04_avar_test_statistic

December 18, 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]
data['Quarter'] = data.index.to_period("Q")

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
)
```

1.2 Asymptotic variance estimator

[5]:
```
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,
```

```python
) -> 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)
```

```python
r = excess_returns                                  # (N, T)

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

W = np.zeros((N * Kp1, N * Kp1))
for i in range(N):
    H_i = np.zeros((T, K))
    for j in range(K):
        Z_ij = np.column_stack([np.ones(T), realized_covariance[i, j, :]])
```

```
            ZTZ_over_T = (Z_ij.T @ Z_ij) * (1.0/T)
            v_ij = residuals[i, j, :]
            h_ij = Z_ij @ np.linalg.inv(ZTZ_over_T) @ (Z_ij.T @ v_ij) / np.
↪sqrt(T)
            H_i[:, j] = h_ij

        W_i = (Xi.T @ M_G @ Xi) * (1.0 / T) * sigma2_hat

        lambda_i = eta[i][1:]
        local_vec = lambda_i * (M_G @ Xi)[:, 1:] * (1.0 / T)

        diag_HTH_over_T = np.diag(np.diag(
            H_i.T @ H_i
        )) * (1.0 / T)

        for j in range(K):
            weight = diag_HTH_over_T[j, j]
            col = local_vec[:, j]
            W_i[1 + j, 1 + j] += weight * (col @ col)

        W[
            i * (K + 1): (i + 1)* (K + 1),
            i * (K + 1): (i + 1)* (K + 1)
        ] = W_i

    avar = (
        np.linalg.inv(
            S_hat - L_hat.T / N
        )
        @ W
        @ np.linalg.inv(
            S_hat - L_hat / N
        )
    )
    return avar
```

[6]:
```
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,
)
```

[7]:
```
print(avar.min())
print(avar.max())
np.percentile(avar, [5, 50, 95])

avar = np.clip(
    avar,
    a_min = np.percentile(avar, 5),
    a_max = np.percentile(avar, 95),
)
print(avar.min())
print(avar.max())
```

```
-89792.78726193552
853444.9656474078
-120.48651836244906
121.01349228310349
```

1.3 Test statistics

[8]:
```
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
    """
    p = N * (K + 1)
    assert avar.shape == (p, p)
    assert eta.shape == (N, (K + 1))

    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
```

```python
    q = (N - 1) * (K + 1)

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

    return gamma_ad


def intercept_homogeneity_test(
    eta: np.array,
    avar: np.array,
    N: int,
    K: int,
    T: int
) -> float:
    """
    Under H_0
        - all intercepts are 0 AND

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

    alpha = eta[:, 0]

    idx = np.arange(0, p, K + 1)

    avar_inv = np.linalg.inv(avar)
    V_alpha = avar_inv[np.ix_(idx, idx)]

    W = alpha.T @ V_alpha @ alpha

    gamma_a = (W - (N-1)*K) / np.sqrt(2 * (N-1)*K)

    return gamma_a

def slope_homogeneity_test(
    eta: np.array,
    avar: np.array,
    N: int,
    K: int,
    T: int
) -> float:
    """
    Under H_0
```

```python
        - all slopes are equal

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

    slopes = eta[:,1:]
    slopes_centered = slopes - slopes.mean(axis = 0)
    slopes_vec = slopes_centered.reshape(N * K)

    avar_inv = np.linalg.inv(avar)
    idx = []
    for i in range(N):
        for j in range(K):
            idx.append(i * (K + 1) + 1 + j)
    idx = np.array(idx)

    V_lambda = avar_inv[np.ix_(idx, idx)]

    W = slopes_vec.T @ V_lambda @ slopes_vec

    q = (N - 1) * K

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

    return gamma_lambda
```

[10]:

```python
gamma_a_lambda = full_homogeneity_test(
    eta = eta,
    avar = avar,
    N = 47,
    K = 5,
    T = 206
)

gamma_a = intercept_homogeneity_test(
    eta = eta,
    avar = avar,
    N = 47,
    K = 5,
    T = 206
)

gamma_lambda = slope_homogeneity_test(
```

```python
    eta = eta,
    avar = avar,
    N = 47,
    K = 5,
    T = 206
)

print(gamma_a_lambda)
print(gamma_a)
print(gamma_lambda)
```

```
11.462892915977163
-10.728115247044245
-10.60388853172224
```