ex_3

April 6, 2025

[1]:
```
import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
```

1 Exercise 3

1.1 a) Additional moments

Expand the programs so that estimation is performed using the first four uncentered moments for the normal distribution.

[2]:
```
# Define the GMM objective function
def myfun(theta, y):
    # Sample moments
    G = np.mean([
        y - theta[0],
        y**2 - theta[0]**2 - theta[1]**2,
        y**3 - theta[0]**3 - 3*theta[0]*theta[1]**2,
        y**4 - theta[0]**4 - 6*theta[0]**2*theta[1]**2 - 3*theta[1]**4
    ], axis=1).reshape(-1, 1)

    # GMM objective function (identity weighting matrix)
    GMM = G.T @ G
    return GMM.item()  # Return as scalar
```

1.2 Simulations

[3]:
```
# Number of simulations to run
NSIM = 20

# Placeholder to store estimates from all simulations
theta_all = []

# Simulation loop
for k in range(NSIM):
    N = 200  # Sample size for each simulation

    # True population parameters
```

```
    musubzero = 5.0
    sigsubzero = 2.0

    # Generate sample data
    y = sigsubzero * np.random.randn(N) + musubzero

    # Starting values for optimization
    startvalues = [0.0, 0.01]

    # Run the optimizer (minimize the GMM objective)
    result = minimize(
        fun=myfun,
        x0=startvalues,
        args=(y,),
        method='BFGS',  # Equivalent to MATLAB's fminunc
        options={'disp': False}
    )

    thetaHAT = result.x      # Estimated parameters
    GMMvalue = result.fun    # GMM objective value at solution

    # Display results for each simulation
    print(f"Simulation {k+1}:")
    print("  thetaHAT =", thetaHAT)
    print("  GMMvalue =", GMMvalue)
    print()

    # Store the estimates in a list
    theta_all.append(thetaHAT)

# Convert list to NumPy array for further processing
theta_all = np.array(theta_all)

# Display all theta estimates
print("All theta estimates across simulations:")
print(theta_all)

# Optional: compute mean and standard deviation across simulations
theta_mean = theta_all.mean(axis=0)
theta_std = theta_all.std(axis=0)
print("\nMean of estimates:", theta_mean)
print("Standard deviation of estimates:", theta_std)

# ------------------------------
# Suggestions for Optimization:
# ------------------------------
```

```
# - Use `np.stack` if preallocating and filling an array instead of appending
↪(faster for large NSIM).
# - For large-scale simulation, consider multiprocessing to parallelize the
↪loop.
# - You can seed each run for reproducibility using `np.random.seed(seed + k)`
↪inside the loop.
```

```
Simulation 1:
  thetaHAT = [4.81242821 1.99522782]
  GMMvalue = 0.06556150852765888

Simulation 2:
  thetaHAT = [4.89444447 2.08902525]
  GMMvalue = 0.026976634799405024

Simulation 3:
  thetaHAT = [5.04155374 1.98099174]
  GMMvalue = 0.03516840100778569

Simulation 4:
  thetaHAT = [5.25252036 1.86724232]
  GMMvalue = 0.00027955196233249O4

Simulation 5:
  thetaHAT = [5.1504288 2.1998054]
  GMMvalue = 0.0039615449240816775

Simulation 6:
  thetaHAT = [5.10470086 2.1317751 ]
  GMMvalue = 0.000517245921862793

Simulation 7:
  thetaHAT = [5.11429241 1.89045257]
  GMMvalue = 0.0042932995145338


Simulation 8:
  thetaHAT = [4.94052029 2.09756007]
  GMMvalue = 0.002004089361536634

Simulation 9:
  thetaHAT = [4.84874374 2.03707892]
  GMMvalue = 0.011184879529137357

Simulation 10:
  thetaHAT = [5.00865765 1.84458535]
  GMMvalue = 0.0013571471387341988
```

```
Simulation 11:
  thetaHAT = [4.74644567 1.90685686]
  GMMvalue = 0.010187709712371052

Simulation 12:
  thetaHAT = [5.17600112 1.95415138]
  GMMvalue = 0.020454530873698928

Simulation 13:
  thetaHAT = [5.01224847 2.04824841]
  GMMvalue = 0.0011976010429938793

Simulation 14:
  thetaHAT = [5.08060617 1.8852893 ]
  GMMvalue = 0.022253400915702905

Simulation 15:
  thetaHAT = [4.7917133  2.23519791]
  GMMvalue = 0.08134869955627656

Simulation 16:
  thetaHAT = [5.00511549 2.22628863]
  GMMvalue = 0.03252648261691514

Simulation 17:
  thetaHAT = [4.85593023 2.04805931]
  GMMvalue = 0.10310997208466845


Simulation 18:
  thetaHAT = [5.37992189 1.92319755]
  GMMvalue = 0.019205642023443777

Simulation 19:
  thetaHAT = [5.10249155 2.13313819]
  GMMvalue = 0.0019544744053132663

Simulation 20:
  thetaHAT = [5.05539947 1.81290815]
  GMMvalue = 0.04076441702615428

All theta estimates across simulations:
[[4.81242821 1.99522782]
 [4.89444447 2.08902525]
 [5.04155374 1.98099174]
 [5.25252036 1.86724232]
 [5.1504288  2.1998054 ]
 [5.10470086 2.1317751 ]
```

```
 [5.11429241 1.89045257]
 [4.94052029 2.09756007]
 [4.84874374 2.03707892]
 [5.00865765 1.84458535]
 [4.74644567 1.90685686]
 [5.17600112 1.95415138]
 [5.01224847 2.04824841]
 [5.08060617 1.8852893 ]
 [4.7917133  2.23519791]
 [5.00511549 2.22628863]
 [4.85593023 2.04805931]
 [5.37992189 1.92319755]
 [5.10249155 2.13313819]
 [5.05539947 1.81290815]]

Mean of estimates: [5.01870819 2.01535401]
Standard deviation of estimates: [0.15842301 0.12634864]
```

1.3 b) Convergence

Use one of the simulation programs to demonstrate that the method of moment estimators converges at the rate $\sqrt{T}$

[4]:
```
# True population parameters
musubzero = 5.0
sigsubzero = 2.0

def simulation(N, seed=0, musubzero=musubzero, sigsubzero=sigsubzero):
    np.random.seed(seed)
    # Generate sample data
    y = sigsubzero * np.random.randn(N) + musubzero
    # Starting values for optimization
    startvalues = [0.0, 0.01]
    # Run the optimizer (minimize the GMM objective)
    result = minimize(
        fun=myfun,
        x0=startvalues,
        args=(y,),
        method='BFGS',  # Equivalent to MATLAB's fminunc
        options={'disp': False}
    )
    return result.x  # Estimated parameters
```

[5]:
```
n_max = 100000
n_step = 1000
n_min = n_step

n_values = np.arange(n_min, n_max, n_step)
```

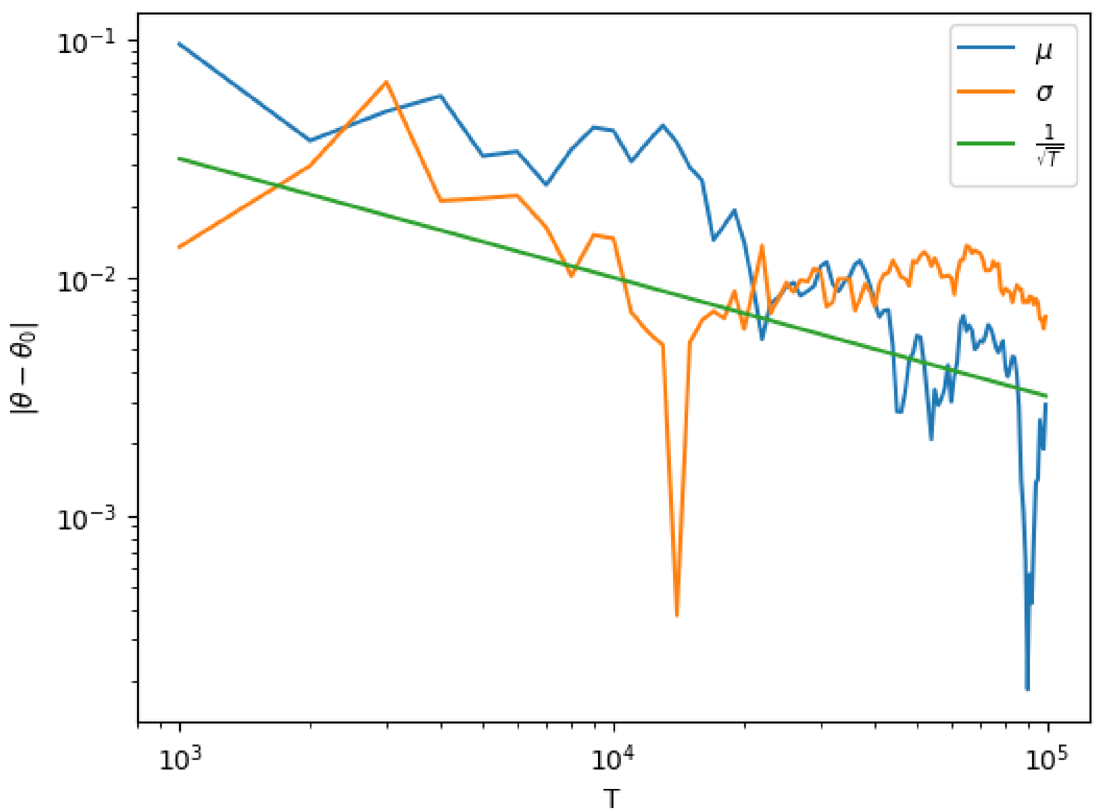
```
thetas = np.empty((len(n_values), 2))

for n in n_values:
    thetas[n // n_step-1] = simulation(n, seed=0)
```

Log-log plot of the absolute difference between the estimated and true parameters. The green line has slope 1/2.

[6]:
```
plt.loglog(n_values, np.abs(thetas[:, 0]-musubzero), label=r'$\mu$')
plt.loglog(n_values, np.abs(thetas[:, 1]-sigsubzero), label=r'$\sigma$')
plt.loglog(n_values, 1/np.sqrt(n_values), label=r'$\frac{1}{\sqrt{T}}$')

plt.xlabel('T')
plt.ylabel(r'$|\theta-\theta_0|$')
plt.legend()
plt.show()
```



1.4 2 Stage GMM (with optimal weighting matrix)

[7]:
```
class GMMEstimator:
    def __init__(self, y, startvalues):
        self.y = y
        self.N = len(y)
        self.startvalues = startvalues
        self.g = None  # Initialize g as None
        self.W = np.eye(2)  # Initial weighting matrix: identity

    def myfun(self, theta, W):
        # Compute the individual moment conditions for each observation
        self.g = np.column_stack([
            self.y - theta[0],
            self.y**2 - theta[0]**2 - theta[1]**2
        ])

        # Compute the sample moments (mean of g)
        G = np.mean(self.g, axis=0).reshape(-1, 1)

        # Compute the GMM objective function with the weighting matrix
        GMM = G.T @ W @ G
        return GMM.item()  # Return the scalar value

    def first_stage_estimation(self):
        # Perform the first-stage GMM estimation
        result_1 = minimize(
            fun=self.myfun,
            x0=self.startvalues,
            args=(self.W,),
            method='BFGS',
            options={'disp': True}
        )
        thetaHAT_1 = result_1.x
        GMMvalue_1 = result_1.fun

        print("\nFirst-stage estimates:")
        print("thetaHAT =", thetaHAT_1)
        print("GMMvalue =", GMMvalue_1)

        return thetaHAT_1

    def second_stage_estimation(self, thetaHAT_1):
        # Compute the optimal weighting matrix
        W_optimal = np.linalg.inv((self.g.T @ self.g) / self.N)

        # Perform the second-stage GMM estimation
```

```
        result_2 = minimize(
            fun=self.myfun,
            x0=self.startvalues,
            args=(W_optimal,),
            method='BFGS',
            options={'disp': True}
        )
        thetaHAT_2 = result_2.x
        GMMvalue_2 = result_2.fun

        print("\nSecond-stage estimates:")
        print("thetaHAT =", thetaHAT_2)
        print("GMMvalue =", GMMvalue_2)

        return thetaHAT_2
```

[8]:
```
# Set random seed for reproducibility
np.random.seed(12)

N = 200  # Sample size

# Population parameters
musubzero = 5.0
sigsubzero = 2.0

# Generate simulated data
y = sigsubzero * np.random.randn(N) + musubzero

# Starting values for the parameters
startvalues = [0.0, 0.01]

# Create GMM estimator object
gmm_estimator = GMMEstimator(y, startvalues)

# First-stage estimation
thetaHAT_1 = gmm_estimator.first_stage_estimation()

# Second-stage estimation
thetaHAT_2 = gmm_estimator.second_stage_estimation(thetaHAT_1);
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 17
         Function evaluations: 72
         Gradient evaluations: 24

First-stage estimates:
thetaHAT = [4.63944894 2.07368061]
```

```
GMMvalue = 2.2824147405684435e-13
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 17
         Function evaluations: 90
         Gradient evaluations: 30

Second-stage estimates:
thetaHAT = [4.63944635 2.073676  ]
GMMvalue = 1.0824782526055311e-11
```