ex_3

April 5, 2025

[1]:
```python
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]:
```python
# 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]:
```python
# 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
```

```python
    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 = [5.24642135 2.11510515]
  GMMvalue = 0.0004114623327071858

Simulation 2:
  thetaHAT = [4.83508566 2.06440932]
  GMMvalue = 0.003967436849551383

Simulation 3:
  thetaHAT = [4.88240188 2.04566202]
  GMMvalue = 0.0031507325588670332

Simulation 4:
  thetaHAT = [5.13997971 2.1452324 ]
  GMMvalue = 0.09579578892126038

Simulation 5:
  thetaHAT = [5.22682749 2.17159115]
  GMMvalue = 0.0054952452103235756

Simulation 6:
  thetaHAT = [5.19002834 1.87149352]
  GMMvalue = 0.02719598778870704

Simulation 7:
  thetaHAT = [5.32350539 2.00027538]
  GMMvalue = 0.010734988404376054

Simulation 8:
  thetaHAT = [5.05819519 2.0490189 ]
  GMMvalue = 0.0014257577050589374

Simulation 9:
  thetaHAT = [4.79770364 1.84630784]
  GMMvalue = 0.013010009141613722


Simulation 10:
  thetaHAT = [4.75286676 1.88014146]
  GMMvalue = 0.01919566469977903
```

```
Simulation 11:
  thetaHAT = [5.16584631 1.97675943]
  GMMvalue = 0.028322858391428406

Simulation 12:
  thetaHAT = [5.07717688 1.85420372]
  GMMvalue = 0.02620604985343288

Simulation 13:
  thetaHAT = [5.07341773 2.14789514]
  GMMvalue = 0.08032230426340593

Simulation 14:
  thetaHAT = [4.77216179 1.88368556]
  GMMvalue = 0.006902729480060072

Simulation 15:
  thetaHAT = [5.10962168 1.89626027]
  GMMvalue = 0.0041917120865801

Simulation 16:
  thetaHAT = [4.75324418 2.06789324]
  GMMvalue = 0.000794028100869762

Simulation 17:
  thetaHAT = [4.71752444 1.90450521]
  GMMvalue = 0.06443170670557218


Simulation 18:
  thetaHAT = [4.95952246 2.0017772 ]
  GMMvalue = 5.209589160643463e-05

Simulation 19:
  thetaHAT = [4.97585531 2.09260855]
  GMMvalue = 0.004598128761644194

Simulation 20:
  thetaHAT = [5.06548994 1.97473489]
  GMMvalue = 0.021843426942446953

All theta estimates across simulations:
[[5.24642135 2.11510515]
 [4.83508566 2.06440932]
 [4.88240188 2.04566202]
 [5.13997971 2.1452324 ]
 [5.22682749 2.17159115]
 [5.19002834 1.87149352]
```

```
 [5.32350539 2.00027538]
 [5.05819519 2.0490189 ]
 [4.79770364 1.84630784]
 [4.75286676 1.88014146]
 [5.16584631 1.97675943]
 [5.07717688 1.85420372]
 [5.07341773 2.14789514]
 [4.77216179 1.88368556]
 [5.10962168 1.89626027]
 [4.75324418 2.06789324]
 [4.71752444 1.90450521]
 [4.95952246 2.0017772 ]
 [4.97585531 2.09260855]
 [5.06548994 1.97473489]]

Mean of estimates: [5.00614381 1.99947802]
Standard deviation of estimates: [0.18277873 0.10436108]
```

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)
```

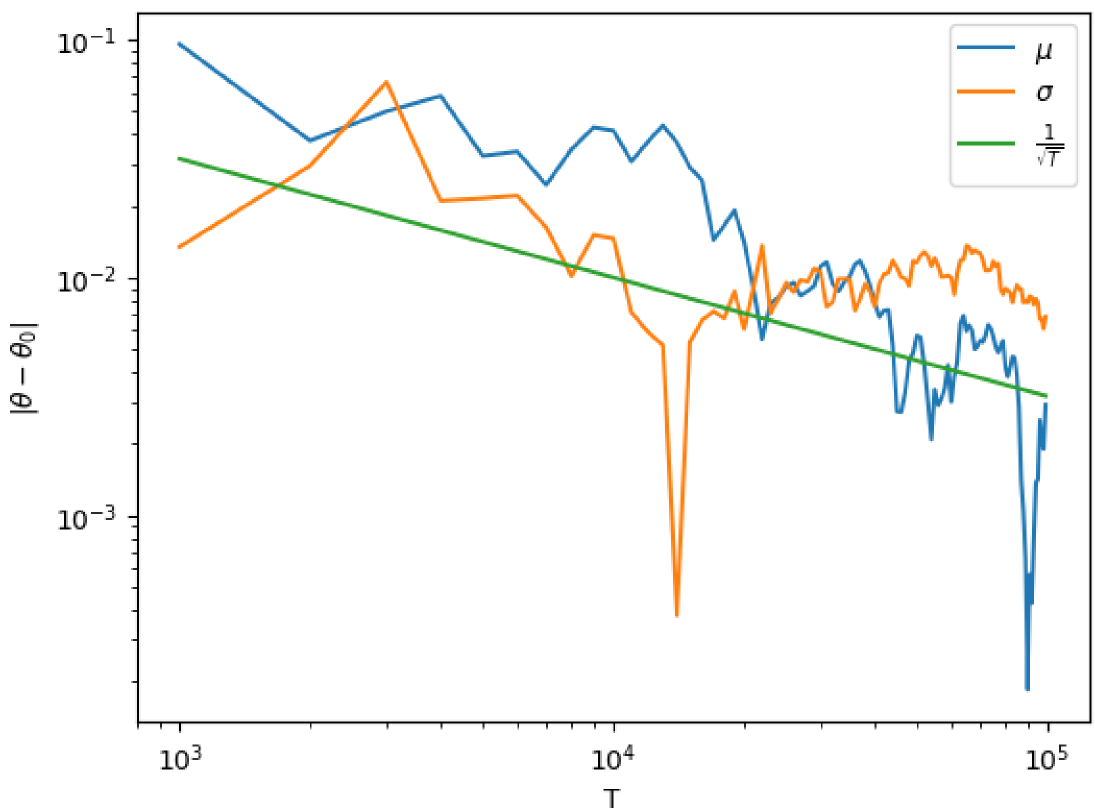
```python
thetas = np.empty((len(n_values), 2))

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]:
```python
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, N, startvalues):
        self.y = y
        self.N = N
        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, N, 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
```