04_avar_test_statistic

December 17, 2025

1 Computation of Asymptotic Variance and Test Statistic

After estimating time-varying β's and η's, now we estimate η's asymptotic variance and compute test statistics.

1.1 Notebook Setup

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import torch
import sys
sys.path.append('../')
from utils import utils
sys.executable
```

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

[2]:
```
data = pd.read_csv(
    '../data/processed/data_galvao.csv',
    index_col=0,
    parse_dates=True
)

factors = ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA']
assets = [col for col in data.columns if col != 'RF' and col not in factors]

beta_loading, returns_df, realized_covariance, residuals = utils.
 ↪calculate_factor_loading(
    data,
    factors=factors,
    assets=assets
)
```

[3]:
```
excess_returns = returns_df.groupby("Quarter").sum()[assets].T.values
excess_returns.shape
industries = beta_loading.index.get_level_values(0).unique().tolist()
factors = beta_loading.index.get_level_values(1).unique().tolist()
```

```
N = len(industries)
K = len(factors)
T = beta_loading.shape[1]

beta_hat_np = np.zeros((N, K, T))

for i, asset in enumerate(industries):
    for j, factor in enumerate(factors):
        beta_hat_np[i, j, :] = beta_loading.loc[(asset, factor)].values

beta_hat_np.shape
```

[3]:
```
(47, 5, 206)
```

[4]:
```
eta, G, beta_star, objective = utils.iterative_convergence(
    beta_hat_np,
    excess_returns,
    N = 47,
    K = 5,
    R = 3,
    T = 206,
    n_iter=2000
)
```

```
Iter 100/2000, frobenius_norm(G - G_old)=0.0251, objective=38.656692
Iter 200/2000, frobenius_norm(G - G_old)=0.0121, objective=38.647078
Iter 300/2000, frobenius_norm(G - G_old)=0.0045, objective=38.646437
Iter 400/2000, frobenius_norm(G - G_old)=0.0014, objective=38.646380
Iter 500/2000, frobenius_norm(G - G_old)=0.0004, objective=38.646375
Iter 600/2000, frobenius_norm(G - G_old)=0.0001, objective=38.646374
Iter 700/2000, frobenius_norm(G - G_old)=0.0000, objective=38.646374
Iter 800/2000, frobenius_norm(G - G_old)=0.0000, objective=38.646374
Iter 900/2000, frobenius_norm(G - G_old)=0.0000, objective=38.646374
Converged at iteration 955
```

1.2 Asymptotic variance estimator

[12]:
```
def estimate_avar(
    beta_hat: np.array,
    excess_returns: np.array,
    eta: np.array,
    G: np.array,
    beta_star: np.array,
    realized_covariance: np.array,
    residuals: np.array,
    N: int,
    K: int,
```

```
    R: int,
    T: int,
) -> np.array:
    """
    Estimates Avar matrix using equation (30)

    Args:
        beta_hat (np.array): estimated time-varying betas, N * K * T
        excess_returns (np.array): excess returns, N * T
        eta (np.array): estimated eta,  N * (1 + K)
        G (np.array): estimated G matrix, T * R
        beta_star (np.array): estimated beta^*, N * R
        realized_covariance (np.array): realized covariance matrix, N * K * T
        residuals (np.array): residuals from AR(1) regression, N * K * T
        N (int): number of assets
        K (int): number of observed factors
        R (int): number of unobserved factors
        T (int): number of time periods

    Returns:
        np.array: estimated asymptotic variance N(K + 1) * N(K + 1)
    """


    assert beta_hat.ndim == 3, f"beta_hat must be 3D, got {beta_hat.ndim}"
    assert beta_hat.shape == (N, K, T), f"beta_hat.shape {beta_hat.shape} !=␣
↪({N}, {K}, {T})"
    assert excess_returns.ndim == 2, f"excess_returns must be 2D, got␣
↪{excess_returns.ndim}"
    assert excess_returns.shape == (N, T), f"excess_returns.shape␣
↪{excess_returns.shape} != ({N}, {T})"
    assert eta.ndim == 2, f"eta must be 2D, got {eta.ndim}"
    assert eta.shape == (N, (1 + K)), f"eta.shape {eta.shape} != ({N}, {1 + K})"
    assert G.ndim == 2, f"G must be 2D, got {G.ndim}"
    assert G.shape == (T, R), f"G.shape {G.shape} != ({T}, {R})"
    assert beta_star.ndim == 2, f"beta_star must be 2D, got {beta_star.ndim}"
    assert beta_star.shape == (N, R), f"beta_star.shape {beta_star.shape} !=␣
↪({N}, {R})"
    assert realized_covariance.ndim == 3, f"realized_covariance must be 3D, got␣
↪{realized_covariance.ndim}"
    assert realized_covariance.shape == (N, K, T), f"realized_covariance.shape␣
↪{realized_covariance.shape} != ({T}, {R})"
    assert residuals.ndim == 3, f"residuals must be 3D, got {residuals.ndim}"
    assert residuals.shape == (N, K, T), f"residuals.shape {residuals.shape} !=␣
↪({T}, {R})"
```

```
beta_hat_t = beta_hat.transpose(0, 2, 1)           # (N, T, K)
ones = np.ones((N, T, 1))
X = np.concatenate([ones, beta_hat_t], axis=2)     # (N, T, K+1)
r = excess_returns                                 # (N, T)

# u_i = r_i - X_i eta_i
U = np.zeros((T, N))                               # (T, N)
for i in range(N):
    U[:, i] = r[i] - X[i] @ eta[i]

# projection matrix G
I_T = np.eye(T)                                    # (T, T)
GtG = G.T @ G                                      # (R, R)
M_G = I_T - G @ np.linalg.inv(GtG) @ G.T           # (T, T)

# Build block diagonal S
Kp1 = K + 1
S_hat = np.zeros((N * Kp1, N * Kp1))               # (N(K+1), N(K+1))

for i in range(N):
    Xi = X[i]                                      # (T, K+1)
    Sii = Xi.T @ M_G @ Xi / T                      # (K+1, K+1)
    r0 = i*Kp1
    S_hat[r0:r0+Kp1, r0:r0+Kp1] = Sii

# Build L
L_hat = np.zeros((N * Kp1, N * Kp1))               # (N(K+1), N(K+1))
GtG_over_N_inv = np.linalg.inv(GtG / N)

for i in range(N):
    Xi = X[i]
    for j in range(N):
        Xj = X[j]

        a_ij = beta_star[i] @ GtG_over_N_inv @ beta_star[j]
        Lij = a_ij * (Xi.T @ M_G @ Xj) / T

        r0 = i*Kp1
        c0 = j*Kp1
        L_hat[r0:r0+Kp1, c0:c0+Kp1] = Lij

# sigma2_hat
eps_sum = 0
for i in range(N):
    for t in range(T):
        pred = X[i,t] @ eta[i] + G[t] @ beta_star[i]
        eps_sum += (r[i,t] - pred)**2
```

```
df = N*T - N*Kp1 - (N+T)*R
sigma2_hat = eps_sum / df

# H tensor and Z matrices
H = np.zeros((N, T, K))

for i in range(N):
    for j in range(K):
        omega_ij = realized_covariance[i, j, :]   # (T, )
        v_ij = residuals[i, j, :]                 # (T, )
        Z_ij = np.column_stack([
            np.ones(T),
            omega_ij
        ])
        A = (1/T) * (Z_ij.T @ Z_ij)               #(2, 2)
        b = (1 / np.sqrt(T)) * (Z_ij.T @ v_ij)    #(2, )

        d_ij = np.linalg.solve(A, b)
        h_ij = Z_ij @ d_ij

        H[i, :, j] = h_ij


W_hat = np.zeros((N * Kp1, N * Kp1))

for i in range(N):
    Xi = X[i]                             # (T, K + 1)
    lam_i = eta[i, 1:]                    # (K, )
    MGXi_over_T = M_G @ Xi / T            # (T, K + 1)

    # hadamard product
    weighted_MGX = MGXi_over_T[:, 1:] * lam_i     #(T, K)

    H_i = H[i]                            #(T, K)
    H_i_T_H_i = H_i.T @ H_i               #(K, K)
    diag_H_i = np.diag(np.diag(H_i_T_H_i)) / T

    term1 = np.zeros((Kp1, Kp1))

    for j in range(K):
        wj = diag_H_i[j, j]
        col = weighted_MGX[:, j] # (T,)
        term1[1 + j, 1 + j] = wj * (col @ col)

    Sii = Xi.T @ M_G @ Xi / T
    W_i = term1 + Sii * sigma2_hat
```

```
        r0 = i * Kp1
        W_hat[r0:r0+Kp1, r0:r0+Kp1] = W_i

    # building sandwich-like estimator, say A_left W A_right
    A_left = S_hat - (1/N)*L_hat.T
    A_right = S_hat - (1/N)*L_hat
    avar = np.linalg.inv(A_left) @ W_hat @ np.linalg.inv(A_right)


    return avar
```

[13]:
```
avar = estimate_avar(
    beta_hat=beta_hat_np,
    excess_returns=excess_returns,
    eta=eta,
    G=G,
    beta_star=beta_star,
    realized_covariance=realized_covariance,
    residuals=residuals,
    N = 47,
    K = 5,
    R = 3,
    T = 206,
)
```

1.3 Test statistics

[14]:
```
def full_homogeneity_test(
    eta: np.array,
    avar: np.array,
    N: int,
    K: int,
    T: int
) -> float:
    """
    Joint hypothesis testing, under H_0
        - all intercepts are 0 AND
        - all slope coefficients equal across assets

    Returns:
        float: gamma_ad, asymptotically standard normal
    """
    Kp1 = K + 1
    p = N * Kp1
    assert avar.shape == (p, p)
    assert eta.shape == (N, Kp1)
```

```
    eta_mean = eta.mean(axis=0)
    eta_centered = eta - eta_mean
    d_vec = eta_centered.reshape(p)

    avar_inv = np.linalg.inv(avar)

    W = T * d_vec.T @ avar_inv @ d_vec
    q = (N - 1) * Kp1

    gamma_ad = (W - q) / np.sqrt(2 * q)

    return gamma_ad
```

[15]:
```
gamma = full_homogeneity_test(
    eta = eta,
    avar = avar,
    N = 47,
    K = 5,
    T = 206
)

gamma
```

[15]: np.float64(591.6508052327182)