assignment-2

April 18, 2024

[ ]:
```
import numpy as np
from scipy.stats import norm

def ar7_conditional_likelihood(params, data):
    """
    Compute the conditional likelihood for an AR(7) model.

    Args:
        params (list): List of parameters [phi1, phi2, ..., phi7, sigma^2].
        data (array): Array containing observed data.

    Returns:
        float: Log likelihood value.
    """
    phi = params[:-1]
    sigma_squared = params[-1]
    n = len(data)
    log_likelihood = 0

    # Given the first 7 observations, likelihood calculation starts from the␣
 ↪8th observation
    for t in range(7, n):
        # Compute the mean based on the preceding 7 observations
        mean = np.dot(phi, data[t-7:t][::-1])
        # Compute the log likelihood contribution of the current observation
        log_likelihood += norm.logpdf(data[t], loc=mean, scale=np.
 ↪sqrt(sigma_squared))

    return log_likelihood
def ar7_unconditional_likelihood(params, data):
    """
    Compute the unconditional likelihood for an AR(7) model.

    Args:
        params (list): List of parameters [phi1, phi2, ..., phi7, sigma^2].
        data (array): Array containing observed data.
```

```
    Returns:
        float: Log likelihood value.
    """
    phi = params[:-1]
    sigma_squared = params[-1]
    n = len(data)
    log_likelihood = 0

    # Calculate the unconditional mean of the AR(7) process
    unconditional_mean = np.dot(phi, data[:7][::-1])

    # Calculate the log likelihood for each observation using the unconditional mean
    for t in range(n):
        log_likelihood += norm.logpdf(data[t], loc=unconditional_mean, scale=np.sqrt(sigma_squared))

    return log_likelihood
# Define sample data size and generate data
data_size = 100
data = np.random.normal(size=data_size)

# Define initial parameters
initial_params = [0.5, -0.2, 0.3, -0.1, 0.2, -0.4, 0.1, 1.0]

# Calculate conditional likelihood
conditional_ll = ar7_conditional_likelihood(initial_params, data)
print("Conditional Log Likelihood:", conditional_ll)

# Calculate unconditional likelihood
unconditional_ll = ar7_unconditional_likelihood(initial_params, data)
print("Unconditional Log Likelihood:", unconditional_ll)
```

```
Conditional Log Likelihood: -155.73183874911425
Unconditional Log Likelihood: -143.1813242908849
```

[ ]:
```
from scipy.optimize import minimize

# Define functions for negative log likelihood (to be minimized)
def neg_log_likelihood_conditional(params, data):
    return -ar7_conditional_likelihood(params, data)

def neg_log_likelihood_unconditional(params, data):
    return -ar7_unconditional_likelihood(params, data)

# Example usage:
# Generate sample data
```

```
data = np.random.normal(size=100)

# Initial parameter guess
initial_params = [0.5, -0.2, 0.3, -0.1, 0.2, -0.4, 0.1, 1.0]

# Maximize conditional likelihood
result_conditional = minimize(neg_log_likelihood_conditional, initial_params,␣
 ↪args=(data,), method='Nelder-Mead')
print("Maximized Conditional Likelihood Parameters:", result_conditional.x)

# Maximize unconditional likelihood
result_unconditional = minimize(neg_log_likelihood_unconditional,␣
 ↪initial_params, args=(data,), method='Nelder-Mead')
print("Maximized Unconditional Likelihood Parameters:", result_unconditional.x)
```

```
Maximized Conditional Likelihood Parameters: [-0.03325704  0.04579174
0.00257266 -0.00571426  0.14923269 -0.01774663
 -0.07561107  0.90455498]
Maximized Unconditional Likelihood Parameters: [ 0.46637547 -0.23227134
-0.02958764 -0.09122037  0.19125784 -0.6580344
  0.12136695  0.89892826]
```

[ ]:
```
import numpy as np
import pandas as pd
from scipy.optimize import minimize

# Step 1: Data Preparation
# Load the FRED-MD dataset
data = pd.read_csv('/content/current.csv')

# Assuming 'INDPRO' is the column name for the INDPRO variable
indpro = data['INDPRO']

# Calculate monthly log differences
log_diff_indpro = np.log(indpro).diff().dropna()

# Step 2: Likelihood Functions
def ar7_conditional_likelihood(params, data):
    phi, sigma_sq = params[:7], params[7]
    n = len(data)
    ll = 0
    for t in range(7, n):
        yt = data[t]
        mu_t = np.dot(phi, data[t-7:t][::-1])  # Reverse the order to align␣
 ↪with AR(7) formula
        ll += -0.5 * (np.log(2 * np.pi * sigma_sq) + (yt - mu_t)**2 / sigma_sq)
    return ll
```

```
def ar7_unconditional_likelihood(params, data):
    phi, sigma_sq = params[:7], params[7]
    n = len(data)
    mu0 = np.mean(data[:7])  # Initial mean
    ll = -0.5 * (7 * np.log(2 * np.pi * sigma_sq) + np.sum((data[:7] - mu0)**2)
 ↪/ sigma_sq)
    for t in range(7, n):
        yt = data[t]
        mu_t = np.dot(phi, data[t-7:t][::-1])  # Reverse the order to align
 ↪with AR(7) formula
        ll += -0.5 * (np.log(2 * np.pi * sigma_sq) + (yt - mu_t)**2 / sigma_sq)
    return ll

# Step 3: Parameter Estimation
# Initial parameter guess
initial_params = np.array([0.5, -0.2, 0.3, -0.1, 0.2, -0.4, 0.1, 1.0])

# Maximize conditional likelihood
result_conditional = minimize(ar7_conditional_likelihood, initial_params,
 ↪args=(log_diff_indpro,), method='Nelder-Mead')
print("Parameters maximizing the Conditional Likelihood:", result_conditional.x)

# Maximize unconditional likelihood
result_unconditional = minimize(ar7_unconditional_likelihood, initial_params,
 ↪args=(log_diff_indpro,), method='Nelder-Mead')
print("Parameters maximizing the Unconditional Likelihood:",
 ↪result_unconditional.x)
```

```
Parameters maximizing the Conditional Likelihood: [-1.05849043e+50
1.06746350e+49 -1.73462824e+49  2.64622188e+49
  2.68764340e+49 -4.69880006e+49 -3.21491762e+49  6.97645179e+50]
Parameters maximizing the Unconditional Likelihood: [-1.05849043e+50
1.06746350e+49 -1.73462824e+49  2.64622188e+49
  2.68764340e+49 -4.69880006e+49 -3.21491762e+49  6.97645179e+50]
```

[ ]:
```
import numpy as np

# Function for forecasting future values
def forecast_ar7(params, data, steps):
    # Extract parameters
    phi = params[:7]
    sigma = params[7]

    # Initialize forecast array with observed data
    forecast = np.copy(data)
```

```
    # Perform forecasting for future values
    for _ in range(steps):
        # Calculate the next forecasted value
        next_forecast = np.dot(phi, forecast[-7:]) + np.random.normal(0, sigma)
        # Append the forecasted value to the forecast array
        forecast = np.append(forecast, next_forecast)

    return forecast

# Forecast using parameters maximizing the conditional likelihood
conditional_forecast = forecast_ar7(result_conditional.x, log_diff_indpro, 8)
print("Forecast based on Conditional Likelihood:", conditional_forecast[-8:])

# Forecast using parameters maximizing the unconditional likelihood
unconditional_forecast = forecast_ar7(result_unconditional.x, log_diff_indpro,␣
 ↪8)
print("Forecast based on Unconditional Likelihood:", unconditional_forecast[-8:
 ↪])
```

```
Forecast based on Conditional Likelihood: [-1.04356353e+051  3.35497079e+100
-1.07859547e+150  3.46759558e+199
 -1.11480341e+249  3.58400113e+298             -inf              nan]
Forecast based on Unconditional Likelihood: [ 4.98897301e+050 -1.60391372e+100
5.15645048e+149 -1.65775635e+199
  5.32955009e+248 -1.71340645e+298              inf              nan]
```