24 ARMA Models

Lab Objective: *Fit and forecast ARMA models.*

An ARMA(p,q) model is a covariance-stationary discrete stochastic process $\{z_t\}$ that satisfies

$$z_t - \mu = \left(\sum_{i=1}^{p} \phi_i (z_{t-i} - \mu)\right) + a_t + \left(\sum_{j=1}^{q} \theta_j a_{t-j}\right) \tag{24.1}$$

where $\mu = E[z_t]$ and a_t are identically-distributed Gaussian variables with variance σ_a^2. We note that the assumption that $\{z_t\}$ is covariance-stationary is eqivalent to the condition that the roots of the polynomial in B

$$\phi(B) = 1 - \sum_{i=1}^{p} \phi_i B^i \tag{24.2}$$

lie outside of the unit circle.

The first sum on the right hand side of 24.1 is interpreted as an "autoregression" since it is a linear combination of previously observed values of z_t. The second sum is interpreted as a "moving average" of the current and previous error terms; though formally similar to an average, note that the θ_j need not be positive nor sum to one. We say that an ARMA(p,q) model is an "autoregressive moving-average model of order p, q".

Likelihood via Kalman Filter

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 a state-space representation of an ARMA(p,q) model. If $r = \max(p, q+1)$,

we write

$$F = \begin{bmatrix} \phi_1 & \phi_2 & \cdots & \phi_{r-1} & \phi_r \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \tag{24.3}$$

$$H = \begin{bmatrix} 1 & \theta_1 & \theta_2 & \cdots & \theta_{r-1} \end{bmatrix} \tag{24.4}$$

$$Q = \begin{bmatrix} \sigma_a^2 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{24.5}$$

$$w_t \sim \text{MVN}(0, Q), \tag{24.6}$$

where $\phi_i = 0$ for $i > p$, and $\theta_j = 0$ for $j > q$. Then the linear stochastic dynamical system

$$x_{t+1} = Fx_t + w_t \tag{24.7}$$

$$z_t = Hx_t + \mu \tag{24.8}$$

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.

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

$$p(\{z_t\}|\Theta) = \prod_{t=1}^{n} p(z_t|z_{t-1}, \ldots, z_1, \Theta) \tag{24.9}$$

Since we have assumed that the error terms are Gaussian, each conditional distribution in 24.9 is also Gaussian, and is completely characterized by its mean and variance. But these two quantities are easily found via the Kalman filter, namely

$$\text{mean} \quad H\hat{x}_{t|t-1} + \mu \tag{24.10}$$

$$\text{variance} \quad HP_{t|t-1}H^T \tag{24.11}$$

where $\hat{x}_{t|t-1}$ and $P_{t|t-1}$ are found during the Predict step. The likelihood becomes

$$p(\{z_t\}|\Theta) = \prod_{t=1}^{n} N(z_t;\ H\hat{x}_{t|t-1} + \mu,\ HP_{t|t-1}H^T) \tag{24.12}$$

We begin the recursion by letting

$$\hat{x}_{1|0} = \mathbb{E}(x_1) = 0 \tag{24.13}$$

$$\text{vec}(P_{1|0}) = \mathbb{E}\left[(x_1 - \mathbb{E}x_1)(x_1 - \mathbb{E}x_1)^T\right] = \left[I_{r^2} - (F \otimes F)\right]^{-1} \cdot \text{vec}(Q) \tag{24.14}$$

where vec flattens a matrix and $\otimes$ is the Kronecker product (`numpy.kron`).

Problem 1. Write a function that computes the log-likelihood of an ARMA(p, q) model, given a time series z_t.

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

Identification and Fitting

When modeling a data set with an ARMA(p, q) model, the order of the model must be determined, as well as the other parameters. The process of choosing p and q is called *model identification*. Different methods have been used; for example, Box and Jenkins propose a methodology that involves examining the estimated autocorrelation and partial-autocorrelation functions of the data. We will choose p and q that minimize the Akaike information criterion with a correction (AICc), given by

$$2k\left(1+\frac{k+1}{n-k}\right)-2\ell(\Theta) \tag{24.15}$$

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 24.12 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`.

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, where the input data is found in `time_series_a.txt` (it may take a minute or so to run):

```
>>> arma_fit(time_series_a)
(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 `time_series_a.txt`, `time_series_b.txt`, `time_series_c.txt`. Report the fitted parameters p, q, Θ.

Forecasting

The Kalman filter provides a straightforward way to predict future states, by giving the mean and variance of the conditional distribution of future observations.

$$z_{t+k}|z_1, \cdots, z_t \sim N(z_{t+k};\ H\hat{x}_{t+k|t} + \mu,\ HP_{t+k|t}H^T) \tag{24.16}$$

Recall the relations

$$\hat{x}_{t+k|t} = F\hat{x}_{t+k-1|t} \tag{24.17}$$

$$P_{t+k|t} = FP_{t+k-1|t}F^T + Q \tag{24.18}$$

Problem 4. Forecast each data set ahead 20 intervals using the parameters discovered from Problem 3, and plot their expected values along with the original data set. Also plot the expected values plus and minus σ_{t+k}, and plus and minus $2\sigma_{t+k}$ to demonstrate credible intervals.

Note that we need the values of $\hat{x}_{n|n}$ and $P_{n|n}$ to get started. As usual, these estimates can be found using the Predict and Update recursions. Initialize $\hat{x}_{1|0}$ and $P_{1|0}$ as before, run the recursions until you obtain $\hat{x}_{n|n}$ and $P_{n|n}$, and then calculate the future estimates $\hat{x}_{t+k|t}$ and $P_{t+k|t}$. Use these to calculate the expected value and standard deviation for forecasted values (given by $H\hat{x}_{t+k|t} + \mu$ and $\sqrt{HP_{t+k|t}H^T}$, respectively).

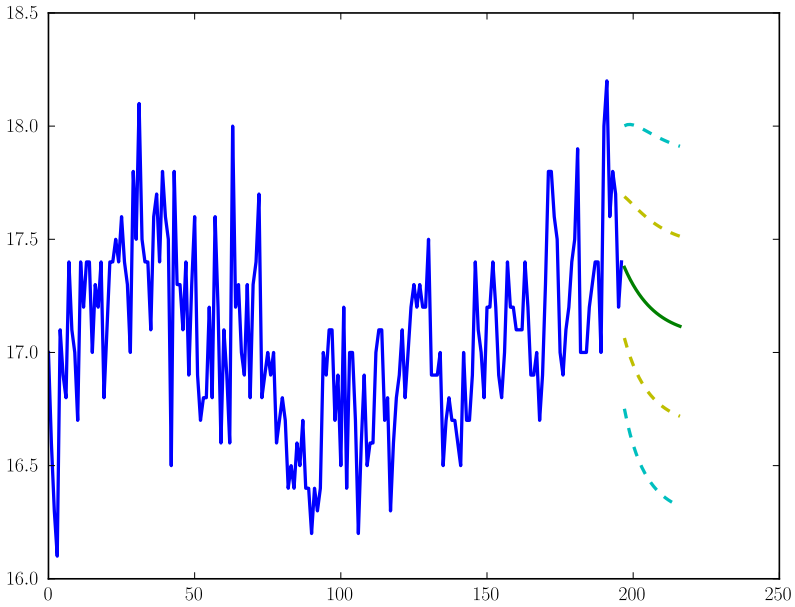
```
def arma_forecast(time_series, phis=array([]), thetas=array([]), mu=0.,
        sigma=1., future_periods=20):
    """
    Return forecasts for a time series modeled with the given ARMA model.

    Parameters
    ----------
```

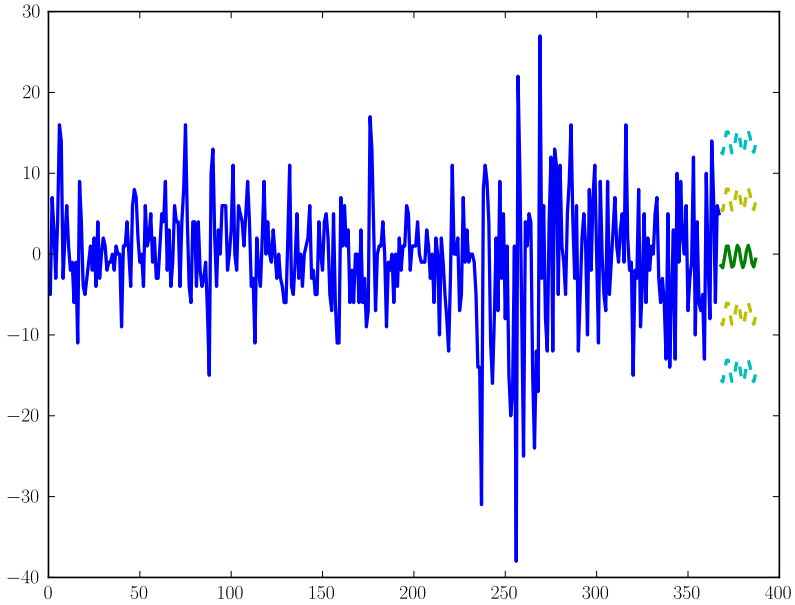
```
    time_series : ndarray of shape (n,1)
        The time series in question, z_t
    phis : ndarray of shape (p,)
        The phi parameters
    thetas : ndarray of shape (q,)
        The theta parameters
    mu : float
        The parameter mu
    sigma : float
        The standard deviation of 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
```

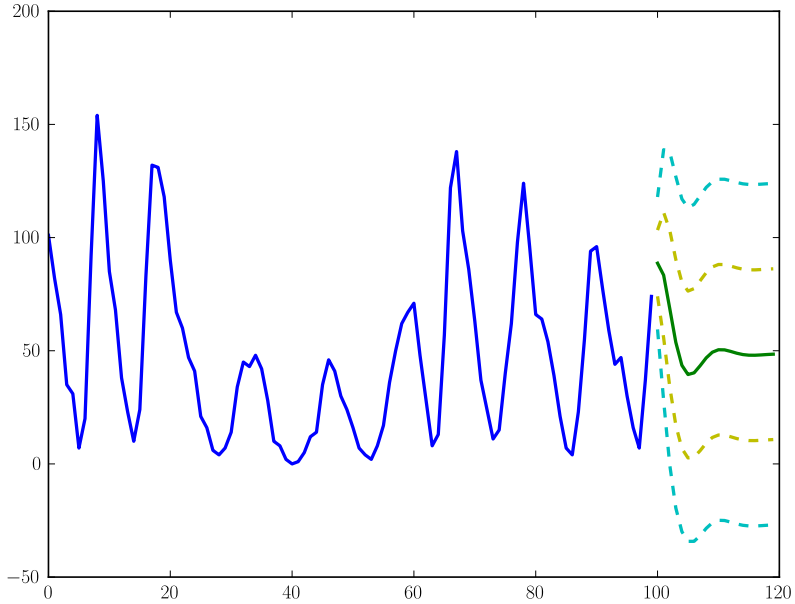
You should get the following result:

```
>>> arma_forecast(time_series_a, 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 24.1.


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