the differences. To create an ARMA model, set $d = 0$. The object can then be fitted based on the MLE using a Kalman Filter.

```
from statsmodels.tsa.arima.model import ARIMA
# Intialize the object with weather data and order (1,1)
>>> model = ARIMA(z,order=(p,0,q),trend='c').fit(method='innovations_mle')

# Access p and q
>>> model.specification.k_ar
p
>>> model.specification.k_ma
q
```

As in the other problems, the time series passed in should be covariance stationary. The AIC of an ARMA model object is saved as the attribute `aic`. Since the AIC is much faster to compute using `statsmodels`, model identification is much faster. Once a model is chosen, the method `predict` will forecast n observations, where n is the number of known observations. It will return the mean of each future observation.

```
# Predict from the beginning of the model to 30 observations in the future
>>> model.predict(start=0,end=len(data)+30)
```

Problem 5. Write a function `sm_arma()` that accepts `filename` containing a time series, integer values for `p_max` and `q_max`, and the number n of values to predict.

As in Problem 3, perform model identification to find the ARMA(i, j) model with the best AIC for $1 \leq i \leq$ `p_max` and $1 \leq j \leq$ `q_max`, but this time use `statsmodels`. Make sure the model is fit using the MLE.

Use the optimal model to predict n future observations of the time series. Plot the original observations along with the predicted observations from the beginning through n observations in the future, as given by `statsmodels`. Return the AIC of the optimal model.

For `p_max` $= 3,$ `q_max` $= 3$, and $n = 30$, your graph should look similar to Figure 1.4. How does this graph compare to Problem 1? Problem 4?

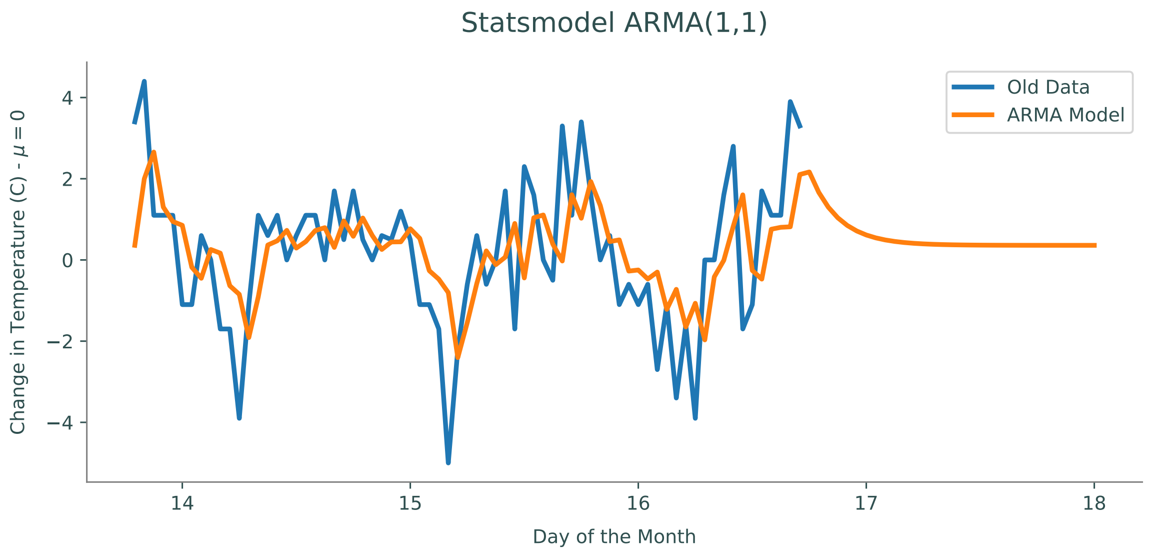

Figure 1.4: Statsmodel forecast on `weather.npy`.

Statsmodel VARMA

Until now we have been dealing with univariate ARMA models. Multivariate ARMA models are used when we have multiple time series that can be useful in predicting one another. For example say we have two time series $z_{t,1}$ and $z_{t,2}$. The multivariate ARMA(1,1) model is as follows:

$$z_{t,1} = c_1 + \phi_{11} z_{t-1,1} + \phi_{12} z_{t-1,2} + \theta_{11} \varepsilon_{t-1,1} + \theta_{12} \varepsilon_{t-1,2} \tag{1.21}$$

$$z_{t,2} = c_1 + \phi_{21} z_{t-1,1} + \phi_{22} z_{t-1,2} + \theta_{21} \varepsilon_{t-1,1} + \theta_{22} \varepsilon_{t-1,2} \tag{1.22}$$

This can be written in matrix form as shown in Equation 1.1. The module `statsmodels` contains a package that includes a VARMAX model class which can be used to create a multivariate ARMA model. VARMAX stands for Vector Autoregression Moving Average with Exogenous Regressors. An exogenous regressor is a time series that affects the model but is not affected by it. In the example below we have two time series corresponding to the price of copper and aluminum. Since aluminum is a substitute for copper, it is reasonable to assume the price of aluminum may help us predict the price of copper and vice versa. Note that when fitting a VARMAX model, setting the parameter `ic='aic'` selects parameters based on AIC criterion.

```
>>>from statsmodels.tsa.api import VARMAX
>>>import statsmodels.api as sm

>>> # Load in world copper data
>>> data = sm.datasets.copper.load_pandas().data
>>> # Create index compatible with VARMAX model
>>> idx = pd.period_range(start='1951', end='1975',freq = 'Y')
>>> data.index = idx

>>> # Initialize and fit model
>>> mod = VARMAX(data[['ALUMPRICE', 'COPPERPRICE']])
>>> mod = mod.fit(maxiter=1000, disp=False, ic='aic')
>>> # Predict until the price of aluminium and copper until 1985
>>> pred = mod.predict('1951','1985')

>>> # Get confidence intervals
>>> forecast_obj = mod.get_forecast('1981')
>>> all_CI = forecast_obj.conf_int(alpha=0.05)
>>> all_CI

>>> # Plot predictions against true price
>>> pred.plot()
>>> plt.plot(data['ALUMPRICE'],'r--', label = 'ALUMPRICE prediction')
>>> plt.plot(data['COPPERPRICE'],'r--',label = 'COPPERPRICE prediction')
>>> plt.legend()
>>> plt.title('VARMA Predictions for World Copper Market Dataset')
```

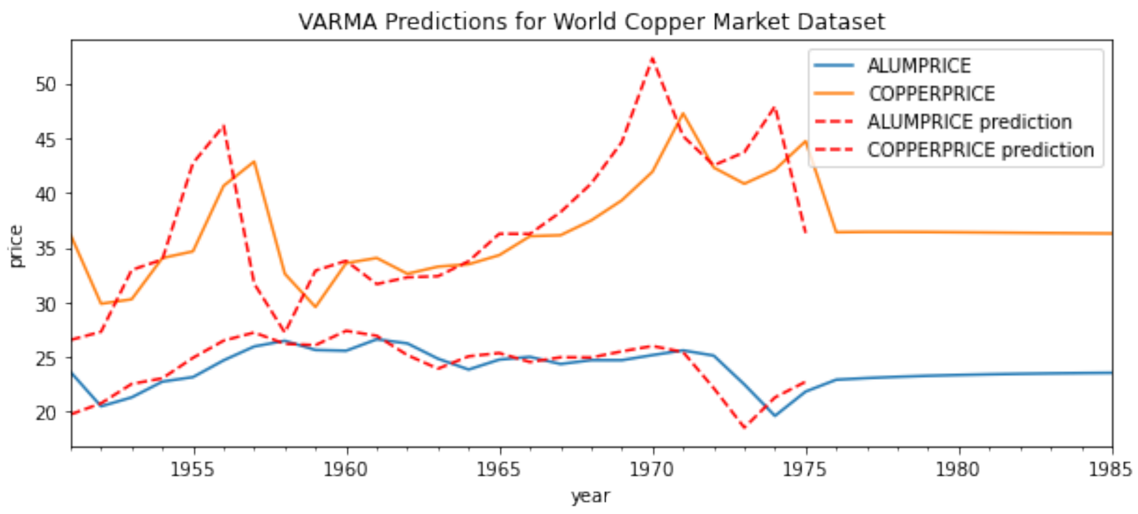

Figure 1.5: Statsmodel VAR(1) forecast.

Problem 6. Write a function `sm_varma()` that accepts start and end dates for forecasting. Use the statsmodels VARMAX class to forecast on macroeconomic data between the start and end dates.

The following code shows how to obtain the data.

```
>>> # Load in data
>>> df = sm.datasets.macrodata.load_pandas().data
>>> # Create DatetimeIndex
>>> dates = df[['year', 'quarter']].astype(int).astype(str)
>>> dates = dates["year"] + "Q" + dates["quarter"]
>>> dates = dates_from_str(dates)
>>> df.index = pd.DatetimeIndex(dates)
>>> # Select columns used in prediction
>>> df = df[['realgdp','realcons','realinv']]
```

Initialize your VARMAX model with the `df` specified above, and include the parameter `freq="Q-DEC"`. Fit your model, using AIC as the criteria for model selection, and predict from the start date until the end date. Then, get the model forecast until the end date. Plot the original data, prediction, and a 95% confidence interval (2 standard deviations away from the mean) around the future observations. Return the AIC of the chosen model.

The plot should be similar to Figure 1.6.

Hint: in the example above, `mod.predict('1951','1985')` returns a dataframe of 2 columns that contain the predicted values of `'ALUMPRICE'` and `'COPPERPRICE'`, respectively, from the years 1951 to 1985. Also, `all_CI` is a dataframe where each column indicates the corresponding dataset and whether it is a lower or upper bound of a confidence interval determined by the alpha value. Thus, the column `'lower ALUMPRICE'` with `alpha=0.05` contains the lower bounds of a 95% confidence interval for the `'ALUMPRICE'` dataset.

The dataset `'realgdp'` contains the real gross domestic product, `'realcons'` contains real personal consumption expenditures, and `'realinv'` contains real gross private domestic investment. Since personal consumption and domestic investment are components of gross domestic product, it is reasonable to assume these time series will be useful in predicting one another.

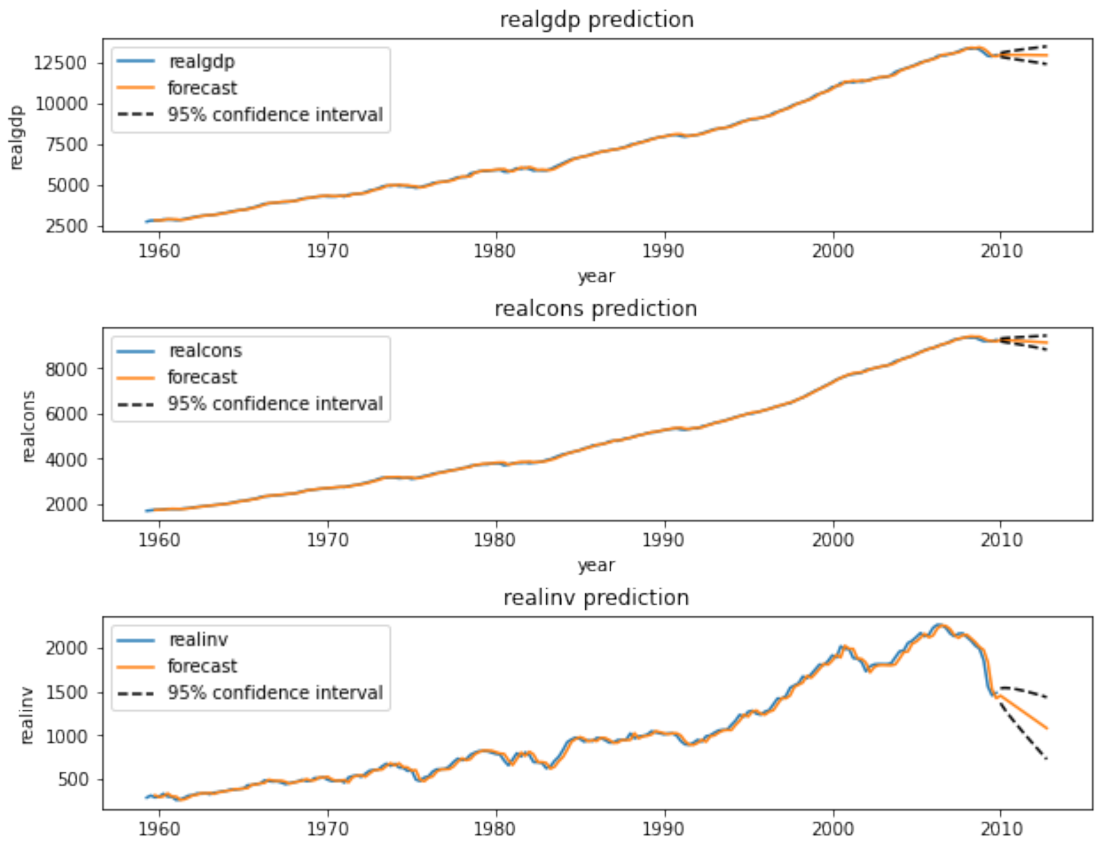

Figure 1.6: Macroeconomic data is forecasted 12 years in the future using statsmodels.

Optional

The `statsmodels` package can help us perform model identification. The method `arma_order_select_ic` will find the optimal order of the ARMA model based on certain criteria. The first parameter `y` is the data. The data must be a NumPy array, not a Pandas DataFrame. The parameter `ic` defines the criteria trying to be minimized. The method will return a dictionary, where the minimal order of each criteria can be accessed.

```
>>> import statsmodels.api as sm
>>> from statsmodel.tsa.stattools import arma_order_select_ic as order_select
>>> import pandas as pd

>>> # Get sunspot data and give DateTimeIndex
>>> sunspot = sm.datasets.sunspots.load_pandas().data
>>> sunspot.index = pd.Index(pd.date_range("1700", end="2009", freq="A-DEC"))
>>> sunspot.drop(columns = ["YEAR"],inplace = True)

>>> # Find best order where p < 5 and q < 5
>>> # Use AICc as basis for minimization
>>> order = order_select(sunspot.values,max_ar=4,max_ma=4,ic=['aic','bic'],↩
    fit_kw={'method':'mle'})
```

```
>>> print(order['aic_min_order'])
(4,2)
>>> print(order['bic_min_order'])
(4,2)

>>> # Fit model
>>> # Note that we need to set the dimensionality to zero in order to have an ↩
    ARMA model.
>>> model = ARIMA(sunspot,order = (4,0,2)).fit(method='innovations_mle')

>>> # Predict values from 1950 to 2012.
>>> prediction = model.predict(start='1950',end='2012')

>>> # Plot the prediction along with the sunspot data.
>>> fig, ax = plt.subplots(figsize=(13,7))
>>> plt.plot(prediction)
>>> plt.plot(sunspot['1950':'2009'])
>>> ax.set_title('Sunspot Dataset')
>>> ax.set_xlabel('Year')
>>> ax.set_ylabel('Number of Sunspots')
>>> plt.show()
```

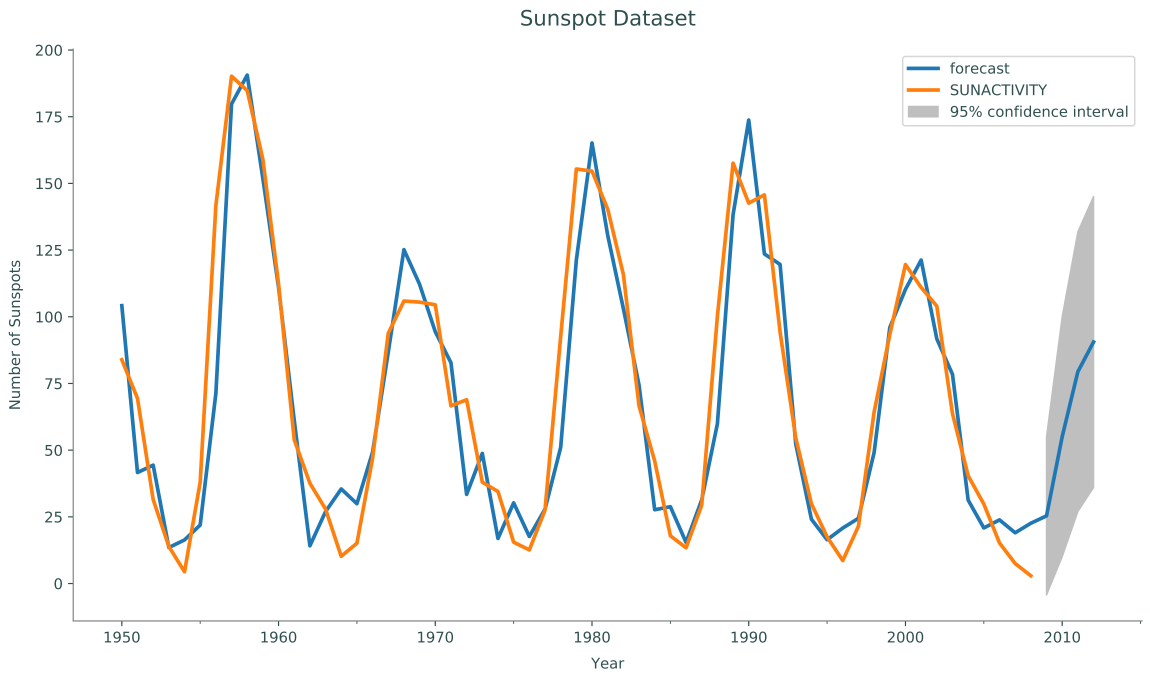

Figure 1.7: Sunspot activity data is forecasted four years in the future using `statsmodels`.

Problem 7. The dataset `manaus` contains data on the height of the Rio Negro from every month between January 1903 and January 1993. Write a function `manaus()` that accepts the forecasting range as strings `start` and `end`, the maximum parameter for the AR model `p` and the maximum parameter of the MA model `q`. The parameters `start` and `end` should be strings corresponding to a DateTimeIndex in the form `Y%M%D`, where `D` is the last day of the month.

The function should determine the optimal order for the ARMA model based on the AIC and the BIC. Then forecast and plot on the range given for both models and compare. Return the order of the AIC model and the order of the BIC model, respectively. For the range `'1983-01-31'` to `'1995-01-31'`, your plot should look like Figure 1.8.

(Hint: The data passed into `arma_order_select_ic` must be a NumPy array. Use the attribute `values` of the Pandas DataFrame.)

To get the `manaus` dataset and set it with a DateTimeIndex, use the following code:

```
>>> # Get dataset
>>> raw = pydata('manaus')
>>> # Convert to DateTimeIndex
>>> manaus = pd.DataFrame(raw.values,index=pd.date_range('1903-01','↩
    1993-01',freq='M'))
>>> manaus = manaus.drop(0,axis=1)
>>> # Set new column title
>>> manaus.columns = ['Water Level']
```

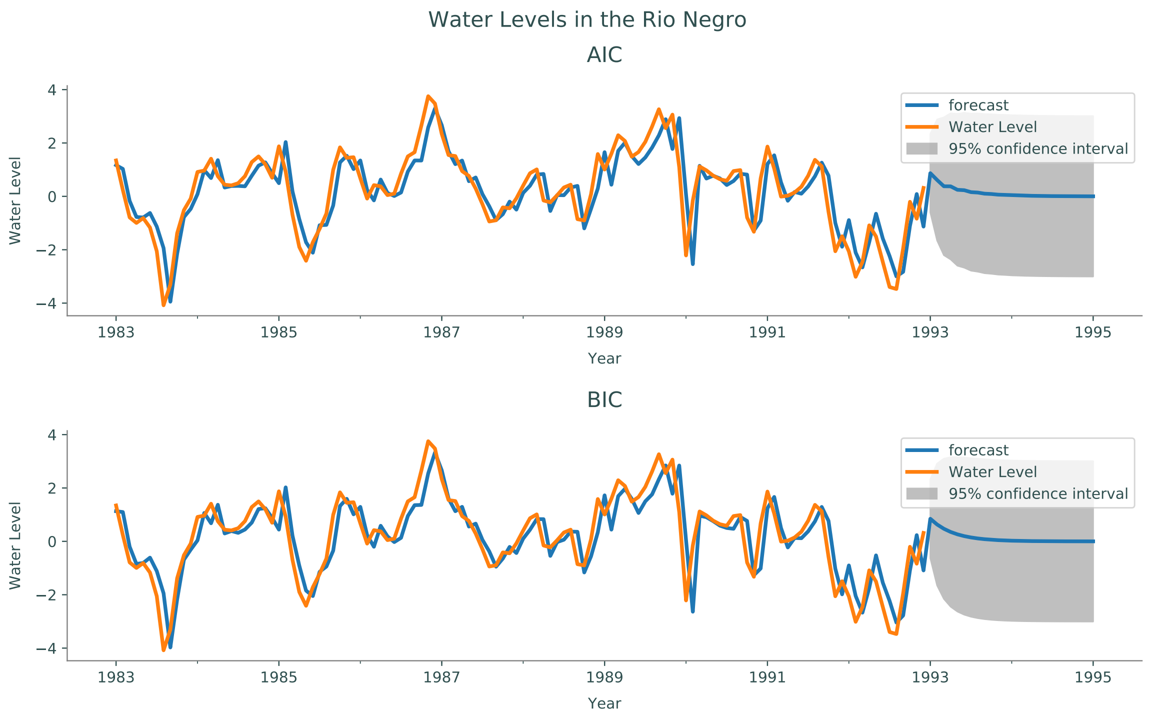

Figure 1.8: AIC and BIC based ARMA models of `manaus` dataset.

Additional Materials

Finding Error Correlation

To find the correlation of the current error with past error, the noise of the data needs to be isolated. Each data point y_t can be decomposed as

$$y_t = T_t + S_t + R_t, \tag{1.23}$$

where T_t is the overall trend of the data, S_t is a seasonal trend, and R_t is noise in the data. The overall trend is what the data tends to do as a whole, while the seasonal trend is what the data does repeatedly. For example, if looking at airfare prices over a decade, the overall trend of the data might be increasing due to inflation. However, we can break this data into individual years. We call each year a season. The seasonal trend of the data might not be strictly increasing, but have increases during busy seasons such as Christmas and summer vacations.

To find T_t, we use an M-fold method. In this case, M is the length of our season. We define the equation

$$T_t = \frac{1}{M} \sum_{-M/2<i<M/2} y_{i+t}. \tag{1.24}$$

This means for each t, we take the average of the season surrounding y_t.

To find the seasonal trend, first subtract the overall trend from the time series. Define $x_t = y_t - T_t$. The value of the seasonal trend can then be found by averaging each day of the season over every season. For example, if the season was one year, we would find the average value on the first day of the year over all seasons, then the second, and so on. Thus,

$$S_t = \frac{1}{K} \sum_{i \equiv t \pmod{M}} x_i \tag{1.25}$$

where K is the number of seasons.

With the overall and seasonal trend known, the noise of the data is simply $R_t = y_t - T_t - S_t$. To determine the strength of correlations with the current error and the past error, plot y_t vs. R_{t-i} as in Figure 1.1.

Proof of Equation 1.15

$$\begin{aligned}
\sum_{i=1}^{p}\phi_i(z_{t-i}-\mu)+a_t+\sum_{j=1}^{q}\theta_j a_{t-j} &= \sum_{i=1}^{p}\phi_i(H\hat{\mathbf{x}}_{t-i})+a_t+\sum_{j=1}^{q}\theta_j a_{t-j} && (1.26)\\
&= \sum_{i=1}^{r}\phi_i(x_{t-i}+\sum_{k=1}^{r-1}\theta_k x_{t-i-k})+a_t+\sum_{j=1}^{r-1}\theta_j a_{t-j} && (1.27)\\
&= a_t+\sum_{i=1}^{r}\phi_i(x_{t-i})+\sum_{j=1}^{r-1}\theta_j\Big(\sum_{i=1}^{r}\phi_i x_{t-j-i}+a_{t-j}\Big) && (1.28)\\
&= a_t+\sum_{i=1}^{r}\phi_i(x_{t-i})+\sum_{j=1}^{r-1}\theta_j x_{t-k} && (1.29)\\
&= x_t+\sum_{j=1}^{r-1}\theta_j x_{t-k}\theta_k x_{t-k} && (1.30)\\
&= z_t. && (1.31)
\end{aligned}$$