05_monte_carlo

December 17, 2025

1 Monte Carlo Simulation

To verify asymptotic power and size of the three tests, we conduct a MC simulation.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import torch
import sys
from scipy.stats import norm
sys.executable

sys.path.append('../')
from utils import utils
sys.executable
```

[1]:
```
'/Users/fanghema/Desktop/aaSTAT_5200/STAT_5200_final_project/env/bin/python'
```

[2]:
```
def simulate_dgp(N, K, R, T,
                 heterogeneity_strength=0.0,
                 sigma_u=0.2,
                 sigma_eps=0.5,
                 sigma_g=1.0,
                 seed=None):
    """
    Simulates data consistent with your estimator structure.

    N: industries
    K: observed factors (Mkt, SMB, ...)
    R: latent factors in your iterative_convergence
    T: number of periods

    heterogeneity_strength = 0  → null hypothesis
    heterogeneity_strength > 0 → alternative
    """
    rng = np.random.default_rng(seed)

    # latent factors
```

```
    G = rng.normal(0, sigma_g, size=(T, R))

    # true parameters
    alpha = np.zeros(N)                  # OK under H0
    lambda_true = np.zeros((N, K))

    # Inject heterogeneity into slopes under H1
    for i in range(N):
        lambda_true[i] = heterogeneity_strength * rng.normal(0, 1, size=K)

    beta_star_true = rng.normal(0, 1, size=(N, R))

    # observed betas
    beta_true = np.zeros((N, K, T))
    for i in range(N):
        beta_i0 = rng.normal(0, 1, size=K)
        for t in range(T):
            beta_true[i,:,t] = beta_i0 + sigma_u * rng.normal(0,1,size=K)

    # realized covariance proxy: omega = beta + noise
    realized_cov = beta_true + sigma_u * rng.normal(0,1,size=(N,K,T))

    # AR(1)-style residuals for omega (simple white noise here)
    residuals = sigma_u * rng.normal(0,1,size=(N,K,T))

    # return equation
    r = np.zeros((N,T))
    for i in range(N):
        for t in range(T):
            mean_part = alpha[i] + lambda_true[i] @ beta_true[i,:,t]
            g_part    = beta_star_true[i] @ G[t]
            r[i,t] = mean_part + g_part + sigma_eps * rng.normal()

    return beta_true, r, realized_cov, residuals, G, beta_star_true
```

[3]:
```
def run_size_mc(N, K, R, T, n_rep=200, alpha=0.05):
    """
    Estimates empirical size of your tests:
    - full homogeneity
    - intercept homogeneity
    - slope homogeneity

    Simulates data under the TRUE null hypothesis (H0):
        - all lambda_i identical (here: all zeros)
        - all alpha_i identical (zeros)
    """
```

```
    reject_full = 0
    reject_alpha = 0
    reject_lambda = 0

    critical = norm.ppf(1 - alpha/2)

    for rep in range(n_rep):
        # ---- simulate DGP under H0 ----
        beta_true, r, realized_cov, residuals, G_true, beta_star_true = \
            simulate_dgp(
                N=N, K=K, R=R, T=T,
                heterogeneity_strength=0.0
            )

        # ---- estimate using your functions ----
        beta_hat = beta_true     # simulation assumes perfect estimation
        eta_hat, G_hat, beta_star_hat, _ = utils.iterative_convergence(
            beta_hat, r, N, K, R, T, n_iter=500
        )
        avar = utils.estimate_avar(
            beta_hat=beta_hat,
            excess_returns=r,
            eta=eta_hat,
            G=G_hat,
            beta_star=beta_star_hat,
            realized_covariance=realized_cov,
            residuals=residuals,
            N=N, K=K, R=R, T=T,
        )

        # ---- compute test statistics ----
        full = utils.full_homogeneity_test(eta_hat, avar, N, K, T)
        inta = utils.intercept_homogeneity_test(eta_hat, avar, N, K, T)
        slope = utils.slope_homogeneity_test(eta_hat, avar, N, K, T)

        # ---- record rejections ----
        reject_full   += (abs(full)  > critical)
        reject_alpha  += (abs(inta)  > critical)
        reject_lambda += (abs(slope) > critical)

    return {
        "size_full": reject_full / n_rep,
        "size_intercept": reject_alpha / n_rep,
        "size_slope": reject_lambda / n_rep
    }

def run_power_mc(N, K, R, T, heterogeneity_strength,
```

```
                n_rep=200, alpha=0.05):
    """
    Estimates empirical power of your tests:
         Prob(reject H0 | H1 true)

    heterogeneity_strength: magnitude of deviation from H0
    """

    reject_full = 0
    reject_alpha = 0
    reject_lambda = 0

    critical = norm.ppf(1 - alpha/2)

    for rep in range(n_rep):
        # ---- simulate under alternative ----
        beta_true, r, realized_cov, residuals, G_true, beta_star_true = \
            simulate_dgp(
                N=N, K=K, R=R, T=T,
                heterogeneity_strength=heterogeneity_strength
            )

        beta_hat = beta_true
        eta_hat, G_hat, beta_star_hat, _ = utils.iterative_convergence(
            beta_hat, r, N, K, R, T, n_iter=500
        )
        avar = utils.estimate_avar(
            beta_hat, r, eta_hat, G_hat, beta_star_hat,
            realized_cov, residuals, N, K, R, T
        )


        full  = utils.full_homogeneity_test(eta_hat, avar, N, K, T)
        inta  = utils.intercept_homogeneity_test(eta_hat, avar, N, K, T)
        slope = utils.slope_homogeneity_test(eta_hat, avar, N, K, T)

        reject_full   += (abs(full)  > critical)
        reject_alpha  += (abs(inta)  > critical)
        reject_lambda += (abs(slope) > critical)

    return {
        "power_full": reject_full / n_rep,
        "power_intercept": reject_alpha / n_rep,
        "power_slope": reject_lambda / n_rep
    }
```

[4]:
```
print("SIZE TEST RESULTS:")
print(run_size_mc(N=20, K=3, R=1, T=200, n_rep=300))
```

```
SIZE TEST RESULTS:
{'size_full': np.float64(0.9733333333333334), 'size_intercept': np.float64(1.0),
'size_slope': np.float64(0.7733333333333333)}
```

[5]:
```
for delta in [0.1, 0.2, 0.5, 1.0]:
    print(f"POWER for heterogeneity_strength={delta}:")
    print(run_power_mc(N=20, K=3, R=1, T=200,
                       heterogeneity_strength=delta,
                       n_rep=300))
```

```
POWER for heterogeneity_strength=0.1:
{'power_full': np.float64(0.5566666666666666), 'power_intercept':
np.float64(1.0), 'power_slope': np.float64(0.88)}
POWER for heterogeneity_strength=0.2:
{'power_full': np.float64(0.8166666666666667), 'power_intercept':
np.float64(1.0), 'power_slope': np.float64(0.98)}
POWER for heterogeneity_strength=0.5:
{'power_full': np.float64(1.0), 'power_intercept': np.float64(1.0),
'power_slope': np.float64(1.0)}
POWER for heterogeneity_strength=1.0:
{'power_full': np.float64(1.0), 'power_intercept': np.float64(1.0),
'power_slope': np.float64(1.0)}
```