Assignment 2

April 16, 2024

This document describes our process to code the conditional and unconditional likelihood functions for the AR(7) process based on the starter code provided in the assignment. Furthermore, it outlines our approach to estimate the AR(7) paramteres for the `INDPRO` variable from the FRED-MD dataset and our forecast.

Contents

1 Conditional Likelihood

1.1 Theoretical Outline

The AR(7) Model is defined as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \phi_3 y_{t-3} + \phi_4 y_{t-4} + \phi_5 y_{t-5} + \phi_6 y_{t-6} + \phi_7 y_{t-7} + u_t \tag{1}$$

where $u_t \sim N(0, \sigma^2)$.

We can then define the conditional likelihood function as:

$$L(y, \phi) = f(y_1, y_2, y_3, y_4, y_5, y_6, y_7; \phi) \prod_{t=8}^{T} f(y_t | y_{t-1}, ..., y_{t-7}; \phi) \tag{2}$$

The conditional log-likelihood function is:

$$\ell(y;\phi) = \log(f(y_1, ..., y_7; \phi)) + \sum_{t=8}^{T} \log(f(y_t|y_{t-1}, ..., y_{t-7}; \phi)) \tag{3}$$

We regard the first 7 observation or their joint distribution as a determenistic quantity and thus:

$$\ell(y;\phi) \propto \sum_{t=8}^{T} \log(f(y_t|y_{t-1}, ..., y_{t-7}; \phi)) \tag{4}$$

Also, since the error terms are normally distributed with mean 0 and varieance σ^2, we can determine the conditional distribution of y_t as:

$$y_t|y_{t-1}, y_{t-2}, ..., y_{t-7} \sim N(c + \phi_1 y_{t-1} + ... + \phi_7 y_{t-7}, \sigma^2) \tag{5}$$

Therefore, to implement the conditional log-likelihood function we can code it as the sum of normally distributed variables.

$$\ell_C(y;\phi) \propto \sum_{t=8}^{T} \log\left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{(y_t - (c + \phi_1 y_{t-1} + ... + \phi_7 y_{t-7}))^2}{2\sigma^2}\right\}\right) \tag{6}$$

1.2 Implementation in Python

```python
from scipy.stats import norm
from scipy.stats import multivariate_normal
import numpy as np

def lagged_matrix(Y, max_lag=7):
    n = len(Y)
    lagged_matrix = np.full((n, max_lag), np.nan)
    # Fill each column with the appropriately lagged data
    for lag in range(1, max_lag + 1):
        lagged_matrix[lag:, lag - 1] = Y[:-lag]
    return lagged_matrix


def cond_loglikelihood_ar7(params, y):
    c = params[0]
    phi = params[1:8]
    sigma2 = params[8]
    mu, Sigma, stationary = unconditional_ar_mean_variance(c, phi, sigma2)
    ## We could check that at phis the process is stationary and return -Inf if it is not
    if not(stationary):
        return -np.inf
    ## The distribution of
    # y_t|y_{t-1}, ..., y_{t-7} ~ N(c+\phi_{1}*y_{t-1}+...+\phi_{7}y_{t-7}, sigma2)
```

```
    ## Create lagged matrix
    X = lagged_matrix(y, 7)
    yf = y[7:]
    Xf = X[7:,:]
    loglik = np.sum(norm.logpdf(yf, loc=(c + Xf@phi), scale=np.sqrt(sigma2)))
    return loglik
```

2 Unconditional Likelihood

2.1 Theoretical Outline

The unconditional likelihood function starts from a similar approach. However, instead of regarding the first 7 observations as determenistic it regards them as multivariate normally distributed with mean μ and covariance Σ.

$$\ell_U(y;\phi) = \log(f(y_1,...,y_7;\phi,\mu,\Sigma)) + \sum_{t=8}^{T} \log(f(y_t|y_{t-1},...,y_{t-7};\phi)) \tag{7}$$

The second term on the right hand side is the conditional log-likelihood, the first term is the multivariate normal distribution of the first seven observations.

As described in the assignment the parameters of the multivariate normal distribution can be obtained through:

$$\mu = (I - A)^{-1}b \tag{8}$$

to compute the mean and the Lyapunov equation

$$\Sigma = A\Sigma A^T + Q \tag{9}$$

to compute the covariance matrix of the autoregressive process.

2.2 Implementation in Python

First, the matrix A is constructed, which is a square matrix where the first row is made up of $\phi_1, \phi_2, ..., \phi_p$ of the autoregressive process of order p and the other rows are in reduced row echelon form.

For this the command `np.zeros((p,p))` is used, which creates an array of dimension $p \times p$ filled with zeros. Next, we replace the first row with $\phi_1, \phi_2, ..., \phi_p$ by subsetting A with the command `A[0, :] = phis`. Next, to fill the remaining rows except the last row with leading ones the comman np.eye is used.

Thus for given p and vector ϕ, this code creates the matrix A and then calculates the mean and the covariance. The code also checks whether the AR(7) process is stationary with respect to the given parameters.

```
def unconditional_ar_mean_variance(c, phis, sigma2):
    ## The length of phis is p
    p = len(phis)
```

```python
    A = np.zeros((p, p))
    A[0, :] = phis
    A[1:, 0:(p-1)] = np.eye(p-1)
    ## Check for stationarity
    eigA = np.linalg.eig(A)
    if all(np.abs(eigA.eigenvalues)<1):
        stationary = True
    else:
        stationary = False
    # Create the vector b
    b = np.zeros((p, 1))
    b[0, 0] = c

    # Compute the mean using matrix algebra
    I = np.eye(p)
    mu = np.linalg.inv(I - A) @ b

    # Solve the discrete Lyapunov equation
    Q = np.zeros((p, p))
    Q[0, 0] = sigma2
    #Sigma = np.linalg.solve(I - np.kron(A, A), Q.flatten()).reshape(7, 7)
    Sigma = scipy.linalg.solve_discrete_lyapunov(A, Q)

    return mu.ravel(), Sigma, stationary
```

This function is used to build the unconditional likelihood function:

```python
def uncond_loglikelihood_ar7(params, y):
    ## The unconditional loglikelihood
    ## is the unconditional "plus" the density of the
    ## first p (7 in our case) observations
    cloglik = cond_loglikelihood_ar7(params, y)

    ## Calculate initial
    # y_1, ..., y_7 ~ N(mu, sigma_y)
    c = params[0]
    phi = params[1:8]
    sigma2 = params[8]
    mu, Sigma, stationary = unconditional_ar_mean_variance(c, phi, sigma2)
    if not(stationary):
        return -np.inf
    mvn = multivariate_normal(mean=mu, cov=Sigma, allow_singular=True)
    uloglik = cloglik + mvn.logpdf(y[0:7])
    return uloglik
```

3 Maximum Likelihood Estimation for `INDPRO`

As in assignment 1, we import the FRED-MD dataset using `pandas`. Since we are only interested in the `INDPRO` variable, we select it and transform it using log differences:

```python
import pandas as pd
import numpy as np
#Read Data
df = df = pd.read_csv('~/Downloads/current.csv')
#Select INDPRO
INDPRO = df['INDPRO']
#Drop first Row
INDPRO = INDPRO.drop(index=0)
#transform INDPRO using log differences
INDPRO = np.log(INDPRO).diff()

#implement Starter Code from the assignment
## Lagged Matrix Function
def lagged_matrix(Y, max_lag=7):
    n = len(Y)
    lagged_matrix = np.full((n, max_lag), np.nan)
    # Fill each column with the appropriately lagged data
    for lag in range(1, max_lag + 1):
        lagged_matrix[lag:, lag - 1] = Y[:-lag]
    return lagged_matrix

## Mean- Variance - Stationarity Function
def unconditional_ar_mean_variance(c, phis, sigma2):
    ## The length of phis is p
    p = len(phis)
    A = np.zeros((p, p))
    A[0, :] = phis
    A[1:, 0:(p-1)] = np.eye(p-1)
    ## Check for stationarity
    eigA = np.linalg.eig(A)
    if all(np.abs(eigA.eigenvalues)<1):
        stationary = True
    else:
        stationary = False
    # Create the vector b
    b = np.zeros((p, 1))
    b[0, 0] = c

    # Compute the mean using matrix algebra
    I = np.eye(p)
    mu = np.linalg.inv(I - A) @ b
```

```python
    # Solve the discrete Lyapunov equation
    Q = np.zeros((p, p))
    Q[0, 0] = sigma2
    #Sigma = np.linalg.solve(I - np.kron(A, A), Q.flatten()).reshape(7, 7)
    Sigma = scipy.linalg.solve_discrete_lyapunov(A, Q)

    return mu.ravel(), Sigma, stationary

## Conditional Likelihood
def cond_loglikelihood_ar7(params, y):
    c = params[0]
    phi = params[1:8]
    sigma2 = params[8]
    mu, Sigma, stationary = unconditional_ar_mean_variance(c, phi, sigma2)
    ## We could check that at phis the process is stationary and return -Inf if it is not
    if not(stationary):
        return -np.inf
    ## The distribution of
    # y_t|y_{t-1}, ..., y_{t-7} ~ N(c+\phi_{1}*y_{t-1}+...+\phi_{7}y_{t-7}, sigma2)
    ## Create lagged matrix
    X = lagged_matrix(y, 7)
    yf = y[7:]
    Xf = X[7:,:]
    loglik = np.sum(norm.logpdf(yf, loc=(c + Xf@phi), scale=np.sqrt(sigma2)))
    return loglik

## Unconditional Likelihood
def uncond_loglikelihood_ar7(params, y):
    ## The unconditional loglikelihood
    ## is the unconditional "plus" the density of the
    ## first p (7 in our case) observations
    cloglik = cond_loglikelihood_ar7(params, y)

    ## Calculate initial
    # y_1, ..., y_7 ~ N(mu, sigma_y)
    c = params[0]
    phi = params[1:8]
    sigma2 = params[8]
    mu, Sigma, stationary = unconditional_ar_mean_variance(c, phi, sigma2)
    if not(stationary):
        return -np.inf
    mvn = multivariate_normal(mean=mu, cov=Sigma, allow_singular=True)
    uloglik = cloglik + mvn.logpdf(y[0:7])
    return uloglik
```

```python
# Using INDPRO as the target variable.
## Computing OLS
X = lagged_matrix(INDPRO, 7)
yf = INDPRO[7:]
Xf = np.hstack((np.ones((len(INDPRO)-7,1)), X[7:,:]))
beta = np.linalg.solve(Xf.T@Xf, Xf.T@yf)
sigma2_hat = np.mean((yf - Xf@beta)**2)
params= np.hstack((beta, sigma2_hat))
print("The parameters of the OLS model are", params)

# to maximize likelihood a function of the negative likelihood is defined to be minimized
params = np.array([
    0.0012, ## c
    0.0291, 0.07, 0.059, 0.04, 0.04, 0.02, 0.06, ## phi
    0.008 ## sigma2
    ])

def cobj(params, y):
    return - cond_loglikelihood_ar7(params,y)

# Maximizing the likelihood
results = scipy.optimize.minimize(fun = cobj, x0 =  params, args = INDPRO, method='L-BFGS-B')
print("The parameters estimated by maximizing the conditional likelihood are:", results.x)

#Same Procedure for unconditional likelihood
params= np.hstack((beta, sigma2_hat))

def uobj(params, y):
    return - uncond_loglikelihood_ar7(params,y)

results = scipy.optimize.minimize(uobj, params, args = INDPRO, method='Nelder-Mead')
print("The parameters estimated by maximizing the unconditional likelihood are:", results.x)
```

As the next step we construct the matrix containing INDPRO and its lagged values .

3.1   Results

We estimate four different models for the conditional likelihood:

1. `Model 1`: We use the OLS coefficients as the initial guess and minimize the conditional likelihood function using the 'L-BFGS-B' method.

2. `Model 2`: We use the OLS coefficients as the initial guess and minimize the conditional likelihood function using the 'Neler-Mead' method.

3. `Model 3`: We supply a different initial guess and minimize the conditional likelihood function using the 'L-BFGS-B' method.

4. `Model 4`: We supply a different initial guess and minimize the conditional likelihood function using the 'Neler-Mead' method.

```
# Now we estimate multiple models:
## OLS model
modOLS = np.hstack((beta, sigma2_hat))
## Conditional Likelihood
## 1. Using the OLS parameters as the initial guess
### L-BFGS-B
mod1 = scipy.optimize.minimize(fun = cobj, x0 =  modOLS, args = INDPRO, method='L-BFGS-B').x
### Nelder-Mead
mod2 = scipy.optimize.minimize(fun = cobj, x0 =  modOLS, args = INDPRO, method='Nelder-Mead').x
## 2. Using a slightly different initial guess
Initial_Guess = np.array([
    0.0012, ## c
    0.0291, 0.07, 0.059, 0.04, 0.04, 0.02, 0.06, ## phi
    0.009 ## sigma2
])
### L-BFGS-B
mod3 = scipy.optimize.minimize(fun = cobj, x0 =  Initial_Guess, args = INDPRO, method='L-BFGS-B').x
### Nelder-Mead
mod4 = scipy.optimize.minimize(fun = cobj, x0 =  Initial_Guess, args = INDPRO, method='Nelder-Mead'

## Unbounded Unconditional Likelihood
## 1. Using the OLS parameters as the initial guess
### L-BFGS-B
mod5 = scipy.optimize.minimize(fun = uobj, x0 =  modOLS, args = INDPRO, method='L-BFGS-B').x
### Nelder-Mead
mod6 = scipy.optimize.minimize(fun = uobj, x0 =  modOLS, args = INDPRO, method='Nelder-Mead').x
## 2. Using a slightly different initial guess
Initial_Guess = np.array([
    0.0012, ## c
    0.00291, 0.007, 0.0509, 0.0024, 0.0409, 0.012, 0.0601, ## phi
    0.009 ## sigma2
])
### L-BFGS-B
#mod7 = scipy.optimize.minimize(fun = uobj, x0 =  Initial_Guess, args = INDPRO, method='L-BFGS-B').
### Nelder-Mead
#mod8 = scipy.optimize.minimize(fun = uobj, x0 =  Initial_Guess, args = INDPRO, method='Nelder-Mead

mods = np.array([modOLS, mod1, mod2, mod3, mod4])
print(mods)
#np.savetxt("mods.txt", mods)
modsDF = pd.DataFrame(mods)
modsDF = modsDF.T
```

```
print(modsDF)
print(modsDF.to_latex())
```

The results are displayed in Table 1. The results show that if the initial guess are the OLS coefficients, the conditional MLE correctly finds identical coefficients. However, when we supply a different initial guess, the minimization procedure using 'L-BFGS-B' returns simply the same initial guess while using the 'Neler-Mead' method changes the estimated coefficients quite dramatically. This could hint towards errors in our code, multiple local minima or a non-continuous likelihood function.

Coefficients	OLS	Model 1	Model 2	Model 3	Model 4
c	0.001263	0.001263	0.001263	0.001200	0.000874
ϕ_1	0.291263	0.291263	0.291263	0.029100	0.056374
ϕ_2	-0.074517	-0.074517	-0.074517	0.070000	0.017666
ϕ_3	0.046809	0.046809	0.046809	0.059000	0.060468
ϕ_4	0.047565	0.047565	0.047565	0.040000	0.009790
ϕ_5	-0.024910	-0.024910	-0.024910	0.040000	0.058602
ϕ_6	0.061501	0.061501	0.061501	0.020000	0.038055
ϕ_7	0.020106	0.020106	0.020106	0.060000	0.114138
σ^2	0.000089	0.000089	0.000089	0.009000	0.000095

Table 1: Estimation results for MLE using the conditional likelihood function. Colums OLS reports the OLS coefficients.

4   Forecast