The AR(7) Model

The AR(7) Model is defined as:

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

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

We can then define the conditional likelihood function as:

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

The conditional log-likelihood function is:

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

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

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

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:

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

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

```
\displaystyle
\ell_C (y; \phi) \propto \sum_{t=8}^T \log \left( \frac{1}{\sigma \sqrt{2\pi}} \exp \left{-\frac{(y_t - (c+ \phi_1y_{t-1} + ... + \phi_7 y_{t-7} ))^2}{2\sigma^2} \right} \right)
```

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.

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

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.

Mean, Covariance

This code implements the equation

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

to compute the mean adn then implement the Lyapunov equation

$$\Sigma = A \Sigma A^T + Q$$

to compute the covariance matrix of the autoregressive process.

First, the matrix A is constructed, which is a square matrix where the first row is made up of $\phi_1, \phi_2, \dots, \phi_p$ 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 $\phi_1, \phi_2, \dots, \phi_p$ 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:

```
    p = len(phis)
    A = np.zeros((p, p))
    A[0, :] = phis
    A[1:, 0:(p-1)] = np.eye(p-1)
```

Next, the vector $\vec{b}$ is constructed which is a vector with the constant c in the first position and zeros elsewhere:

```
    b = np.zeros((p, 1))
    b[0, 0] = c
```

Then the equation to find the mean is solved:

```
    I = np.eye(p)
    mu = np.linalg.inv(I - A) @ b
```

Then, the matrix Q of the Lyapunov equation is construed as a matrix with σ^2 on position [0,0] and zeros elsewhere.

```
    Q = np.zeros((p,p))
    Q[0,0] = sigma2
```

Lastly the Lyapunov equation in solved:

```
Sigma = scipy.linalg.solve_discrete_lyapunov(A, Q)
```

The entire code is below. **ravel()** is a function that flattens a multideimensional array into one dimension.

```
import numpy as np
import scipy

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

    # 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

# Example usage:
phis = [0.2, -0.1, 0.05, -0.05, 0.02, -0.02, 0.01]
c = 0
sigma2 = 0.5
```

```
mu, Sigma, stationary = unconditional_ar_mean_variance(c, phis, sigma2)
print("The process is stationary:", stationary)
print("Mean vector (mu):", mu)
print("Variance-covariance matrix (Sigma);", Sigma)
```

Conditional and Unconditional Likelihood Functions

The parameters of the model are the constant c, the coefficients $\phi_1, \ldots, \phi_7$ and the variance of the normally distributed error terms σ^2. They are passed to the function as a single list of parameters $\text{params} = (c; \phi_1, \ldots, \phi_7; \sigma^2)$.

```
c = params[0]
    phi = params[1:8]
    sigma2 = params[8]
```

The function defined above is used to retrieve the Covariance Matrix, the mean and to check whether the process is stationary:

```
mu, Sigma, stationary = unconditional_ar_mean_variance(c, phi, sigma2)
```

As described above, the conditional log-likelihood function is a sum of normally distributed variables, specifically the conditional distribution is $N(c + \phi_1 y_{t-1} + \ldots + \phi_7 y_{t-7}, \sigma^2)$.

```
loglik = np.sum(norm.logpdf(yf, loc=(c + Xf@phi), scale=np.sqrt(sigma2)))
```

The unconditional log-likelihood as stated above is the conditional log-likelihood plus the multivariate distribution of the first seven observations. In the code the parameters of the multivariate normal distribution are derived via the equation $\mu = (I - A)^{-1}b$ and the Lyapunov equation.

```
    mu, Sigma, stationary = unconditional_ar_mean_variance(c, phi, sigma2)
```

Then the multivariate normal distribution is implemented:

```
    mvn = multivariate_normal(mean=mu, cov=Sigma, allow_singular=True)
    uloglik = cloglik + mvn.logpdf(y[0:7])
```

Thus, the unconditional log-likelihood is coded as:

```
    uloglik = cloglik + mvn.logpdf(y[0:7])
```

The entire code chunk is:

```
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

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


## Example
params = np.array([
    0.0, ## c
    0.2, -0.1, 0.05, -0.05, 0.02, -0.02, 0.01, ## phi
    1.0 ## sigma2
    ])

## Fake data
y = np.random.normal(size=100)

## The conditional distribution
cond_loglikelihood_ar7(params, y)
## The unconditional distribution
uncond_loglikelihood_ar7(params, y)
```

Maximum Likelihood with Simulated Data

```
## Unconditional - define the negative loglikelihood

## Starting value.
## These estimates should be close to the OLS
X = lagged_matrix(y, 7)
yf = y[7:]
Xf = np.hstack((np.ones((93,1)), 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))
```

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

results = scipy.optimize.minimize(cobj, params, args = y, method='L-BFGS-B')

## If everything is correct, results.x should be equal to the OLS parameters.

## Not the conditional

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

bounds_constant = tuple((-np.inf, np.inf) for _ in range(1))
bounds_phi = tuple((-1, 1) for _ in range(7))
bounds_sigma = tuple((0,np.inf) for _ in range(1))
bounds = bounds_constant + bounds_phi + bounds_sigma

## L-BFGS-B support bounds
results = scipy.optimize.minimize(uobj, results.x, args = y, method='L-BFGS-B', bounds = bounds)
```

Next Steps

Next, we need to actually estimate the parameters with MLE using the INDPRO variable from the FREDMD dataset.