untitled4

April 21, 2024

[ ]:
```
#1.Coding the Likelihood Function
#First of all in this assignment we should implement the likelihood function␣
↪for an AR(7) model in Python
#to code both the conditional and unconditional likelihood functions.
```

[1]:
```
# Installing packages
import pandas as pd
from numpy.linalg import solve
import numpy as np
import scipy
```

[2]:
```
# Installing packages
from numpy import linalg as la
from scipy.optimize import approx_fprime
from scipy.optimize import minimize
from scipy.optimize import Bounds
from scipy.stats import norm
from scipy.stats import multivariate_normal
from google.colab import files
```

[ ]:
```
#CONDITIONAL LIKELIHOOD -Implementation in python
#the provided code below calculates the log-likelihood of an autoregressive␣
↪model with lag 7 (AR(7)) given a set of parameters (params) and observed␣
↪data (y). It first calculates the unconditional mean and covariance matrix␣
↪of the AR(7) process using the parameters. Then, it checks if the process is␣
↪stationary; if not, it returns negative infinity as the log-likelihood.
# If the process is stationary, it computes the conditional likelihood by␣
↪assuming that the observed data follows a multivariate normal distribution␣
↪with mean c plus a linear combination of lagged values (Xf @ phi) and␣
↪variance sigma2. Finally, it sums the log probability density function of␣
↪each observed data point given the calculated mean and variance.
```

[3]:
```
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
```

[ ]:
```
#UNCONDITIONAL LIKELIHOOD-Implementation in python
#So the provided code below in the  unconditional_ar_mean_variance function␣
 ↪calculates the mean vector and covariance matrix of an autoregressive (AR)␣
 ↪process of order 7 (AR(7)), considering parameters such as the constant term␣
 ↪(c), autoregressive coefficients (phis), and error variance (sigma2). It␣
 ↪constructs the transition matrix A from the autoregressive coefficients,␣
 ↪checks for the stationarity of the process, and computes the mean vector␣
 ↪using matrix algebra. Additionally, it solves the discrete Lyapunov equation␣
 ↪to find the covariance matrix.
#The uncond_loglikelihood_ar7 function utilizes these statistics to compute the␣
 ↪unconditional log-likelihood by combining the conditional log-likelihood of␣
 ↪observed data with the probability density function of the initial␣
 ↪observations under the AR(7) process. If the process is not stationary, it␣
 ↪returns negative infinity as the log-likelihood.
```

[4]:
```
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
```

[5]:
```
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
```

[ ]:
```
#2.Maximizing the Likelihood (INDPRO VARIABLE)
```

[24]:
```
# Setting directory to the csv file
from google.colab import drive
drive.mount('/content/drive')

SNG_Team = '/content/drive/My Drive/current.csv'

# Loading the dataframe
df = pd.read_csv(SNG_Team)
df_cleaned = df.drop(index=0)
df_cleaned.reset_index(drop=True, inplace=True)
df_cleaned['sasdate'] = pd.to_datetime(df_cleaned['sasdate'], format='%m/%d/%Y')
df_cleaned
#selecting our variable "INDPRO"
Y = df_cleaned['INDPRO']
```

```
Y
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[24]:
```
0       21.9665
1       22.3966
2       22.7193
3       23.2032
4       23.5528
         …
776    103.2096
777    102.3722
778    102.6710
779    102.6715
780    102.5739
Name: INDPRO, Length: 781, dtype: float64
```

[25]:
```
#codes implementation from assignment
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
```

[26]:
```
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
## 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
    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
# 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)
#Same Procedure for unconditional likelihood
params= np.hstack((beta, sigma2_hat))
```

[28]:

```
#3.Parameter estimation
# Define y:
y = Y

# Ordinary Least Squares:
X = lagged_matrix(y, 7)
yf = y[7:]
Xf = np.hstack((np.ones((len(yf),1)), X[7:,:]))

# Estimate the parameters and the variance:
beta = np.linalg.solve(Xf.T@Xf, Xf.T@yf)
beta        # To see the estimates
sigma2_hat = np.mean((yf - Xf@beta)**2)

# They are concatenated into a single vector:
params = np.hstack((beta, sigma2_hat))
# Negative value of the conditional log-likelihood:
def cobj(params, y):

    # Compute the value of the objective function:
    value = -cond_loglikelihood_ar7(params, y)

    # Handle invalid values:
    if np.isnan(value):
        # If the value is invalid, return a large value to indicate an error:
        return 1e12
    else:
        # Otherwise, return the computed value:
        return value

# Minimize the conditional log-likelihood using the L-BFGS-B algorithm:
results1 = scipy.optimize.minimize(cobj, params, args = y, method='L-BFGS-B')
results1

# We can see that the values of result.x are 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
results2 = scipy.optimize.minimize(uobj, results1.x, args = y,␣
↪method='L-BFGS-B', bounds = bounds)
results2
```

```
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_numdiff.py:576:
RuntimeWarning: invalid value encountered in subtract
  df = fun(x) - f0
```

[28]:

```
  message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  success: True
   status: 0
      fun: 32118.58642427295
        x: [ 1.861e-01  1.000e+00 -3.541e-01  1.538e-01  6.122e-02
            -3.235e-02  9.619e-02 -6.772e-02  1.318e+00]
      nit: 3
      jac: [-5.670e+03 -4.387e+05 -4.381e+05 -4.374e+05 -4.367e+05
            -4.361e+05 -4.354e+05 -4.348e+05 -2.345e+04]
     nfev: 50
     njev: 5
 hess_inv: <9x9 LbfgsInvHessProduct with dtype=float64>
```

[38]:

```
# 4.forecasting!
# Define the function for the AR(7) model:
def forecast_ar7(params, y):
    c = params[0]
    phi = params[1:8]
    sigma2 = params[8]
    # Create the lagged matrix (h=8):
    X_forecast = lagged_matrix(y, 7)[-8:, :]
    # Compute the forecast:
    forecast = c + np.dot(X_forecast, phi)
    return forecast

# the starting date:
start_date = '2000-01-01'
```

```
forecast_dates = pd.date_range(start=start_date, periods=8, freq='MS')

# Forecast using the parameters from the conditional approach:
forecast_conditional = forecast_ar7(results1.x, Y)

# Forecast using the parameters from the unconditional approach:
forecast_unconditional = forecast_ar7(results2.x, Y)

# Create a Dataframe for the forecasts:
forecast_df = pd.DataFrame({'Date': forecast_dates,
                            'Conditional Forecast': forecast_conditional,
                            'Unconditional Forecast': forecast_unconditional})

# View:
print(forecast_df)
```

```
        Date  Conditional Forecast  Unconditional Forecast
0 2000-01-01            102.717976               88.117999
1 2000-02-01            102.314043               87.864974
2 2000-03-01            103.551574               88.897732
3 2000-04-01            103.012569               88.388536
4 2000-05-01            103.296057               88.658018
5 2000-06-01            102.217203               87.794609
6 2000-07-01            102.836164               88.326577
7 2000-08-01            102.742699               88.228601
```

[39]:

```
from sklearn.metrics import mean_squared_error
errors_conditional = []
errors_unconditional = []
# Calculate the error for each monthly forecast:
for i in range(8):
    error_conditional = mean_squared_error([Y[i]], [forecast_conditional[i]])
    errors_conditional.append(error_conditional)

    error_unconditional = mean_squared_error([Y[i]],␣
 ↪[forecast_unconditional[i]])
    errors_unconditional.append(error_unconditional)

# Square root of the mean squared errors
rmses_conditional = [np.sqrt(error) for error in errors_conditional]
rmses_unconditional = [np.sqrt(error) for error in errors_unconditional]

print("Conditional monthly forecast MSE):", errors_conditional)
print("Conditional monthly forecast RMSE:", rmses_conditional)

print("Unconditional monthly forecast MSE):", errors_unconditional)
print("Unconditional monthly forecast RMSE:", rmses_unconditional)
```

```
Conditional monthly forecast MSE): [6520.8008919832355, 6386.797717312388,
6533.856543663836, 6369.535309977541, 6358.987024851885, 6183.856956252999,
6371.418250842376, 6481.441329996744]
Conditional monthly forecast RMSE: [80.75147609785988, 79.91744313547818,
80.83227414630765, 79.80936856019812, 79.74325692403016, 78.63750349707828,
79.82116417869622, 80.50739922514417]
Unconditional monthly forecast MSE): [4376.020838430186, 4286.1079801554215,
4379.584822773864, 4249.127979597197, 4238.689456593092, 4123.554501034424,
4265.602077390552, 4355.115734107759]
Unconditional monthly forecast RMSE: [66.15149913970345, 65.46837389270809,
66.17843170379504, 65.18533561773842, 65.10521835147388, 64.21490871312069,
65.3115769017297, 65.99330067596073]
```

[ ]:
```
#The presence of positive values in both conditional and unconditional
↪forecasts indicates an expected uptrend in industrial production, while
↪negative values imply a projected decline. Fluctuations between positive and
↪negative values suggest inherent uncertainty or volatility in the industrial
↪production outlook. Considering the pivotal role of industrial production in
↪driving economic growth, these forecasts suggest potential periods of both
↪expansion and contraction in the industrial sector over the eight-month
↪period. Various factors including interest rate changes, governmental
↪policies impacting the industrial sector, and shifts in global demand for
↪manufactured goods could contribute to fluctuations in industrial production.
```