Topics in Time Series Analysis: Assignment 9

Lucas Cruz Fernandez

1544674, lcruzfer@mail.uni-mannheim.de

18th of May 2020

Q15: MLE estimation and comparison

In this part we will first derive the log-likelihood objective function which we will then apply to a generated sequence of x_t 1000 times to again calculate biases and MSE of the coefficient as well as the variance estimate.
The log-likelihood function is given by:

$$\mathcal{L}(b, \sigma^2) = \sum_{t=2}^{T} ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2\sigma^2}\left(x_t - \bar{x}\right)^2$$

We can derive the conditional distribution of $x_t|x_{t-1}, x_{t-2}, ...$ (denoted by $x_t|I_{t-1}$) and plug in the expected value for $\bar{x}$ to approximate the likelihood function. We assume that the underlying error of our MA(1) process follows a standard normal distribution and can therefore derive:

$$\begin{aligned} E\left[x_t|I_{t-1}\right] &= E[\epsilon_t|I_{t-1}] + bE[\epsilon_{t-1}|I_{t-1}] \\ &= b\epsilon_{t-1} \end{aligned}$$

Since by assumption, ϵ_t is iid, we the conditional expectation of today is independent of the past, thus $E[\epsilon_t|I_t] = E[\epsilon_t]$, while the past observation in $t-1$ conditional on all information we have in $t-1$ is a deterministic variable, which leads to the result presented above.
If we plug this into our objective function we get:

$$\begin{aligned} \mathcal{L}(b, \sigma^2) &= \sum_{t=2}^{T} ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2\sigma^2}\left(\epsilon_t + b\epsilon_{t-1} - b\epsilon_{t-1}\right)^2 \\ &= (T-1)ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2\sigma^2}\sum_{t=2}^{T}\epsilon_t^2 \\ &= (T-1)\left[ln(1) - ln(\sqrt{2\pi\sigma^2})\right] - \frac{1}{2\sigma^2}\sum_{t=2}^{T}\epsilon_t^2 \end{aligned}$$

However, this function still depends on two parameters, namely b and σ^2. Therefore, by taking the first derivative with respect to σ^2, finding $\sigma^2(b)$ and plugging back into the objective function allows us to "single out" σ^2 as we did for the Whittle-Estimator as well.

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = -\frac{(T-1)}{2}\frac{1}{\sqrt{2\pi\sigma^2}}(2\pi\sigma^2)^{-\frac{1}{2}}2\pi + \frac{2}{(2\sigma^2)^2}\sum_{t=2}^{T}\epsilon_t^2 \stackrel{!}{=} 0$$

$$\frac{(T-1)}{2}\frac{1}{2\pi\sigma^2}2\pi = \frac{1}{2\sigma^4}\sum_{t=2}^{T}\epsilon_t^2$$

$$\sigma^2 = \frac{1}{T-1}\sum_{t=2}^{T}\epsilon_t^2$$

Still, this function does not depend on b but we are able to rewrite, and therefore implicitly "estimate" ϵ_t as:

$$\epsilon_t = \sum_{s=0}^{t}(-1)^s b^s x_{t-s}$$

Putting everything together we end up with the following objective function:

$$\mathcal{L}(b) = (T-1)ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2\sigma^2}\sum_{t=2}^{T}(\epsilon_t)^2$$

where ϵ_t and σ^2 are defined as above.

Implementation

Now, we can implement the estimator in Python and see how well it does in comparison with the frequency domain estimates obtained by the Whittle estimator.
The first function presented below calculates all ϵ_t for all $t = 1, ..., T$ based on a given series $\{x_t\}_{t=1}^{T}$ and a coefficient b. We can supply an array of dimensions 3 x T, where each row is representing a series of one of the different specifications.

```
def get_eps_t(xt, b, T):
    '''
    Note: this cannot handle single array (due to reshape)
    If want single array of x to be handled, remove reshaping of res in loop
    Get *all* epsilon_t following summation approach.
    Returns an array with all eps_t and and array with all epsilon_t^2, which we
        need for sigma^2.
    *xt = 1D array of x
    *b = scalar parameter
    *T = # of periods, i.e. len(xt)
    '''
    final_norm = np.empty((n_specs, 1))
```

```
    for t in range(T+1):
        #get x_{t-s} up to x_t
        #turn around order such that x_t is first element and x_1 is last
        sub_x = np.flip(xt[:, :t], axis = 1)
        sub_x_II = np.empty((n_specs, len(sub_x[0])))
        #loop over all different x-specifications and calculate epsilon as
            summation of x_t following formula
        for j in range(n_specs):
            sub_x_II[j] = np.array([b**i * x_t * (-1)**i for i, x_t in
                enumerate(sub_x[j])])
        res = np.sum(sub_x_II, axis = 1)
        res = np.reshape(res, (n_specs, 1))
        final_norm = np.hstack((final_norm, res))
    #drop first observation since this is just initializing, empty array
    final_norm = final_norm[:, 2:]
    return(final_norm)
```

The function below finds the optimal estimate for b on a grid between -0.9 and 0.9. This is done by first calculating the ϵ_t of interest for a value on the grid, then finding σ^2 based on these estimates and finally putting all together in the objective function, calculating the value of $\mathcal{L}(b = parameter from grid)$. Finally, a 3 x 1 is returned, which contains the optimal estimate for b in each specification.

```
def lll_hood_opt(xt, bs, T):
    #set up empty array on which results are stacked
    values = np.empty((n_specs, 1))
    for param in bs:
        eps_hat = get_eps_t(xt, param, T = T)
        sigma = 1/(T-1) * np.sum(eps_hat**2, axis = 1)
        #can get sum of sqrd epsilon directly and save it
        res = (T-1)*np.log(1/np.sqrt(2*np.pi*sigma)) - (1/(2*sigma)) *
            np.sum(eps_hat**2)
        res = np.reshape(res, (n_specs, 1))
        values = np.hstack((values, res))
    #discard initializing empty array again
    values = values[:, 1:]
    max_i = np.argmax(values, 1)
    opt_param = np.empty((n_specs, 1))
    for i in range(n_specs):
        opt_param[i] = bs[max_i[i]]
    return(opt_param)
```

We then set up the basic paramters (T = 1000, S = 1000, b = 0.25) and save the results of each iteration in an array, where we end up with a 3 x 1000 array containing the optimal b for each model in each iteration.

```
for s in range(S):
    eps = np.empty((n_specs, T+1))
    eps[0] = rd.normal(0, 1, T+1)
    eps[1] = t.rvs(df = 5, size = T+1)
    eps[2] = rd.uniform(0, 1, T+1)
    eps[:, 0] = 0
    eps_lag = eps[:, :T]
    #start epsilon from second element because we added eps[0] = 0
    x = eps[:, 1:] + b*eps_lag
    results = lll_hood_opt(x, bs, T)
    opt_params = np.hstack((opt_params, results))
```

We then calculate our estimate for σ^2 based on these estimates for b, getting an array of teh same form as the $\hat{b}$ array only filled with the respective σ^2. Again, we apply bias and MSE functions (see Appendix) to calculate the respective variable for the two parameters we are estimating. The results are presented in Table (1), while for comparison the results based on the Whittle estimator are presented in Table (2).

Note: the following discussion is with respect to the absolute values of the biases and MSEs of interest.

The bias and MSE for the first two specifications, i.e. ϵ_t being standard normal or t(5) distributed, do only differ slightly. The Whittle estimates for bias and MSE are marginally larger. The picture turns however when looking at the series building on uniformly distributed ϵ_t. Here, we clearly see that the Whittle estimator yields more precise (less biased) results for $\hat{b}$ as well as $\hat{\sigma^2}$. Taking a closer look at the estimation approaches the reason for this pattern becomes evident. The Whittle estimator places no assumptions on the distribution of x_t, or in other words the distribution does not matter when we transfer the series into the frequency domain. On the other hand, the MLE in the time domain assumes that the underlying x_t are normally distributed. Since the Student's t-distribution is fairly close to the standard normal distribution, the bias and MSE are relatively low as well. However, the weakness of the time domain approach becomes clear when looking at the last specification: assuming the wrong underlying distribution leads to strongly biased estimates. On the other hand, the frequency domain perform worse compared to MLE when the underlying distribution is indeed normal or a t-distribution (however, since we only look at a t-distribution with 5 degrees of freedom we cannot make any generalizing statements here; maybe a higher or lower degree of freedom lets the frequency estimator perform better).

	b	σ_ϵ^2
Bias model I	3.1	-0.66
Bias model II	-0.07	3
Bias model III	407	166
MSE model I	1.2	2.07
MSE model II	1.17	22.3
MSE model III	162	28

Table 1: Results using the MLE Approach

	b	σ_ϵ^2
Bias model I	-2.58	1.9788
Bias model II	-0.917	-0.77
Bias model III	2.083	-0.295
MSE model I	1.979	4.668547
MSE model II	2.108	32.43751
MSE model III	2.216	0.02183261

Table 2: Results using the Whittle Estimation Approach

Appendix

```
def bias(estimate, true_val, T):
    '''
    Calculate bias as given in assignment.
    '''
    reps = len(estimate)
    summation = np.sum(estimate)
    bias = 1/reps * summation - true_val
    return(bias)

def MSE(bias, parameter, T):
    '''
    *parameter = parameter estimates
    '''
    bias2 = bias**2
    variance = np.var(parameter)
    return((bias2 + variance)*T)
```