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 create an ARMA model using a Kalman filter, and then use said model to forecast future weather data.*

Time Series and ARMA Models

A time series is any discrete-time stochastic process. In other words, it is a sequence of random variables, $(y_t)_{t\in T}$, that are determined by their time t. Examples of time series include the monitoring of heart rate over time, pollution readings over time, stock prices at the closing of each day, and air temperature. Often when using time series, we want to forecast what future observations will be, i.e. what will the pollution be like next week, or how much stock will a company have in 3 months.

One way to forecast a time series is using an ARMA model. An ARMA(p, q) model is a covariance-stationary discrete stochastic process $\{z_t\}$ that is made from an autoregressive model of order p and a moving-average model of order q. The model itself is a stochastic process z_t which satisfies the equation

$$z_t - \mu = \underbrace{\left(\sum_{i=1}^{p} \phi_i (z_{t-i} - \mu)\right)}_{\text{AR(p)}} + a_t + \underbrace{\left(\sum_{j=1}^{q} \theta_j a_{t-j}\right)}_{\text{MA(q)}} \tag{1.1}$$

where $\mu = E[z_t]$ and a_t are identically-distributed Gaussian variables with variance σ_a^2, and ϕ_i and θ_j are constants.

Note

In this lab, we assume that the stochastic process $\{z_t\}$ has $\mu = 0$. This is because the Kalman filter in use assumes that the observed values have $\mu = 0$. Often a stochastic process may be made to have mean 0 by taking the difference $z_t = y_t - y_{t-1}$, where $\{y_t\}$ is the original stochastic process and $\{z_t\}$ is the change in each element.

The first sum on the right hand side of 1.1 is the autoregressive part of the ARMA model; it is a linear combination of p previously observed values of $(z_t - \mu)$. The second sum is the moving average part of the ARMA model. This is similar to finding the average of the current and the previous q error terms in the observations; however, note that the θ_j need not be positive nor sum to one.

Likelihood via Kalman Filter

Let $\Theta = \{\phi_i, \theta_j, \mu, \sigma_a^2\}$ be the set of parameters for an ARMA(p,q) model. Suppose we have a set of observations $z_1, z_2, \ldots, z_n$, denoted collectively by $\{z_t\}$. Using the chain rule, we can factorize the likelihood of the model under these data as

$$p(\{z_t\}|\Theta) = \prod_{t=1}^{n} p(z_t|z_{t-1}, \ldots, z_1, \Theta) \tag{1.2}$$

Our goal is find the p, q, and Θ that maximize this likelihood.

In a general ARMA(p,q) model, the likelihood is a function of the unobserved error terms a_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

$$\hat{\mathbf{x}}_{t|t-1} = \begin{bmatrix} x_{t-1} & x_{t-2} & \cdots & x_{t-r} \end{bmatrix}^T \tag{1.3}$$

$$F = \begin{bmatrix} \phi_1 & \phi_2 & \cdots & \phi_{r-1} & \phi_r \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \tag{1.4}$$

$$H = \begin{bmatrix} 1 & \theta_1 & \theta_2 & \cdots & \theta_{r-1} \end{bmatrix} \tag{1.5}$$

$$Q = \begin{bmatrix} \sigma_a^2 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{1.6}$$

$$w_t \sim \text{MVN}(0, Q), \tag{1.7}$$

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

$$F\hat{\mathbf{x}}_{t-1|t-2} + w_t = \begin{bmatrix} \sum_{i=1}^{r} \phi_i x_{t-i} \\ x_{t-1} \\ x_{t-2} \\ \vdots \\ x_{t-(r-1)} \end{bmatrix} + \begin{bmatrix} a_t \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{1.8}$$

$$= \begin{bmatrix} x_t & x_{t-1} & \cdots & x_{t-(r-1)} \end{bmatrix}^T \tag{1.9}$$

$$= \hat{\mathbf{x}}_{t|t-1} \tag{1.10}$$

We let $z_{t|t-1} = H\hat{\mathbf{x}}_{t|t-1} + \mu$ and verify that this update function satisfies our ARMA(p,q).

$$\begin{aligned}
\sum_{i=1}^{p} \phi_i(z_{t-i} - \mu) + a_t + \sum_{j=1}^{q} \theta_j a_{t-j} &= \sum_{i=1}^{p} \phi_i(H\hat{\mathbf{x}}_{t-i}) + a_t + \sum_{j=1}^{q} \theta_j a_{t-j} && (1.11)\\
&= \sum_{i=1}^{r} \phi_i(x_{t-i} + \sum_{k=1}^{r-1} \theta_k x_{t-i-k}) + a_t + \sum_{j=1}^{r-1} \theta_j a_{t-j} && (1.12)\\
&= a_t + \sum_{i=1}^{r} \phi_i(x_{t-i}) + \sum_{j=1}^{r-1} \theta_j \left(\sum_{i=1}^{r} \phi_i x_{t-j-i} + a_{t-j}\right) && (1.13)\\
&= a_t + \sum_{i=1}^{r} \phi_i(x_{t-i}) + \sum_{j=1}^{r-1} \theta_j x_{t-k} && (1.14)\\
&= x_t + \sum_{j=1}^{r-1} \theta_j x_{t-k} \theta_k x_{t-k} && (1.15)\\
&= z_t. && (1.16)
\end{aligned}$$

Then the linear stochastic dynamical system

$$\hat{\mathbf{x}}_{t+1|t} = F\hat{\mathbf{x}}_{t|t-1} + w_t \quad (1.17)$$
$$z_{t|t-1} = H\hat{\mathbf{x}}_{t|t-1} + \mu \quad (1.18)$$

describes the same process as the original ARMA model. Note that the equation for z_t involves a deterministic component, namely μ. The Kalman filter theory developed in the previous lab, however, assumed no deterministic component for the observations z_t, so you should subtract off the mean μ from the time series observations z_t when using them in the predict and update steps.

Since we have assumed that the error terms in the model are Gaussian, each conditional distribution in 1.2 is also Gaussian, and is completely characterized by its mean and variance. But these two quantities are easily found via the Kalman filter, namely

$$\text{mean} \quad H\hat{\mathbf{x}}_{t|t-1} + \mu \quad (1.19)$$
$$\text{variance} \quad HP_{t|t-1}H^T \quad (1.20)$$

where $\hat{\mathbf{x}}_{t|t-1}$ and $P_{t|t-1}$ are found during the Predict step. Given that each conditional distribution is Guassian, the likelihood can be found as follows:

$$p(\{z_t\}|\Theta) = \prod_{t=1}^{n} N(z_t;\ H\hat{\mathbf{x}}_{t|t-1} + \mu,\ HP_{t|t-1}H^T) \quad (1.21)$$

Problem 1. Write a function that returns the log-likelihood of an ARMA(p,q) model, given a time series z_t. Use the `state_space_rep()` function provided to create F, Q, and H and use the `kalman()` filter provided to calculate the mean and covariance of the error terms.

```
def arma_likelihood(time_series, phis=array([]), thetas=array([]), mu=0.,
        sigma=1.):
    """
    Return the log-likelihood of the ARMA model parameters, given the time
    series.
    Parameters
    ----------
    time_series : ndarray of shape (n,1)
        The time series in question, z_t
    phis : ndarray of shape (p,)
        The phi parameters
    thetas : ndarray of shape (q,)
        The theta parameters
    mu : float
        The parameter mu
    sigma : float
        The standard deviation of the a_t random variables

    Returns
    -------
    log_likelihood : float
        The log-likelihood of the model
    """
    pass
```

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

```
>>> arma_likelihood(ta, phis=array([0.9]), mu=17., sigma=0.4)
-77.6035
```

Identification and Fitting

Now that we can compute the likelihood of a given ARMA model, we want to find the best choice of ARMA model given our time series. In this lab, we define the "best" choice of ARMA model as the model which minimizes the AICc, given by

$$2k\left(1+\frac{k+1}{n-k}\right)-2\ell(\Theta) \tag{1.22}$$

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 1.21 over the space of parameters Θ. We can do so by using the function from Problem 1 along with some optimization routine, such as `scipy.optimize.fmin`.

To minimize the AICc, we perform *model identification*. This is choosing the order of our model out of possible p's and q's. The order of our model which minimizes the AICc is then the optimal model.

Problem 2. Write a function that accepts a time series $\{z_t\}$ and returns the parameters of the model that minimize the AICc, given the constraint that $p \leq 3$, $q \leq 3$.

```
def arma_fit(time_series):
    """
    Return the ARMA model that minimizes AICc for the given time series,
    subject to p,q <= 3.

    Parameters
    ----------
    time_series : ndarray of shape (n,1)
        The time series in question, z_t

    Returns
    -------
    phis : ndarray of shape (p,)
        The phi parameters
    thetas : ndarray of shape (q,)
        The theta parameters
    mu : float
        The parameter mu
    sigma : float
        The standard deviation of the a_t random variables
    """
    pass
```

Here's a hint for performing the optimization at each step, using `scipy.optimize.fmin`.

```
>>> # assume p, q, and time_series are defined
>>> def f(x): # x contains the phis, thetas, mu, and sigma
>>>     return -1*arma_likelihood(time_series, phis=x[:p], thetas=x[p:p+q↩
    ], mu=x[-2],sigma=x[-1])
>>> # create initial point
>>> x0 = np.zeros(p+q+2)
>>> x0[-2] = time_series.mean()
>>> x0[-1] = time_series.std()
>>> sol = op.fmin(f,x0,maxiter=10000, maxfun=10000)
```

The variable `sol` is a flat array of length $p+q+2$, whose first p entries give the optimal values for the ϕ polynomial, the next q entries give the optimal values for the θ polynomial, and the last two entries give the optimal values for μ and σ_a, respectively. Notice that we defined a wrapper function f to feed into the `scipy.optimize.fmin` routine. This wrapper function returns the *negative* of the log likelihood, since the optimization routine we are calling finds the minimum of a function, and we are interested in the *maximum* of the log likelihood.

Your code should produce the following output (it may take a minute or so to run):

```
>>> arma_fit(ta)
(array([ 0.9087]), array([-0.5759]), 17.0652..., 0.3125...)
```

Problem 3. Use your solution from Problem 2 to fit models to the data found in `ta`, `tb`, and `tc`. Report the fitted parameters p, q, Θ.

Forecasting with Kalman Filter

Now that we have identified an 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

$$z_{t+k}|z_1, \cdots, z_t \sim N(z_{t+k};\ H\hat{x}_{t+k|t} + \mu,\ HP_{t+k|t}H^T) \tag{1.23}$$

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

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

Problem 4. The datasets `ta`, `tb` and `tc` contain mock weather data, giving the temperature of the past 197 days. For each data set, find the mean and standard deviation of the next 20 intervals using the Kalman filter. Begin the recursion of the Kalman filter using the update rule and then predict forward using the predict rule. Return the future means and standard deviations (given by $H\hat{x}_{t+k|t} + \mu$ and $\sqrt{HP_{t+k|t}H^T}$, respectively). For each dataset, plot the original data, their expected values, and plus and minus two standard deviations to demonstrate credible intervals.

```python
def arma_forecast(time_series, phis=array([]), thetas=array([]), mu=0.,
        sigma=1., future_periods=20):
    """
    Return forecasts for a time series modeled with the given ARMA model.

    Parameters
    ----------
    time_series : ndarray of shape (n,1)
```

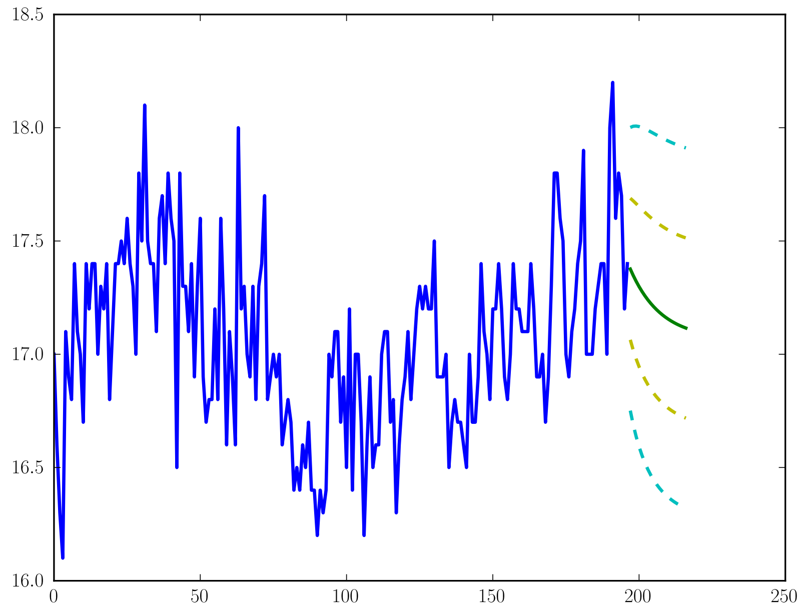
```
        The time series in question, z_t
    phis : ndarray of shape (p,)
        The phi parameters
    thetas : ndarray of shape (q,)
        The theta parameters
    mu : float
        The parameter mu
    sigma : float
        The standard deviation of the a_t random variables
    future_periods : int
        The number of future periods to return

    Returns
    -------
    e_vals : ndarray of shape (future_periods,)
        The expected values of z for times n+1, ..., n+future_periods
    sigs : ndarray of shape (future_periods,)
        The standard deviations of z for times n+1, ..., n+future_periods
    """
    pass
```

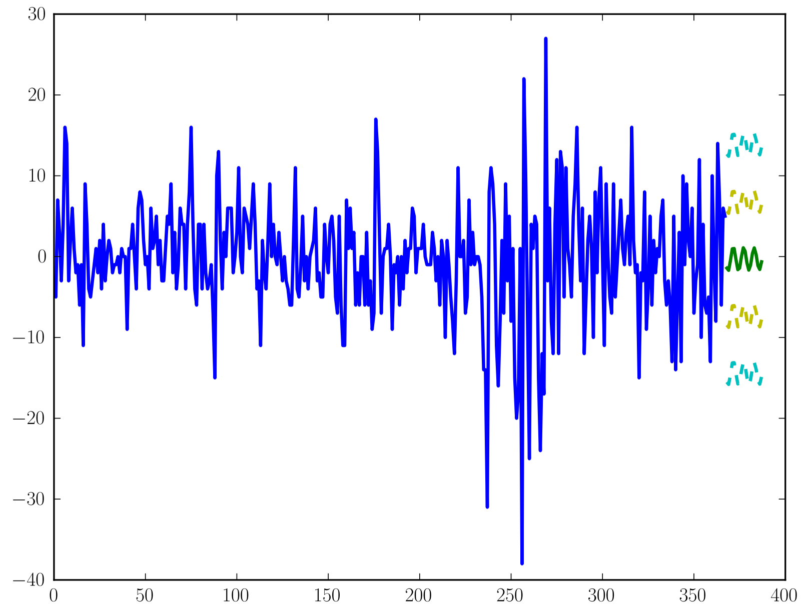
You should get the following result:

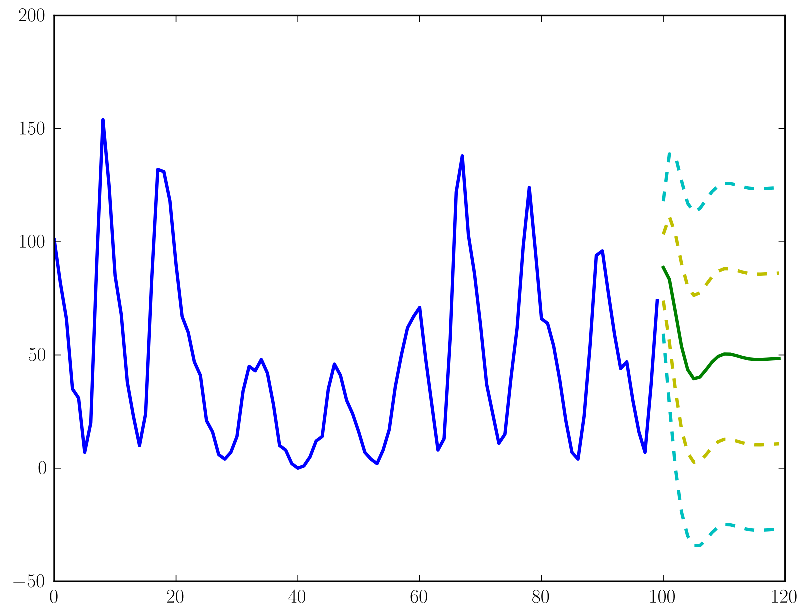
```
>>> arma_forecast(ta, phis, thetas, mu, sigma, 4)
(array([ 17.3762,  17.3478,  17.322 ,  17.2986]),
 array([ 0.3125,  0.3294,  0.3427,  0.3533]))
```

Your results (when using twenty future periods) should match those in Figure 1.1.

Problem 5. Modify your `arma_forecast()` function to generate possible observations using future means and covariance matrices. Plot these generated observations with the original data for each dataset.


18.5
18.0
17.5
17.0
16.5
16.0
0
50
100
150
200
250


30
20
10
0
−10
−20
−30
−40
0
50
100
150
200
250
300
350
400


200
150
100
50
0
−50
0
20
40
60
80
100
120