1 Introduction

1.1 Background

This project seeks to recreate parts of the analysis done in Tests of asset pricing with time-varying factor loads (Galvao et al, 2018). Galvao et al estimated factor risk premia using time-series regressions of excess returns against factor loadings. The authors proposed a Wald-like test statistic that is claimed to be adjusted for the presence of generated regressors in the time-series regression. Using this test statistic, the author tests for the correctness of asset pricing models by testing the joint hypothesis of a zero intercept and homogeneous factor risk premia across risky assets.

When applied to US-industry portfolios over the time period from 1963 to 2024, the paper found overwhelming evidence rejecting CAPM, Fama-French 3 factors and Fama-French 5 factors.

In this project, the same approach was taken with an extended dataset covering US-industry portfolios from 1963 to 2025. Overwhelming evidence was found against the null hypotheses of zero intercept and homogenous risk premia when tested separately and jointly. However, under MC simulations, this project was unable to verify the asymptotic level of the proposed tests, and the proposed tests seemed to reject the null too often. This project also makes an attempt at jackknife resampling to estimate standard error. This led to small improvements in Type I error rate but was nonetheless unsuccessful. The failure of jackknife resampling was likely because of the naive treatment of time-series panel data.

1.2 Original methodology

Using US-industry portfolio returns and Fama-French factor returns, Galvao et al's first stage involves estimating factor-loads. The factor loads are generated as the fitted values in a AR(1) regression of a time-series of realized covariances between asset and factor returns.

Following the estimation of factor loads, the second stage assumes the presence of some known number R of unobservable common factors. Galvao et al estimate factor risk premia, unobservable factor returns, as well as loading on unobservable factors jointly by solving a constrained minimization problem.

Galvao et al then describes an asymptotic variance estimator for the estimated factor risk premia, and derives 3 test statistics to test for zero-intercept, homogenous risk premia, and the

joint null hypothesis.

1.3 Replication methodology

In replicating this study, this project seeks to 1) extend dataset with data from 1963-2025; 2) implement estimation of time-varying factor loads and verify the implementation with intermediate data provided by Galvao et al; 3) implement Galvao et al's factor risk premia estimator via two different approaches, a penalty-based unconstrained optimization approach and Galvao et al's iterative convergence approach; 4) implement estimation of asymptotic variance of the above estimator 5) implement calculation of test statistics and verify asymptotic level and power and 6) conduct empirical homogeneity tests on the extended dataset.

2 Data

2.1 Original dataset

Galvao et al published an anonymized replication dataset with 12965 rows and 53 columns. From their description, this dataset is created from Fama-French's 49-US industry portfolios (daily) data, and Fama-French's 5 factor data, with Healthcare and Software industries removed due to data availability issues. Their replication data covers the period from July 1963 to December 2014.

2.2 Replication dataset

To extend the dataset, this project retrieved updated data from Kenneth French's data library. Since Galvao et al's dataset contained no indices or column names, Appendix 1 outlines the steps taken to align the original and replication dataset. The replication set contains 15,690 daily returns across 47 industry portfolios, 5 factors and the risk free rate. This yields 15,690 daily observations to estimate factor-loadings at quarterly frequency, and 206 quarterly observations for estimating factor risk premia.

3 Models

As a starting point, Galvao et al assumes asset pricing models of the form

$$r^e_{i,t+1} = \alpha_i + \beta_{it}^\top f_{t+1} + \phi_{it}^\top h_{t+1} + e_{i,t+1} \tag{1}$$

where $r^e_{i,t+1}$ is the excess return on asset i, $f_{t+1} \in \mathbb{R}^K$ is a realized vector of known systematic risk factors, $h_{t+1} \in \mathbb{R}^R$ is a realized vector of unknown factors. $\beta_{it} \in \mathbb{R}^K$ are the factor loadings on the known factors and ϕ_{it} are the loadings on the unknown factors, and $e_{i,t+1}$ satisfies $\mathbb{E}\left[e_{i,t+1}|f_{t+1}, h_{t+1}\right] = 0$.

Using the notation that $\tilde{f}_{t+1} = f_{t+1} - \mathbb{E}\left[f_{t+1}\right], \tilde{h}_{t+1} = h_{t+1} - \mathbb{E}\left[h_{t+1}\right]$, $\tilde{g}_{t+1} = (\tilde{f}_{t+1}^\top, \tilde{h}_{t+1}^\top)^\top, \beta_i^{*\top} = \left(\beta_i^\top, \phi_i^\top\right)$, and let $\lambda_{i,t+1} \in \mathbb{R}^K$ denote a vector of factor risk premia capturing the price of risk f_{t+1} for asset i, Galvao et al describes the time-series regression model

$$r^e_{i,t+1} = \alpha_i + \beta_{it}^\top \lambda_i + v_{i,t+1} \tag{2}$$

$$v^e_{i,t+1} = \beta_{it}^{*\top} \tilde{g}_{it} + \epsilon_{i,t+1} \tag{3}$$

3.1 Estimating time-varying factor loadings

To estimate the panel data of λ_{it}, the Galvao et al first generates the necessary regressors β_{it}. Let $\triangle$ denote sampling frequency and $m = 1/\triangle$ denote number of obsevations per period t, let intra-period returns from $t+h\triangle$ to $t+(h+1)\triangle$ be $R_{t+(h+1)\triangle} = p_{t+(h+1)\triangle} - p_{t+h\triangle}$, $h = 0, \ldots m-1$, and let inter-period return be $R_{t+1} = \sum_{h=0}^{m-1} R_{t+(h+1)\triangle}$. For each time step, the author stacks N risky asset returns and K factor returns in a $(N+K)$ vector denoted $R_{t+1} = (r^e_{1,t+1}, \ldots r^e_{N,t+1}, f_{1,t+1}, \ldots f_{K,t+1})$. The author denotes a realized covariance matrix as

$$\hat{\Omega}_{t+1} = \sum_{h=0}^{m-1} R_{t+(h+1)\triangle} R_{t+(h+1)\triangle}^\top$$

More concretely, the realized convariance matrices form a $(N+K) \times (N+K) \times T$ time-series.

Under a series of assumptions about the stochastic process that generates prices, the authors propose the following autoregressive process for each element of the realized covariacne matrix.

Using $\omega_{(i,j),t+1}$ to denote the entry corresponding to the i-th asset and j-th factor

$$\omega_{(i,j),t+1} = \delta_{ij,0} + \delta_{ij,1}\omega_{(i,j),t} + v_{ij,t+1} \tag{4}$$

The authors then show that a consistent estimator of the time series $\beta_{ij,t}$ is

$$\hat{\beta}_{ij,t} = \hat{\delta}_{ij,0} + \hat{\delta}_{ij,1}\omega_{(i,j),t} \tag{5}$$

Where $\hat{\delta}_{ij,0}, \hat{\delta}_{ij,1}$ are the OLS estimates in Equation (4).

3.2 Estimation of factor risk premia

The authors adopt the notation $\hat{X}_{it} = \left(1, \beta_{it}^{\top}\right)^{\top}$ and $\eta_i = (\alpha_i, \lambda_i^{\top})^{\top}$, and fit the following model

$$r_{i,t+1}^{e} = \hat{X}_{it}\eta_i \tag{6}$$

$$r_{i,t+1}^{e} = {\beta_{it}^{*}}^{\top}\tilde{g}_{it} + \epsilon_{i,t+1} \tag{7}$$

The authors propose that the estimators are

$$\hat{\eta}_i, \hat{\beta}_i^{*}, \hat{\tilde{g}}_{t+1} = \arg\min_{\eta,\beta^{*},\tilde{g}} l\left(\eta_i, \beta_i^{*}, \tilde{g}_{t+1}\right) \tag{8}$$

$$= \arg\min_{\eta,\beta^{*},\tilde{g}} \sum_{i=1}^{N}\sum_{i=1}^{T}\left(r_{i,t+1}^{e} - \hat{X}_{it}\eta_i - {\beta_i^{*}}^{\top}\tilde{g}_{t+1}\right)^2 \tag{9}$$

Subject to $\frac{G^{\top}G}{N} = I$, $\frac{{\beta^{*}}^{\top}\beta^{*}}{N}$ diagonal, where $G = \left(\tilde{g}_1, \ldots \tilde{g}_T\right)^{\top} \in \mathbb{R}^{T\times R}$, $\beta^{8} = \left(\beta_1^{*}, \ldots \beta_N^{*}\right)^{\top} \in \mathbb{R}^{N\times R}$. The authors argue that the solutions to this minimization problem should simultaneously solve

$$\hat{\eta}_i = \left(\hat{X}_i^{\top} M_{\hat{G}} \hat{X}_i\right)^{-1} \hat{X}_i^{\top} M_{\hat{G}} r_i^{e} \tag{10}$$

$$\left[\frac{1}{NT}\sum_{i=1}^{N}\left(r_i^{e} - \hat{X}_i\hat{\eta}_i\right)\left(r_i^{e} - \hat{X}_i\hat{\eta}_i\right)^{\top}\right]\hat{G} = \hat{G}\hat{V}_{NT} \tag{11}$$

Where $\hat{G}$ is the estimate of G consisting of $\hat{\tilde{g}}$, $M_{\hat{G}} = I - \hat{G}\left(\hat{G}^{\top}\hat{G}\right)^{-1}\hat{G}^{\top}$, and $\hat{V}_{NT}$ is a diagonal matrix of the R largest eigenvalues of $\hat{G}$. The authors propose solving this system by iterating Equation (10) and (11) until convergence.

As an extension to the original study and for ease of computation, this project proposes an

alternative solution to the system by solving the following unconstrained minization for some fixed φ

$$\hat{\eta}_i, \hat{\beta}_i^*, \hat{\tilde{g}}_{t+1} = \arg\min_{\eta,\beta^*,\tilde{g}} \sum_{i=1}^{N}\sum_{i=1}^{T} \left(r_{i,t+1}^e - \hat{X}_{it}\eta_i - {\beta_i^*}^\top \tilde{g}_{t+1}\right)^2 - \varphi \| \frac{G^\top G}{T} - I \|_F^2 \tag{12}$$

3.3 Estimation of asymptotic variance

The author argues that a consistent estimator of the asymptotic variance of $\sqrt{T}\,(\hat{\eta} - \eta)$ is

$$\widehat{\text{Avar}}\Big(\sqrt{T}\,(\hat{\eta} - \eta)\Big) = \left(\hat{S} - \frac{1}{N}\hat{L}^\top\right)^{-1} \hat{W} \left(\hat{S} - \frac{1}{N}\hat{L}\right)^{-1}, \tag{13}$$

where:

- $\hat{\eta} = (\hat{\eta}_1^\top, \ldots, \hat{\eta}_N^\top)^\top \in \mathbb{R}^{N(K+1)}$ stacks the individual parameter vectors $\hat{\eta}_i = (\hat{\alpha}_i, \hat{\lambda}_i^\top)^\top$.

  $\hat{X}_i = (\hat{X}_{i1}, \ldots, \hat{X}_{iT})^\top$, where $\hat{X}_{it} = (1, \hat{\beta}_{it}^\top)^\top$. Let $M_{\hat{G}} = I_T - \hat{G}(\hat{G}^\top\hat{G})^{-1}\hat{G}^\top$ denote the projection matrix onto the orthogonal complement of the space spanned by the estimated latent factors $\hat{G} = (\hat{\tilde{g}}_1, \ldots, \hat{\tilde{g}}_T)^\top \in \mathbb{R}^{T\times R}$.

- $\hat{S}$ is an $N(K+1) \times N(K+1)$ block-diagonal matrix with i-th diagonal block

$$\hat{S}_{ii} = \frac{1}{T}\,\hat{X}_i^\top M_{\hat{G}} \hat{X}_i \quad \text{for } i = 1, \ldots, N. \tag{14}$$

  Collecting these blocks gives $\hat{S} = \text{diag}(\hat{S}_{11}, \ldots, \hat{S}_{NN})$.

- $\hat{L}$ is an $N(K+1) \times N(K+1)$ matrix with (i,j) block

$$\hat{L}_{ij} = \hat{a}_{ij}\,\frac{1}{T}\,\hat{X}_i^\top M_{\hat{G}} \hat{X}_j, \qquad \hat{a}_{ij} = (\hat{\beta}_i^*)^\top \left(\frac{\hat{G}^\top\hat{G}}{N}\right)^{-1} \hat{\beta}_j^*, \tag{15}$$

  where $\hat{\beta}^* = ({\hat{\beta}_1^*}^\top, \ldots, {\hat{\beta}_N^*}^\top)^\top \in \mathbb{R}^{N\times R}$ collects the loadings on the latent factors.

- $\hat{W}$ is an $N(K+1)\times N(K+1)$ block-diagonal matrix $\hat{W} = \text{diag}(\hat{W}_1, \ldots, \hat{W}_N)$. For each asset i,

$$\hat{W}_i = \left(\hat{\lambda}_i \odot \frac{M_{\hat{G}}\hat{X}_i}{T}\right)^\top T^{-1}\text{diag}\big(\hat{H}_i^\top \hat{H}_i\big) \left(\hat{\lambda}_i \odot \frac{M_{\hat{G}}\hat{X}_i}{T}\right) + \left(\frac{1}{T}\,\hat{X}_i^\top M_{\hat{G}}\hat{X}_i\right)\hat{\sigma}^2. \tag{16}$$

Here $\hat{\lambda}_i \in \mathbb{R}^K$ is the vector of estimated factor risk premia for asset i, and $\odot$ denotes the element-wise product between $\hat{\lambda}_i$ and the K slope columns of $M_{\hat{G}}\hat{X}_i/T$.

- $\hat{H}_i$ is a $T \times K$ matrix

$$\hat{Z}_{ij} = \big(1,\, \hat{\omega}_{ij}\big) \in \mathbb{R}^{T\times 2}, \tag{17}$$

$$\hat{d}_{ij} = \left(\frac{1}{T}\hat{Z}_{ij}^{\top}\hat{Z}_{ij}\right)^{-1}\left(\frac{1}{\sqrt{T}}\hat{Z}_{ij}^{\top}\hat{v}_{ij}\right), \tag{18}$$

$$\hat{h}_{ij} = \hat{Z}_{ij}\hat{d}_{ij} \in \mathbb{R}^{T}, \tag{19}$$

and then set $\hat{H}_i = (\hat{h}_{i1}, \ldots, \hat{h}_{iK})$.

- $\hat{\sigma}^2$ is the sample variance of the pricing error $\hat{\epsilon}_{i,t+1}$, obtained from the regression

$$\hat{\sigma^2} = \frac{\sum_{t=1}^{T}\sum_{i=1}^{N}\Big(r^{e}_{i,t+1} - \hat{X}_{it}\hat{\eta}_i - \beta_i^{*\top}\hat{\tilde{g}}_{[}t+1]\Big)^2}{NT - N(K+1) - (N+T)R}$$

appropriate degrees of freedom.

3.4 Test statistics

Using the asymptotic variance estimator, Galvão et al. propose the following Wald-type test statistics for assessing the homogeneity of intercepts and slope parameters across assets.

Let $\hat{\eta}_i = (\hat{\alpha}_i, \hat{\lambda}_i^{\top})^{\top}$, $\hat{\eta}_{\cdot} = \frac{1}{N}\sum_{i=1}^{N}\hat{\eta}_i$, $\tilde{\eta} = (\hat{\eta}_1 - \hat{\eta}_{\cdot},\ \ldots,\ \hat{\eta}_N - \hat{\eta}_{\cdot}) \in \mathbb{R}^{N(K+1)}$, and $\widehat{V}^{-1} = \widehat{\mathrm{Avar}}\Big(\sqrt{T}(\hat{\eta} - \eta)\Big)^{-1}$ denote the inverse of the asymptotic variance estimator defined previously.

Joint intercept and slope homogeneity

To test the joint null hypothesis

$$H_0^{\alpha,\lambda}: \quad \alpha_1 = \cdots = \alpha_N = 0, \qquad \lambda_1 = \cdots = \lambda_N,$$

the proposed statistic is

$$\hat{\Gamma}_{\alpha,\lambda} = \frac{T\,\tilde{\eta}^{\top}\widehat{V}^{-1}\tilde{\eta} \;-\; \big[(N-1)K + N\big]}{\sqrt{2\big[(N-1)K + N\big]}}, \tag{33}$$

which satisfies $\hat{\Gamma}_{\alpha,\lambda} \xrightarrow{d} N(0,1)$ under $H_0^{\alpha,\lambda}$.

Intercept homogeneity

For the null hypothesis

$$H_0^{\alpha}: \quad \alpha_1 = \cdots = \alpha_N = 0,$$

let $\tilde{\alpha}$ denote the subvector of $\tilde{\eta}$ containing only the intercept parameters, and let $\widehat{V}_{\alpha}^{-1}$ be the corresponding submatrix of $\widehat{V}^{-1}$. The test statistic is

$$\hat{\Gamma}_{\alpha} = \frac{T\,\tilde{\alpha}^{\top}\widehat{V}_{\alpha}^{-1}\tilde{\alpha} \;-\; N}{\sqrt{2N}}, \tag{34}$$

which is asymptotically standard normal under H_0^{α}.

Slope homogeneity

For the null of slope homogeneity,

$$H_0^{\lambda}: \quad \lambda_1 = \cdots = \lambda_N,$$

let $\tilde{\lambda}$ be the slope subvector of $\tilde{\eta}$ and let $\widehat{V}_{\lambda}^{-1}$ denote the corresponding submatrix of $\widehat{V}^{-1}$. The Wald statistic is

$$\hat{\Gamma}_{\lambda} = \frac{T\,\tilde{\lambda}^{\top}\widehat{V}_{\lambda}^{-1}\tilde{\lambda} \;-\; (N-1)K}{\sqrt{2(N-1)K}}, \tag{35}$$

which satisfies $\hat{\Gamma}_{\lambda} \xrightarrow{d} N(0,1)$ under H_0^{λ}.

4 Results

4.1 Estimating factor loading

Appendix 2 contains the replication code for estimating factor loadings based on OLS estimates of the AR(1) equations as per Equation (5). Comparing our beta estimates against intermediate data provided by Galvao et al, we verify our replication implementation.

4.2 Solving for factor risk premia

We compare the two ways of estimating factor risk premia, with $\hat{\eta}_{pen}$ denoting the unconstrained penalty-based version and $\hat{\eta}_{iter}$ denoting the iterative approach discussed by Galvao et al. Appendix 3 shows that when applied to the empirical dataset, both approaches converge to the same objective value.

We also conduct MC simulations and find that both methods of computing the estimator yields similar RMSE, and that the estimators generally converge to the same value.

Table 1: Monte Carlo Results: Difference Between Estimators

	Mean	Std
$\|\hat{\eta}_{\text{pen}} - \hat{\eta}_{\text{iter}}\|$	0.135	0.063

Table 2: Monte Carlo Results: RMSE of Estimators Relative to True η

	$\hat{\eta}_{\text{pen}}$	$\hat{\eta}_{\text{iter}}$
RMSE	0.215	0.233

4.3 Empirical results

Following Galvao et al, we compute the test statistic for $K \in \{1, 3, 5\}$ corresponding to CAPM, Fama-French 3 factor and Fama-French 5 factor asset pricing models. We also let R vary for $R \in \{1, 2, 5\}$. We compute test statistics over 20 year periods as well as the full sample period.

Similar to Galvao et al, the test statistics present overwhelming evidence against the null hypotheses of a zero-intercept and homogenous slopes.

4.4 Asymptotic level and power of proposed tests

The test statistics appear very large in magnitude. While this tracks with Galvao et al's findings, this observation raises questions about the asymptotic distribution of the proposed test statistics.

Since the authors did not publish results of a MC study of the behavior of test statistics, we extend the analysis by simulating the null world to test for asymptotic levels of the proposed test. We also simulate the alternative hypothesis under different degrees of heterogeneity to test for asymptotic power.

Table 3: Joint Homogeneity Test ($\Gamma_{\alpha,\lambda}$) with p-values

		K=1			K=3			K=5		
Period	Type	R=1	R=2	R=5	R=1	R=2	R=5	R=1	R=2	R = 5
1963–1983	γ	-19.45	88.12	-54.07	-5.32	-5.49	80.83	-318.54	-7.21	1.33
	p	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.18
1973–1993	γ	38.51	-42.37	54.92	-10.30	-8.93	-8.79	-10.78	-11.08	-7.92
	p	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1983–2003	γ	-29.99	-66.68	-110.75	-2.41	28.53	160.97	-11.42	-9.91	3.28
	p	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00
1993–2013	γ	-11.01	-52.80	23007.89	-0.42	-5.06	20.58	-5.14	-1.92	39.09
	p	0.00	0.00	0.00	0.68	0.00	0.00	0.00	0.05	0.00
2003–2023	γ	24.69	-4.79	2327.37	-3.49	6.25	190.73	8.60	0.34	1.24
	p	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.74	0.21
1963–2025	γ	44.02	-2.67	2438.24	-8.92	-6.18	25.32	-6.99	-8.44	-9.92
	p	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 4: Intercept Homogeneity Test (Γ_{α}) with p-values

		K=1			K=3			K=5		
Period	Type	R=1	R=2	R=5	R=1	R=2	R=5	R=1	R=2	R = 5
1963–1983	γ	-5.29	0.49	-2.33	-8.28	-8.32	-8.14	-10.74	-10.73	-10.59
	p	0.00	0.63	0.02	0.00	0.00	0.00	0.00	0.00	0.00
1973–1993	γ	-4.78	-4.86	-4.62	-8.31	-8.31	-8.31	-10.72	-10.72	-10.72
	p	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1983–2003	γ	-4.86	-4.30	-0.64	-8.30	-7.92	-5.57	-10.72	-10.72	-10.56
	p	0.00	0.00	0.52	0.00	0.00	0.00	0.00	0.00	0.00
1993–2013	γ	-5.29	-6.25	-4.42	-8.32	-8.31	-8.31	-10.73	-10.72	-10.72
	p	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2003–2023	γ	-3.75	-5.20	-4.63	-8.41	-8.30	-8.28	-10.72	-10.72	-10.72
	p	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1963–2025	γ	-4.70	-5.34	-4.41	-8.34	-8.33	-8.30	-10.73	-10.72	-10.71
	p	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 5: Slope Homogeneity Test (Γ_λ) with p-values

		K=1			K=3			K=5		
Period	Type	R=1	R=2	R=5	R=1	R=2	R=5	R=1	R=2	R = 5
1963–1983	γ	-4.82	-2.31	-2.93	-8.28	-8.23	-5.36	-14.80	-10.68	-10.59
	p	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1973–1993	γ	-4.79	-4.80	-4.38	-8.30	-8.30	-8.29	-10.71	-10.71	-10.67
	p	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1983–2003	γ	-4.80	-5.39	-3.27	-8.32	-7.48	-8.02	-10.71	-10.70	-10.59
	p	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1993–2013	γ	-4.74	-5.06	405.04	-8.19	-8.21	-7.91	-10.63	-10.46	-10.03
	p	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2003–2023	γ	-4.30	-4.77	52.71	-8.19	-8.12	-5.35	-10.44	-10.58	-10.54
	p	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1963–2025	γ	-4.79	-4.80	10.28	-8.32	-8.30	-8.15	-10.71	-10.69	-10.71
	p	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

The procedure taken for the MC study is presented in Appendix 6. As suspected, the asymptotic level of the proposed tests are $.996, .942$ and $.912$ respectively for $\hat{\Gamma}_{\alpha,\lambda}, \hat{\Gamma}_\alpha, \hat{\Gamma}_\lambda$ respectively. This aligns with our suspicion that all test statistics are large in magnitude, and the test rejects too often.

The distribution of test statistics under the null shows a general bell shape but with extreme outliers on both ends. The individual standardized parameter estimates of $\hat{\eta}$ again follows a bell shape but with much greater variability than standard normal.

Conducting a sanity check on the estimated values of asymptotic variance, we see that the estimated asymptotic variance contains values that are extremely small. Hence the inverse covariance matrix tends to blow up the Wald-type test statistic.

4.5 Jackknife-based approach

Although time-series analysis has not been covered in STAT5200 yet, I attempt a jackknife resampling to correct for the *small* asymptotic variance that results in inflated test statistics.

Let

$$\hat{\eta} = \begin{pmatrix} \hat{\eta}_1^\top \\ \vdots \\ \hat{\eta}_N^\top \end{pmatrix} \in \mathbb{R}^{N(K+1)}, \qquad \hat{\eta}_i = \begin{pmatrix} \hat{\alpha}_i \\ \hat{\lambda}_i \end{pmatrix} \in \mathbb{R}^{K+1},$$

denote the full-sample estimator obtained from the iterative procedure in Section 3, we consider a

leave-one-period-out jackknife over time. For each $t = 1, \ldots, T$, we drop period t from the sample, re-estimate the model using the remaining $T - 1$ periods, and obtain a jackknife replicate

$$\hat{\eta}^{(-t)} = \begin{pmatrix} \hat{\eta}_1^{(-t)\top} \\ \vdots \\ \hat{\eta}_N^{(-t)\top} \end{pmatrix} \in \mathbb{R}^{N(K+1)}$$

The jackknife mean of the parameter vector is $\bar{\theta}^{(\cdot)} = \frac{1}{T}\sum_{t=1}^{T} \hat{\theta}^{(-t)}$, where θ denotes a flattened version of $\hat{\eta}$, and the (leave-one-out) jackknife covariance estimator of $\hat{\theta}$ is

$$\widehat{V}_{\eta}^{\mathrm{JK}} = \frac{T-1}{T} \sum_{t=1}^{T} \big(\hat{\theta}^{(-t)} - \bar{\theta}^{(\cdot)}\big)\big(\hat{\theta}^{(-t)} - \bar{\theta}^{(\cdot)}\big)^{\top} \in \mathbb{R}^{p \times p}.$$

Using this jackknife covariance matrix, we form a Wald-type statistic for the full homogeneity hypothesis,

$$H_0^{(\alpha,\lambda)} : \quad \alpha_1 = \cdots = \alpha_N, \quad \lambda_1 = \cdots = \lambda_N.$$

Let

$$\bar{\eta} = \frac{1}{N}\sum_{i=1}^{N} \hat{\eta}_i \quad \text{and} \quad D = \begin{pmatrix} \hat{\eta}_1 - \bar{\eta} \\ \vdots \\ \hat{\eta}_N - \bar{\eta} \end{pmatrix}, \qquad d = \mathrm{vec}(D) \in \mathbb{R}^p,$$

so that d collects the cross-sectional deviations of $(\hat{\alpha}_i, \hat{\lambda}_i)$ from their mean. The jackknife-based Wald statistic is then

$$W_{\alpha,\lambda}^{\mathrm{JK}} = d^{\top}\big(\widehat{V}_{\eta}^{\mathrm{JK}}\big)^{-1} d.$$

$$\hat{\Gamma}_{\alpha,\lambda}^{\mathrm{JK}} = \frac{W_{\alpha,\lambda}^{\mathrm{JK}} - q_{\alpha,\lambda}}{\sqrt{2q_{\alpha,\lambda}}}, \qquad q_{\alpha,\lambda} = (N-1)(K+1)$$

However, this approach also seems to run into the problem of yielding inflated test statistics. Under a MC simulation, the Type I error rate of the joint hypothesis test, intercept test, and slopes test are $99.5\%, 10.5\%$, and 61% respectively. Furthermore, I am unsure if it this approach to resampling makes economic sense under a time-series context. Observing the distribution of the test statistic under both H_0 and H_1 also shows extreme outliers.

5 Conclusion

Taken at face value, this replication study seems to agree with the original paper and overwhelmingly rejects the null hypotheses of zero intercept and homogenous slopes in CAPM, Fama-French 3 factors and Fama-French 5 factors asset pricing models.

This replication builds upon the original study with an extended dataset, and a more stable way of solving a constrained optimization problem for the estimation of factor risk premia.

The replication implementation of estimating time-varying factor loads and factor risk premia are verified by matching against the author's intermediate datasets (whenever available) and MC simulation.

The biggest limitation of this project is the failure to replicate the estimation of asymptotic variance and the computation of the test statistic. The hypothesized reason for the failure to find asymptotic normality of the test statistic is the presence of numerical instability when calculating matrix inverses. Attemps to address this, such as by adding a small epsilon or using Python's pseudo-inverse function causes drastically different results.

Another major limitation of this study is the naive treatment of time-series and the use of jackknife resampling. The current implementation is based on my understanding of Notes 5, but it likely wrongly assumes independence of observation across time steps. I unfortunately did not have time to fully read up on this in time for the submission of this project, but I will be sure to read about it and update it over break.

01_setup

December 18, 2025

1 Setup and data cleaning

1.1 Notebook setup

[14]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sys
sys.executable
```

[14]:
```
'/Users/fanghema/Desktop/aaSTAT_5200/STAT_5200_final_project/env/bin/python'
```

1.2 Data Preprocessing

I then load in the data from replication dataset.

[2]:
```
data_galvao_et_al = pd.read_csv('../gmo-files/datareturns.txt', sep="\t",␣
↪header=None)
data_galvao_et_al
```

[2]:

	0	1	2	3	4	5	6	7	8	9	…	43	\
0	-0.67	0.00	-0.31	0.01	0.15	0.012	0.33	-0.45	-0.14	-0.20	…	-1.28	
1	0.79	-0.26	0.26	-0.08	-0.19	0.012	0.86	0.65	-0.03	0.29	…	0.34	
2	0.63	-0.17	-0.09	0.19	-0.33	0.012	0.96	0.55	0.83	1.29	…	1.14	
3	0.40	0.08	-0.28	0.07	-0.33	0.012	-0.01	0.09	0.32	0.35	…	1.38	
4	-0.63	0.04	-0.16	-0.31	0.14	0.012	-0.15	-0.28	0.26	-0.37	…	0.18	
…	…	…	…	…	…	…	…	…	…	…	…	…	
12960	0.07	0.32	-0.18	-0.24	-0.08	0.000	-1.30	-0.04	-0.04	-0.10	…	-0.34	
12961	0.39	0.32	-0.24	-0.20	-0.18	0.000	0.23	0.24	-0.06	-0.03	…	0.02	
12962	0.13	0.22	0.61	0.10	0.17	0.000	0.02	-0.22	-0.21	-0.38	…	0.53	
12963	-0.48	0.06	0.23	0.11	0.25	0.000	-0.12	-0.56	-0.39	-0.82	…	-0.30	
12964	-0.93	0.48	-0.45	-0.08	-0.22	0.000	-1.41	-1.51	-1.26	-1.22	…	-1.56	

	44	45	46	47	48	49	50	51	52
0	-0.91	-0.86	-0.33	-0.25	-0.23	-1.35	-0.52	-0.24	-0.07
1	1.15	0.29	0.43	0.72	0.59	0.71	0.37	0.27	-0.26
2	0.60	1.13	0.80	0.96	0.35	0.99	0.41	0.49	0.07

```
3      -0.26  0.61  0.07 -0.16  0.17  1.11  0.03  0.71  0.83
4      -1.27 -0.71 -0.03 -0.42 -0.39 -0.79 -1.08 -0.03 -0.68
…      …    …    …    …    …    …    …    …    …
12960   0.53  0.01 -0.31 -0.02  0.01 -0.03  0.14 -0.08 -0.20
12961   0.21  0.20  0.39  0.67  0.00  0.01  0.46 -0.12 -0.01
12962   0.29  0.21  0.53  0.59  0.44  0.17  0.42  0.11  0.26
12963  -0.10 -0.27 -0.14 -0.68 -0.15  0.16 -0.35 -0.34 -0.23
12964  -0.46 -0.98 -0.32 -0.21 -1.07 -1.26 -1.33 -0.96 -1.09

[12965 rows x 53 columns]
```

Comparing this to the 49-industry dataset from Kenneth French's data library.

[3]:
```
industry_portfolios = pd.read_csv(
    '../data/raw/49_Industry_Portfolios_Daily.csv',
    index_col = 0,
    parse_dates=True
)
ff5_factors = pd.read_csv(
    '../data/raw/F-F_Research_Data_5_Factors_2x3_daily.csv',
    index_col = 0,
    parse_dates=True
)

replication_data = (
    ff5_factors
    .merge(industry_portfolios, left_index=True, right_index=True, how='left')
    .loc["1963-07-01":]
)

replication_data
```

[3]:
```
            Mkt-RF   SMB   HML   RMW   CMA    RF  Agric  Food  Soda  Beer  \
1963-07-01   -0.67  0.00 -0.34 -0.01  0.16  0.01   0.33 -0.45 -0.14 -0.20
1963-07-02    0.79 -0.26  0.26 -0.07 -0.20  0.01   0.86  0.65 -0.03  0.29
1963-07-03    0.63 -0.17 -0.09  0.18 -0.34  0.01   0.96  0.55  0.83  1.29
1963-07-05    0.40  0.08 -0.27  0.09 -0.34  0.01  -0.01  0.09  0.32  0.35
1963-07-08   -0.63  0.04 -0.18 -0.29  0.14  0.01  -0.15 -0.28  0.26 -0.37
…             …    …    …    …    …    …    …    …    …    …
2025-10-27    1.17 -0.81 -1.21 -0.23 -1.19  0.02   0.36  0.59  0.30  1.42
2025-10-28    0.18 -0.34 -0.61  0.63 -1.01  0.02  -0.91 -1.06 -0.01 -1.60
2025-10-29   -0.09 -0.98 -0.81  0.30 -1.37  0.02  -0.46 -3.01 -2.64 -3.19
2025-10-30   -1.10  0.00  0.67 -0.27  0.42  0.02  -2.09 -0.15  0.69  0.82
2025-10-31    0.40 -0.07 -0.24 -1.31  0.18  0.02  -0.29 -0.22 -0.06 -0.69

            …  Boxes  Trans  Whlsl  Rtail  Meals  Banks  Insur  RlEst   Fin  \
1963-07-01  …  -1.28  -0.91  -0.83  -0.33  -0.25  -0.27  -1.35  -0.51 -0.20
1963-07-02  …   0.34   1.15   0.33   0.43   0.69   0.47   0.71   0.33  0.43
```

```
1963-07-03  …   1.14   0.60   1.13   0.80   0.96  -0.01   0.99   0.38  0.80
1963-07-05  …   1.38  -0.26   0.60   0.07  -0.16   0.14   1.11   0.02  0.64
1963-07-08  …   0.18  -1.27  -0.71  -0.03  -0.42  -0.45  -0.79  -1.06 -0.02
…           …   …     …      …      …      …      …      …      …     …
2025-10-27  …   0.11   1.26   0.31   0.56   0.49   0.52   0.33  -0.64  0.91
2025-10-28  …  -0.67  -0.72  -0.93   0.31  -1.57  -0.17  -1.20  -1.14 -0.54
2025-10-29  …  -4.25  -0.13  -0.52  -0.49  -1.19  -1.04  -1.53  -3.46 -1.34
2025-10-30  …  -0.47   0.39   1.36  -1.78  -1.99   0.65  -1.42  -0.67 -0.55
2025-10-31  …   0.22   0.85  -0.06   3.93  -1.09   0.32  -0.38   0.13  0.46

            Other
1963-07-01   0.14
1963-07-02   0.13
1963-07-03   0.09
1963-07-05   1.02
1963-07-08  -0.38
…           …
2025-10-27  -0.72
2025-10-28  -1.40
2025-10-29  -1.51
2025-10-30   0.53
2025-10-31  -0.16

[15690 rows x 55 columns]
```

[4]:
```python
corr_df = pd.DataFrame(
    index = np.arange(data_galvao_et_al.shape[1]),
    columns = replication_data.columns
)

for i in range(data_galvao_et_al.shape[1]):
    for col in replication_data.columns:
        corr = np.corrcoef(
            data_galvao_et_al.iloc[:, i].values,
            replication_data[col][:data_galvao_et_al.shape[0]].values
        )[0, 1]
        corr_df.loc[i, col] = corr
```

[5]:
```python
columns_map = {}
for i in corr_df.index:
    argmax = corr_df.loc[i].argmax()
    max = corr_df.loc[i].max()
    print(f"column number {i}: {replication_data.columns[argmax]} ({max})")
    columns_map[i] = replication_data.columns[argmax]

print(f"Number of keys: {len(set(columns_map.keys()))}")
print(f"Number of values: {len(set(columns_map.values()))}")
```

3

```
column number 0: Mkt-RF (0.9999631176102544)
column number 1: SMB (0.9983317073995388)
column number 2: HML (0.9707746741309701)
column number 3: RMW (0.9695277483402874)
column number 4: CMA (0.996411334405961)
column number 5: RF (0.9765065560785002)
column number 6: Agric (0.9993517050995383)
column number 7: Food (0.9998997012505297)
column number 8: Soda (0.9999736609358353)
column number 9: Beer (0.9999633984001192)
column number 10: Smoke (0.9996248704879636)
column number 11: Toys (0.9986361761212148)
column number 12: Fun (0.9996860847221328)
column number 13: Books (0.9981487902396878)
column number 14: Hshld (0.9999021300411858)
column number 15: Clths (0.9979290120722208)
column number 16: MedEq (0.9998955127299733)
column number 17: Drugs (0.9998955062570346)
column number 18: Chems (0.9999833203244106)
column number 19: Rubbr (0.9989401195566953)
column number 20: Txtls (0.9978286804583132)
column number 21: BldMt (0.9925498232310794)
column number 22: Cnstr (0.9955760178553862)
column number 23: Steel (0.9990462794382083)
column number 24: FabPr (0.9462112861585128)
column number 25: Mach (0.9897169114843933)
column number 26: ElcEq (0.9981289474892405)
column number 27: Autos (0.9990608501760571)
column number 28: Aero (0.9997757459782651)
column number 29: Ships (0.998256024412168)
column number 30: Guns (0.9997896242645089)
column number 31: Gold (0.9985885553277145)
column number 32: Mines (0.9995911312732187)
column number 33: Coal (0.9986400720334226)
column number 34: Oil (0.9996662148490669)
column number 35: Util (0.9998391848589837)
column number 36: Telcm (0.9989044702521956)
column number 37: PerSv (0.9994444860801095)
column number 38: BusSv (0.9905182743345038)
column number 39: Hardw (0.9961413009522394)
column number 40: Chips (0.9974214473872154)
column number 41: LabEq (0.9971031439150029)
column number 42: Paper (0.9696560686178576)
column number 43: Boxes (0.9978022637534089)
column number 44: Trans (0.9996326924037361)
column number 45: Whlsl (0.9983352779355394)
column number 46: Rtail (0.9999036487527976)
column number 47: Meals (0.9933848285355421)
```

```
column number 48: Banks (0.9989749220253604)
column number 49: Insur (0.9994260713508203)
column number 50: RlEst (0.9459155034739062)
column number 51: Fin (0.9984710426181009)
column number 52: Other (0.984467019976098)
Number of keys: 53
Number of values: 53
```

We check for the two columns that were not matched to any columns in Galvao et al's dataset. This indeed matches their description.

[6]:
```
for col in replication_data.columns:
    if col not in columns_map.values():
        print(col)
```

```
Hlth
Softw
```

We update column names and indices for the original dataframe. We also drop Healthcare and Software from our extended dataframe.

[7]:
```
replication_data = replication_data.drop(columns=['Hlth', 'Softw'])
data_galvao_et_al.rename(columns = columns_map, inplace=True)
data_galvao_et_al.index = replication_data.index[:data_galvao_et_al.shape[0]]
```

Data cleaning and replacing invalid values with 0.

[8]:
```
replication_data = (
    replication_data
    .replace([np.inf, -np.inf], np.nan)
    .fillna(0)
)

data_galvao_et_al = (
    data_galvao_et_al
    .replace([np.inf, -np.inf], np.nan)
    .fillna(0)
)
```

[9]:
```
data_galvao_et_al.to_csv('../data/processed/data_galvao.csv')
replication_data.to_csv("../data/processed/data_extended.csv")
```

1.3 Summary Statistics

[11]:
```
replication_data.describe()
```

[11]:

	Mkt-RF	SMB	HML	RMW	CMA \
count	15690.000000	15690.000000	15690.000000	15690.000000	15690.000000
mean	0.028841	0.005815	0.014004	0.012736	0.011643
std	1.023264	0.551013	0.585571	0.403702	0.382175

```
min      -17.440000    -11.150000     -5.030000     -2.970000     -5.310000
25%       -0.420000     -0.280000     -0.240000     -0.180000     -0.180000
50%        0.050000      0.020000      0.010000      0.010000      0.010000
75%        0.510000      0.310000      0.260000      0.190000      0.200000
max       11.360000      6.080000      6.730000      4.570000      2.480000

                RF         Agric          Food          Soda          Beer  \
count  15690.000000  15690.000000  15690.000000  15690.000000  15690.000000
mean       0.017183      0.049493      0.044612      0.053917      0.049760
std        0.012740      1.437237      0.905245      1.356966      1.126727
min        0.000000    -15.270000    -16.080000    -19.220000    -14.730000
25%        0.010000     -0.630000     -0.410000     -0.610000     -0.530000
50%        0.020000      0.040000      0.060000      0.050000      0.040000
75%        0.020000      0.730000      0.510000      0.690000      0.620000
max        0.060000     20.320000     10.010000     11.680000     11.460000

       …         Boxes         Trans         Whlsl         Rtail  \
count  …  15690.000000  15690.000000  15690.000000  15690.000000
mean   …      0.045228      0.045016      0.048480      0.052345
std    …      1.260816      1.252522      1.088046      1.156068
min    …    -21.350000    -17.570000    -13.250000    -17.970000
25%    …     -0.610000     -0.600000     -0.480000     -0.520000
50%    …      0.050000      0.050000      0.070000      0.070000
75%    …      0.700000      0.680000      0.610000      0.620000
max    …     10.910000     12.710000     10.640000     11.750000

              Meals         Banks         Insur         RlEst           Fin  \
count  15690.000000  15690.000000  15690.000000  15690.000000  15690.000000
mean       0.054479      0.047213      0.047164      0.035504      0.055850
std        1.247532      1.417553      1.174384      1.449594      1.427951
min      -15.260000    -16.960000    -15.200000    -17.640000    -16.360000
25%       -0.577500     -0.540000     -0.490000     -0.600000     -0.500000
50%        0.070000      0.040000      0.060000      0.050000      0.070000
75%        0.690000      0.630000      0.600000      0.680000      0.640000
max       15.890000     16.940000     17.820000     17.120000     18.180000

              Other
count  15690.000000
mean       0.030916
std        1.408428
min      -17.230000
25%       -0.590000
50%        0.050000
75%        0.680000
max       16.840000

[8 rows x 53 columns]
```

[12]:
```
data_galvao_et_al.describe()
```

[12]:

	Mkt-RF	SMB	HML	RMW	CMA	\
count	12965.000000	12965.000000	12965.000000	12965.000000	12965.000000	
mean	0.024251	0.009328	0.017895	0.012640	0.015580	
std	0.992087	0.520785	0.495309	0.348898	0.360934	
min	-17.440000	-11.180000	-5.090000	-2.860000	-5.240000	
25%	-0.420000	-0.260000	-0.210000	-0.160000	-0.170000	
50%	0.050000	0.030000	0.010000	0.010000	0.010000	
75%	0.490000	0.300000	0.240000	0.170000	0.190000	
max	11.350000	6.090000	4.040000	4.100000	2.490000	

	RF	Agric	Food	Soda	Beer	\
count	12965.000000	12965.000000	12965.000000	12965.000000	12965.000000	
mean	0.019350	0.051073	0.050685	0.056383	0.054464	
std	0.012195	1.396318	0.879694	1.406782	1.128794	
min	0.000000	-15.270000	-16.040000	-19.220000	-14.730000	
25%	0.012000	-0.630000	-0.390000	-0.630000	-0.530000	
50%	0.019000	0.040000	0.060000	0.050000	0.040000	
75%	0.025000	0.730000	0.500000	0.720000	0.620000	
max	0.061000	20.320000	9.980000	11.680000	10.120000	

	…	Boxes	Trans	Whlsl	Rtail	\
count	…	12965.000000	12965.000000	12965.000000	12965.000000	
mean	…	0.048595	0.046272	0.048008	0.050200	
std	…	1.243543	1.215566	1.059366	1.133851	
min	…	-21.430000	-17.560000	-13.200000	-18.000000	
25%	…	-0.600000	-0.590000	-0.470000	-0.520000	
50%	…	0.050000	0.050000	0.080000	0.060000	
75%	…	0.680000	0.670000	0.590000	0.600000	
max	…	10.920000	9.330000	9.750000	11.750000	

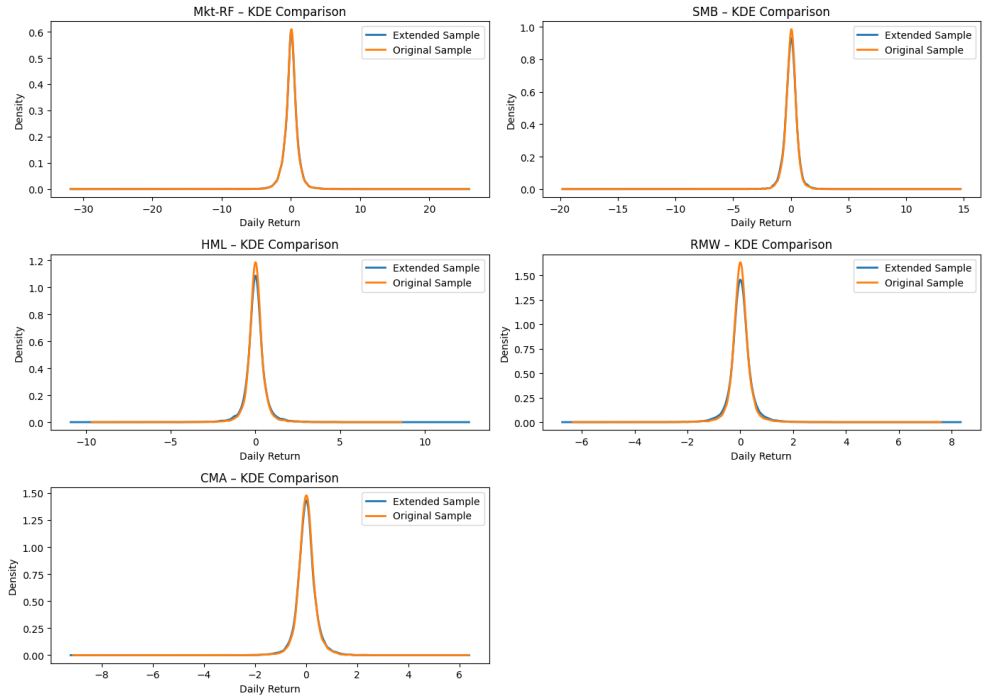
	Meals	Banks	Insur	RlEst	Fin	\
count	12965.000000	12965.000000	12965.000000	12965.000000	12965.000000	
mean	0.055287	0.044963	0.046565	0.029755	0.054201	
std	1.252471	1.382958	1.153460	1.431005	1.414357	
min	-15.480000	-16.980000	-13.980000	-15.500000	-16.280000	
25%	-0.590000	-0.520000	-0.490000	-0.600000	-0.480000	
50%	0.070000	0.040000	0.060000	0.030000	0.060000	
75%	0.720000	0.590000	0.580000	0.680000	0.610000	
max	11.490000	16.960000	17.840000	21.900000	17.940000	

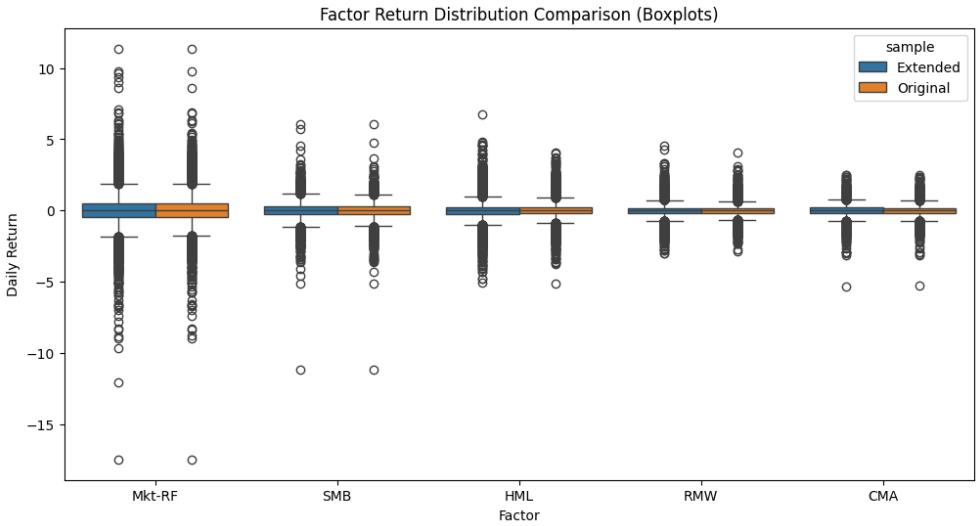
	Other
count	12965.000000
mean	0.029523
std	1.433517
min	-17.260000

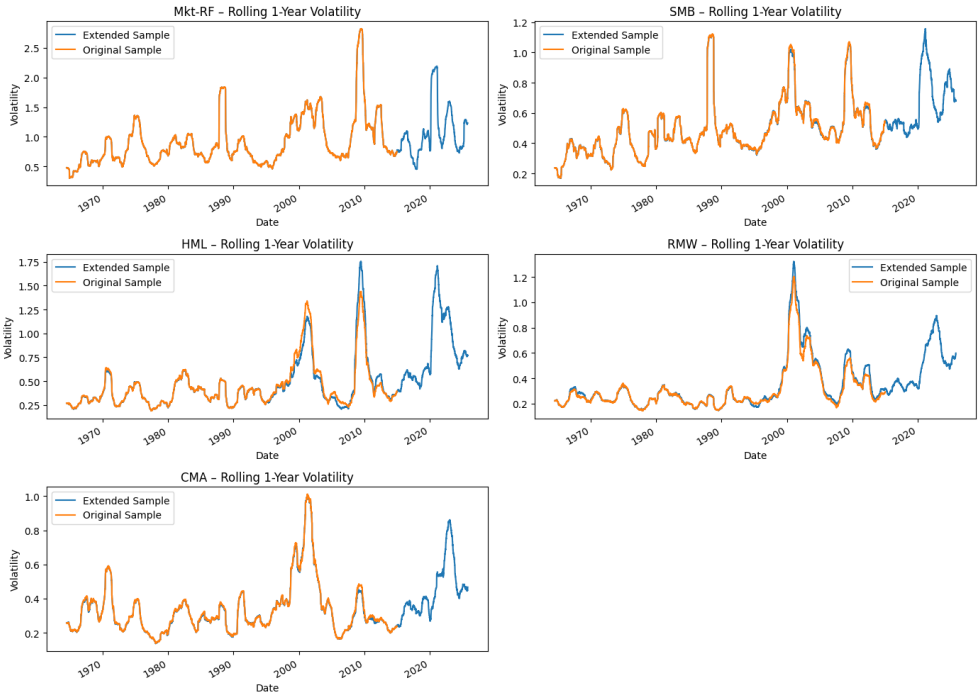
```
25%       -0.610000
50%        0.050000
75%        0.700000
max       16.840000

[8 rows x 53 columns]
```

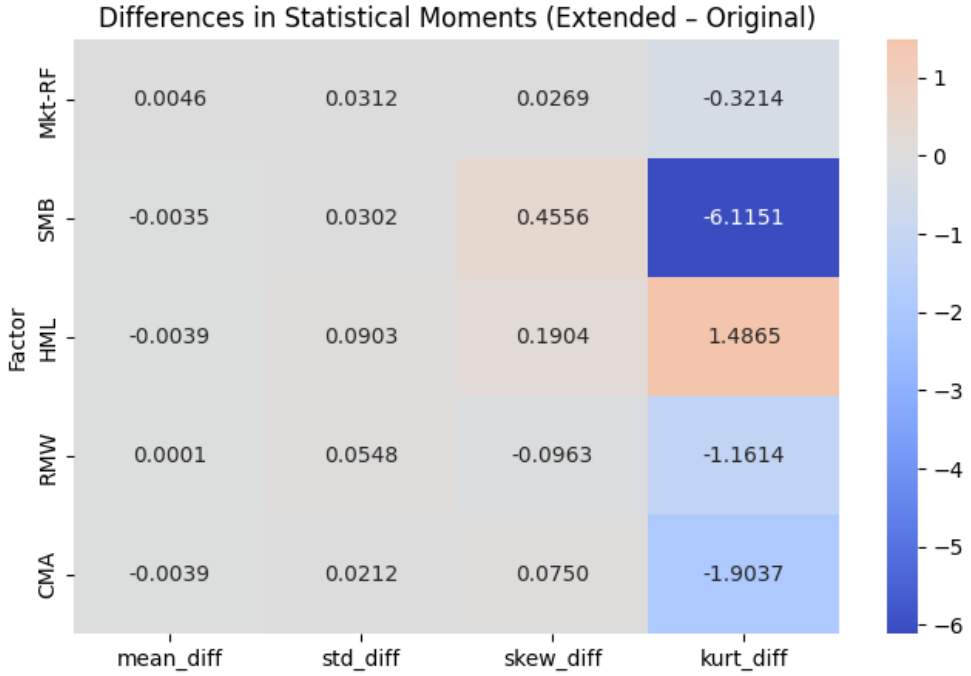
[15]:
```
factors = ["Mkt-RF", "SMB", "HML", "RMW", "CMA"]

plt.figure(figsize=(14, 10))

for i, fac in enumerate(factors, 1):
    plt.subplot(3, 2, i)
    replication_data[fac].plot(kind="kde", label="Extended Sample", linewidth=2)
    data_galvao_et_al[fac].plot(kind="kde", label="Original Sample",␣
 ↪linewidth=2)
    plt.title(f"{fac} - KDE Comparison")
    plt.xlabel("Daily Return")
    plt.legend()

plt.tight_layout()
plt.show()

combined = pd.concat([
    replication_data[factors].assign(sample="Extended"),
    data_galvao_et_al[factors].assign(sample="Original")
])

combined_melted = combined.melt(id_vars="sample")

plt.figure(figsize=(12, 6))
sns.boxplot(data=combined_melted, x="variable", y="value", hue="sample")
plt.title("Factor Return Distribution Comparison (Boxplots)")
plt.xlabel("Factor")
plt.ylabel("Daily Return")
plt.show()

plt.figure(figsize=(14, 10))

for i, fac in enumerate(factors, 1):
    plt.subplot(3, 2, i)
    replication_data[fac].rolling(252).std().plot(label="Extended Sample",␣
 ↪linewidth=1.5)
    data_galvao_et_al[fac].rolling(252).std().plot(label="Original Sample",␣
 ↪linewidth=1.5)
    plt.title(f"{fac} - Rolling 1-Year Volatility")
    plt.xlabel("Date")
```

```
    plt.ylabel("Volatility")
    plt.legend()

plt.tight_layout()
plt.show()


stats = pd.DataFrame({
    "mean_diff": replication_data[factors].mean() - data_galvao_et_al[factors].
 ↪mean(),
    "std_diff": replication_data[factors].std() - data_galvao_et_al[factors].
 ↪std(),
    "skew_diff": replication_data[factors].skew() - data_galvao_et_al[factors].
 ↪skew(),
    "kurt_diff": replication_data[factors].kurt() - data_galvao_et_al[factors].
 ↪kurt()
})

plt.figure(figsize=(8, 5))
sns.heatmap(stats, annot=True, cmap="coolwarm", center=0, fmt=".4f")
plt.title("Differences in Statistical Moments (Extended - Original)")
plt.ylabel("Factor")
plt.show()
```




Factor Return Distribution Comparison (Boxplots)
sample
Extended
Original
Daily Return
10
5
0
−5
−10
−15
Mkt-RF
SMB
HML
RMW
CMA
Factor


Mkt-RF – Rolling 1-Year Volatility
SMB – Rolling 1-Year Volatility
HML – Rolling 1-Year Volatility
RMW – Rolling 1-Year Volatility
CMA – Rolling 1-Year Volatility
Extended Sample
Original Sample
Volatility
Date
1970
1980
1990
2000
2010
2020

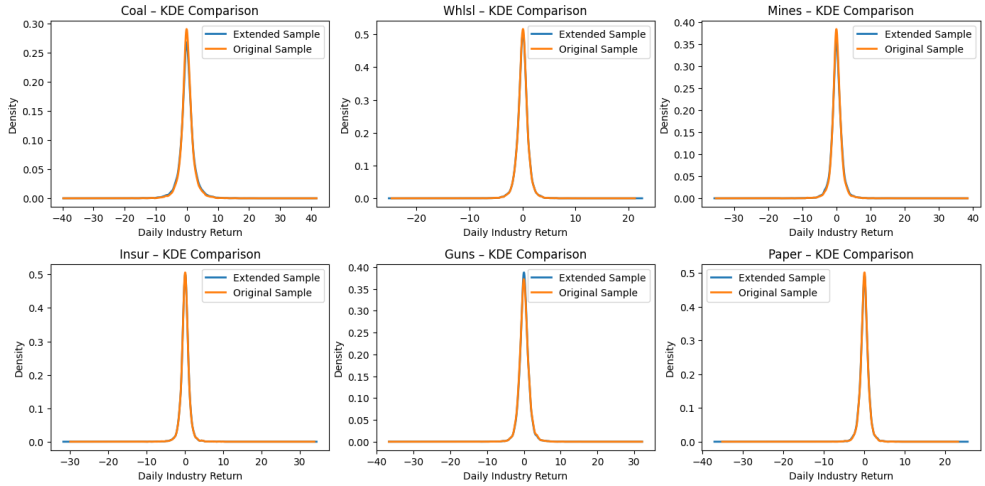


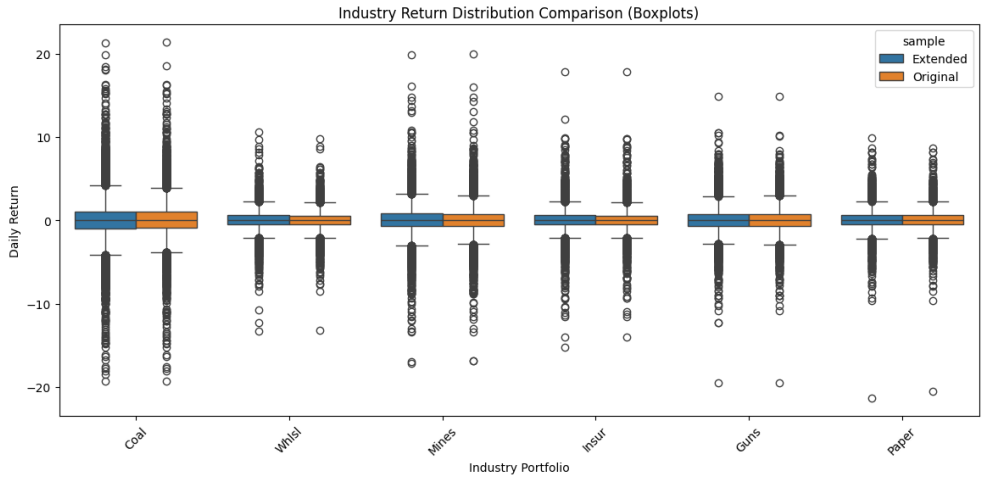
[16]:
```
N_SAMPLE_INDUSTRIES = 6

factor_cols = ["Mkt-RF", "SMB", "HML", "RMW", "CMA", "RF"]
industry_cols = [col for col in replication_data.columns if col not in␣
 ↪factor_cols]

np.random.seed(42)
sampled_industries = np.random.choice(industry_cols, size=N_SAMPLE_INDUSTRIES,␣
 ↪replace=False).tolist()

print("Sampled industries:", sampled_industries)

plt.figure(figsize=(14, 10))

for i, ind in enumerate(sampled_industries, 1):
    plt.subplot(3, 3, i)
    replication_data[ind].plot(kind="kde", label="Extended Sample", linewidth=2)
    data_galvao_et_al[ind].plot(kind="kde", label="Original Sample",␣
 ↪linewidth=2)
    plt.title(f"{ind} - KDE Comparison")
    plt.xlabel("Daily Industry Return")
    plt.legend()
```

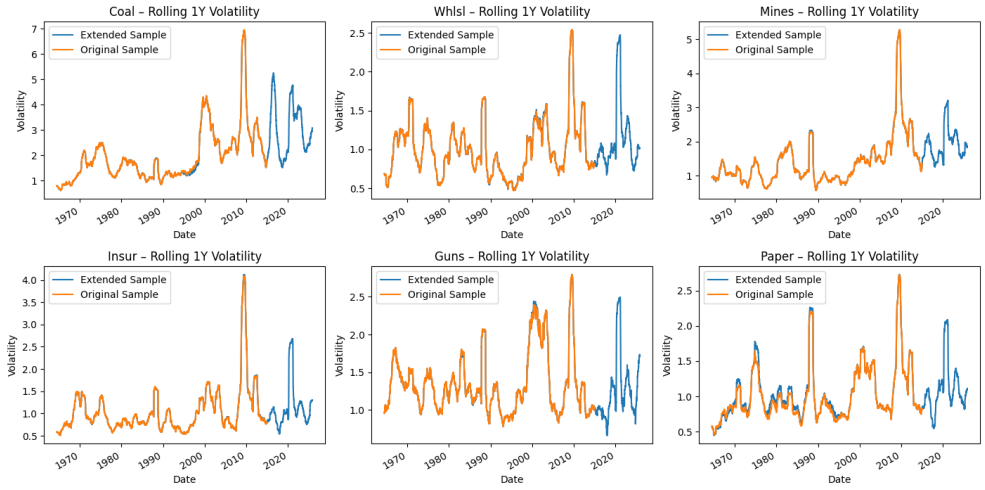
```
plt.tight_layout()
plt.show()


combined_ind = pd.concat([
    replication_data[sampled_industries].assign(sample="Extended"),
    data_galvao_et_al[sampled_industries].assign(sample="Original")
])

combined_ind_melted = combined_ind.melt(id_vars="sample")

plt.figure(figsize=(14, 6))
sns.boxplot(data=combined_ind_melted, x="variable", y="value", hue="sample")
plt.title("Industry Return Distribution Comparison (Boxplots)")
plt.xlabel("Industry Portfolio")
plt.ylabel("Daily Return")
plt.xticks(rotation=45)
plt.show()


plt.figure(figsize=(14, 10))

for i, ind in enumerate(sampled_industries, 1):
    plt.subplot(3, 3, i)
    replication_data[ind].rolling(252).std().plot(label="Extended Sample",␣
 ↪linewidth=1.5)
    data_galvao_et_al[ind].rolling(252).std().plot(label="Original Sample",␣
 ↪linewidth=1.5)
    plt.title(f"{ind} - Rolling 1Y Volatility")
    plt.xlabel("Date")
    plt.ylabel("Volatility")
    plt.legend()

plt.tight_layout()
plt.show()


stats_ind = pd.DataFrame({
    "mean_diff": replication_data[sampled_industries].mean() -␣
 ↪data_galvao_et_al[sampled_industries].mean(),
    "std_diff": replication_data[sampled_industries].std() -␣
 ↪data_galvao_et_al[sampled_industries].std(),
    "skew_diff": replication_data[sampled_industries].skew() -␣
 ↪data_galvao_et_al[sampled_industries].skew(),
    "kurt_diff": replication_data[sampled_industries].kurt() -␣
 ↪data_galvao_et_al[sampled_industries].kurt()
```

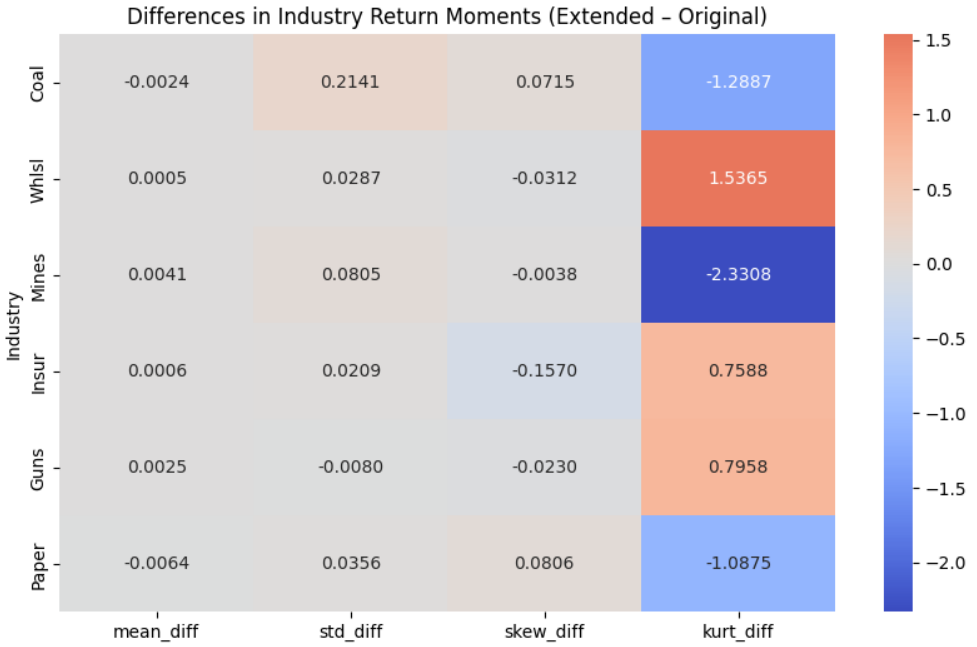
```
})

plt.figure(figsize=(10, 6))
sns.heatmap(stats_ind, annot=True, cmap="coolwarm", center=0, fmt=".4f")
plt.title("Differences in Industry Return Moments (Extended - Original)")
plt.ylabel("Industry")
plt.show()
```

```
Sampled industries: ['Coal', 'Whlsl', 'Mines', 'Insur', 'Guns', 'Paper']
```






Coal – Rolling 1Y Volatility
Whlsl – Rolling 1Y Volatility
Mines – Rolling 1Y Volatility
Insur – Rolling 1Y Volatility
Guns – Rolling 1Y Volatility
Paper – Rolling 1Y Volatility
Extended Sample
Original Sample
Volatility
Date


Differences in Industry Return Moments (Extended – Original)
Industry
Coal
Whlsl
Mines
Insur
Guns
Paper
-0.0024
0.2141
0.0715
-1.2887
0.0005
0.0287
-0.0312
1.5365
0.0041
0.0805
-0.0038
-2.3308
0.0006
0.0209
-0.1570
0.7588
0.0025
-0.0080
-0.0230
0.7958
-0.0064
0.0356
0.0806
-1.0875
mean_diff
std_diff
skew_diff
kurt_diff

02_factor_loading

December 18, 2025

1 Estimating Factor Loading

Galvao et al estimated quarterly factor loading from daily data.

1.1 Notebook setup

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sys
sys.executable
```

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

For now, we load in Galvao et al's replication data.

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

[3]:
```
factors = ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA']
assets = [col for col in data.columns if col != 'RF' and col not in factors]
```

1.2 Factor Loading Function

[4]:
```
def calculate_factor_loading(
    input_df: pd.DataFrame,
    factors: list[str],
    assets: list[str],
) -> tuple[pd.DataFrame, pd.DataFrame]:
    """
    Given DataFrame of (non-excess) asset returns
    and factor returns,
    returns panel data of factor loadings

    Args:
```

```
        input_df (pd.DataFrame): DataFrame indexed on date,
            with column names corresponding to assets
        factors (list[str]): list of factors
        assets (list[str]): list of risky assets

    Returns:
        pd.DataFrame: panel data of factor loadings
        pd.DataFrame: modified returns dataframe with excess returns
    """

    assert type(input_df.index) == pd.DatetimeIndex, "input_df has wrong index"
    for factor in factors:
        assert factor in input_df.columns, f"missing factor {factor}"
    for asset in assets:
        assert asset in input_df.columns, f"missing asset {asset}"
    assert "RF" in input_df.columns, f"Missing risk free"

    input_df.sort_index(inplace=True)
    N = len(assets)
    K = len(factors)
    input_df['Quarter'] = input_df.index.to_period("Q")
    T = input_df['Quarter'].nunique()

    for col in assets:
        input_df[col] = input_df[col] - input_df["RF"]

    cols = list(assets) + list(factors)

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

    quarters = sorted(input_df['Quarter'].unique())
    for i, quarter in enumerate(quarters):
        returns = (
            input_df.loc[
                input_df['Quarter'] == quarter,
                cols
            ]
            .values
        )
        Omega_hat_t = returns.T @ returns
        realized_covariance_matrices[:, :, i] = Omega_hat_t[:N, N:N+K]

    beta_loading = pd.DataFrame(
        index = pd.MultiIndex.from_product([assets, factors]),
        columns = input_df['Quarter'].unique(),
    )
```

```
    for i, asset in enumerate(assets):
        for j, factor in enumerate(factors):
            omega_i_j_series = realized_covariance_matrices[i, j, :]
            Y = omega_i_j_series[1:]
            X = (
                np.column_stack([
                    np.ones(len(Y)),
                    omega_i_j_series[:-1]
                ])
            )
            b = np.linalg.lstsq(X, Y, rcond=None)[0]
            delta0, delta1 = b
            beta_loading.loc[(asset, factor)] = delta0 + delta1 *␣
 ↪omega_i_j_series

    return beta_loading, input_df

beta_loading, _ = calculate_factor_loading(data, factors=factors, assets=assets)
```

[5]:
```
beta_loading
```

[5]:

		1963Q3	1963Q4	1964Q1	1964Q2	1964Q3	\
Agric	Mkt-RF	30.361188	29.34117	30.059214	30.543622	30.323886	
	SMB	1.816013	1.79645	1.741765	1.853506	1.473105	
	HML	-1.600764	-3.201627	-1.574742	0.111486	-3.858092	
	RMW	-1.466798	0.215607	-2.085727	-1.616463	-0.83943	
	CMA	-3.153214	-2.825331	-3.644188	-2.283014	-4.513127	
…		…	…	…	…	…	
Other	Mkt-RF	37.969554	45.784074	34.019282	34.391633	34.792978	
	SMB	-1.998767	-2.640143	-0.85192	-0.306529	-0.633693	
	HML	-0.917399	1.711179	-1.088099	-3.044638	-1.742651	
	RMW	-1.124262	-2.770248	-2.015389	-2.132075	-3.193049	
	CMA	-5.789944	-6.603539	-5.826818	-6.053781	-5.221453	

		1964Q4	1965Q1	1965Q2	1965Q3	1965Q4	…	\
Agric	Mkt-RF	30.703206	31.228282	37.229192	32.067519	31.861254	…	
	SMB	2.00402	3.045183	4.653787	1.888597	5.220041	…	
	HML	-0.235879	0.086952	-1.560099	-0.679981	0.349221	…	
	RMW	-2.021263	-0.083306	0.020307	0.034482	-0.414664	…	
	CMA	-2.645516	-0.616308	-3.777817	-3.716449	-2.888483	…	
…		…	…	…	…	…	…	
Other	Mkt-RF	34.918142	35.850948	44.439513	35.915049	35.431179	…	
	SMB	1.262379	0.756938	7.587363	1.695747	2.722596	…	
	HML	-1.884642	-2.515519	-2.530227	-2.782289	-0.82352	…	
	RMW	-2.799131	-1.728237	-1.088422	-0.85667	-2.905094	…	
	CMA	-4.204414	-4.85491	-7.189725	-5.199381	-4.323135	…	

```
                  2012Q3     2012Q4     2013Q1     2013Q2     2013Q3  \
Agric Mkt-RF   38.230127  44.228211  39.335887  48.107139  35.757627
      SMB       2.825341   2.55903   1.395667   6.672073   0.987797
      HML      -0.932763  -0.81881   0.285446   3.049976  -1.072834
      RMW      -2.202299   0.605827  -3.652465  -4.020965  -3.187997
      CMA       -4.54621  -5.343885  -2.406726  -2.818723  -1.857074
…                    …          …          …          …          …
Other Mkt-RF   45.704996   49.68986  45.765309  55.216365  43.633398
      SMB       0.412122  -0.559866   1.345408   4.362684   1.628486
      HML       0.311998   1.521048   1.230703   2.823396  -0.727235
      RMW      -2.588129  -2.488581  -5.192548  -5.965385  -5.007897
      CMA      -5.914405  -4.596196  -3.697755  -4.133301  -3.866694

                  2013Q4     2014Q1     2014Q2     2014Q3     2014Q4
Agric Mkt-RF   38.445064  40.051037  39.497776  33.855069  45.157696
      SMB       3.595121   4.930176   4.457616   2.713055   3.671996
      HML      -2.233552  -3.741736   -4.58758  -3.889497  -3.510954
      RMW      -2.908149  -2.537573  -3.156205  -1.500751  -4.065958
      CMA      -3.305422  -3.470351  -5.198915  -3.512337  -4.358914
…                    …          …          …          …          …
Other Mkt-RF   46.574096  49.724401  40.962994  42.389573  54.260984
      SMB       1.340596     2.8245   2.506789   0.791631  -1.422422
      HML        -2.7199  -2.724412   -3.61055  -4.402122  -3.550449
      RMW       -4.39195  -3.473228  -3.751173  -3.608661  -4.698762
      CMA      -5.743321  -5.302565  -5.333998  -5.560285  -5.226873

[235 rows x 206 columns]
```

1.3 Sanity Check

As a sanity check, we call our function on Galvao's original dataset, and compare the estimated beta loading against the original paper's beta loadings.

[6]:
```
agric_betas = pd.read_csv("../gmo-files/omegareg1.txt",
                          sep=r"\s+",
                          header=None)
agric_betas.columns = factors
agric_betas
```

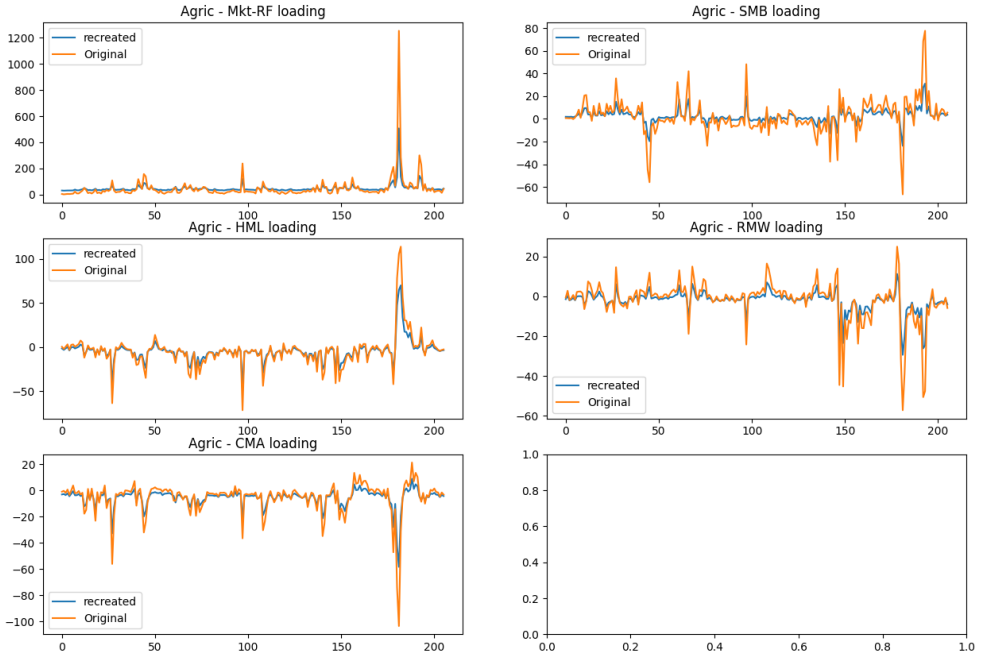
[6]:
```
       Mkt-RF      SMB      HML      RMW      CMA
0     3.60917  0.67097  0.32962 -0.66613 -1.01100
1     0.93072  0.61959 -2.20990  2.73658 -0.40266
2     2.81622  0.47597  0.37090 -1.91793 -1.92193
3     4.08822  0.76944  3.04584 -0.96883  0.60353
4     3.51122 -0.22962 -3.25128  0.60274 -3.53412
..        …        …        …        …        …
201  24.83650  5.34350 -0.67420 -3.58130 -1.29340
```

```
202  29.05360  8.84980 -3.06670 -2.83180 -1.59940
203  27.60080  7.60870 -4.40850 -4.08300 -4.80650
204  12.78370  3.02690 -3.30110 -0.73480 -1.67730
205  42.46310  5.54540 -2.70060 -5.92300 -3.24800

[206 rows x 5 columns]
```

[7]:
```
beta_loading.loc['Agric'].T
```

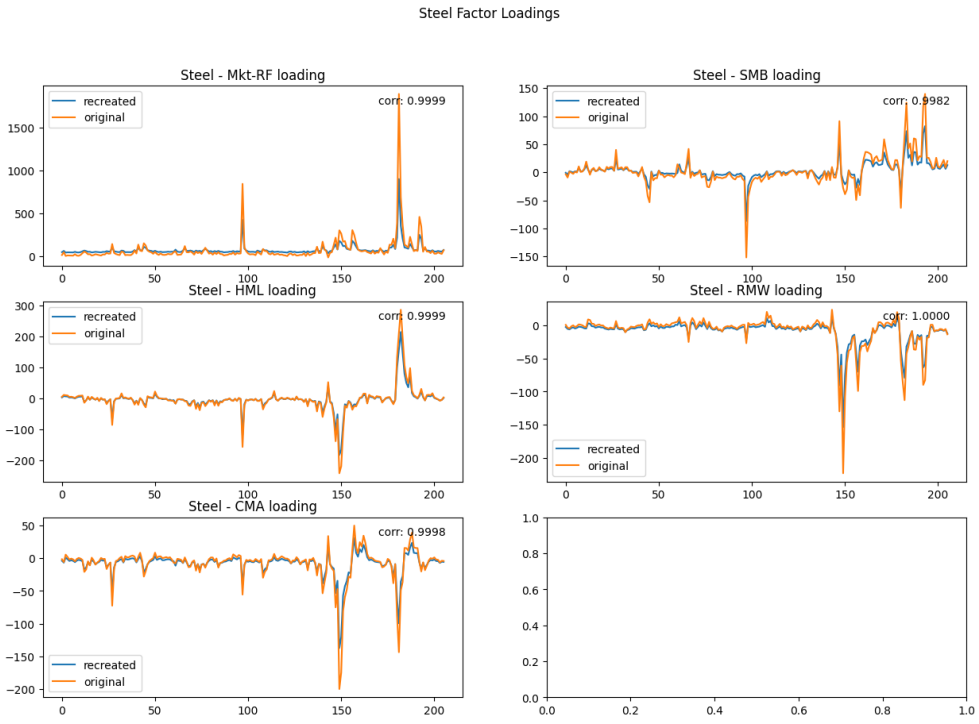
[7]:
```
           Mkt-RF       SMB       HML       RMW       CMA
1963Q3  30.361188  1.816013 -1.600764 -1.466798 -3.153214
1963Q4   29.34117   1.79645 -3.201627  0.215607 -2.825331
1964Q1  30.059214  1.741765 -1.574742 -2.085727 -3.644188
1964Q2  30.543622  1.853506  0.111486 -1.616463 -2.283014
1964Q3  30.323886  1.473105 -3.858092  -0.83943 -4.513127
…            …         …         …         …         …
2013Q4  38.445064  3.595121 -2.233552 -2.908149 -3.305422
2014Q1  40.051037  4.930176 -3.741736 -2.537573 -3.470351
2014Q2  39.497776  4.457616  -4.58758 -3.156205 -5.198915
2014Q3  33.855069  2.713055 -3.889497 -1.500751 -3.512337
2014Q4  45.157696  3.671996 -3.510954 -4.065958 -4.358914

[206 rows x 5 columns]
```

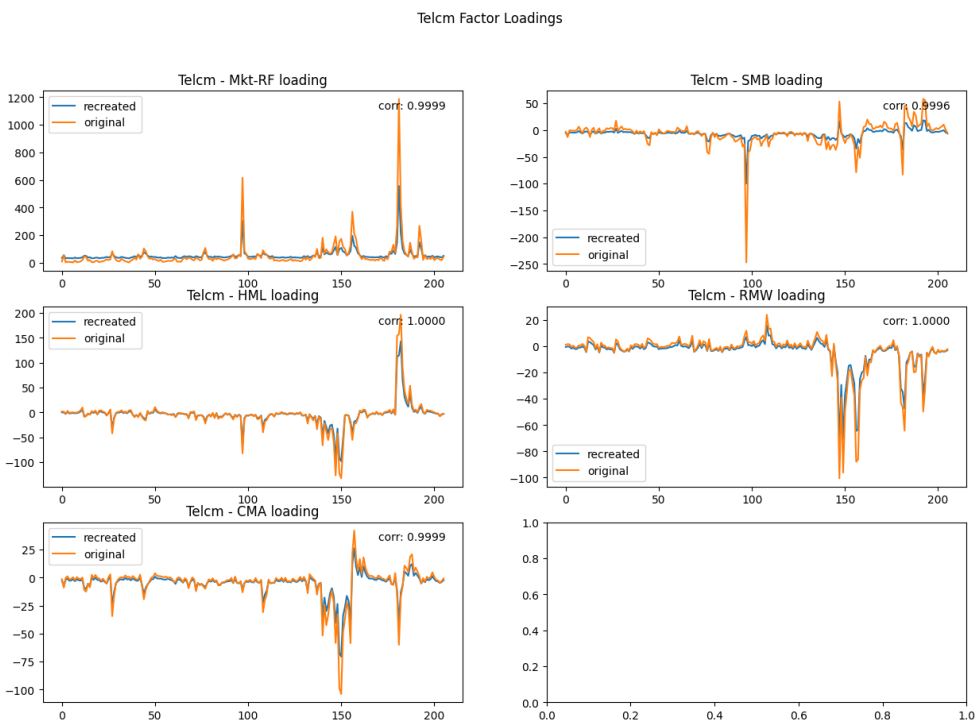
[8]:
```
fig, axes = plt.subplots(3, 2, figsize = (15, 10))
axes = np.ravel(axes)
for i, factor in enumerate(factors):
    axes[i].plot(
        beta_loading.loc['Agric'].T[factor].values, label='recreated'
    )
    axes[i].plot(
        agric_betas[factor], label="Original"
    )
    axes[i].legend()
    axes[i].set_title(f"Agric - {factor} loading")
```

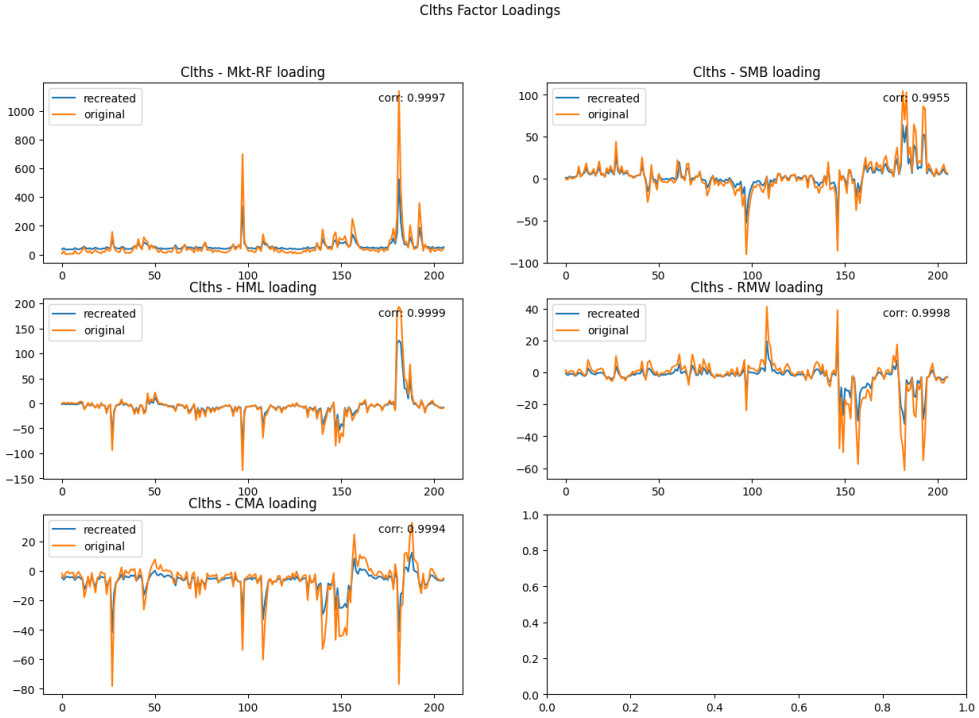


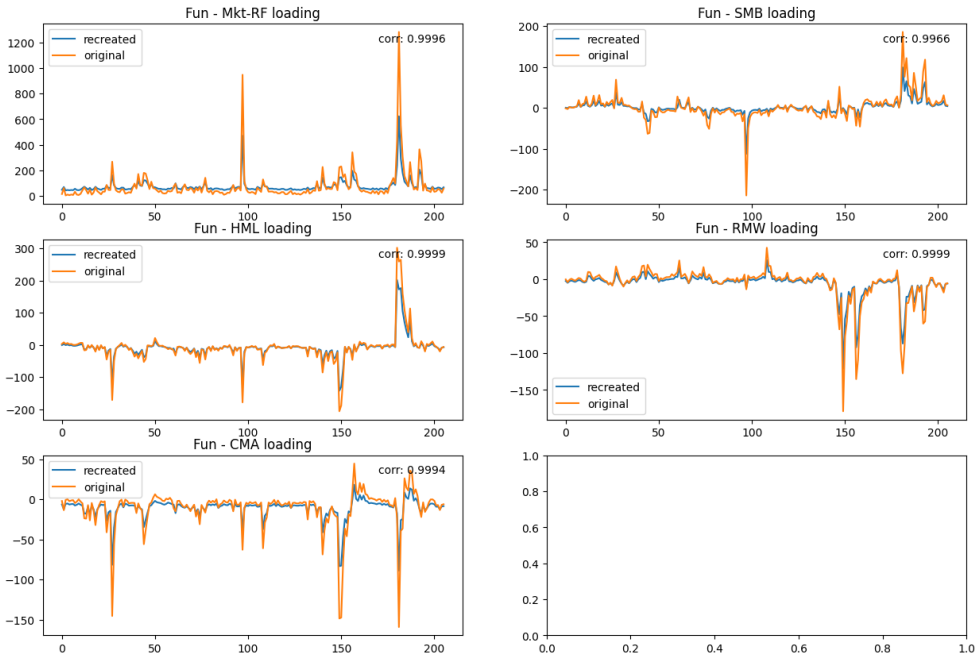
[9]:
```
random_sample_assets = np.random.choice(47, 5, replace=False)

for rng in random_sample_assets:
    galvao_estimated_betas = pd.read_csv(f"../gmo-files/omegareg{rng + 1}.txt",
                     sep=r"\s+",
                     header=None)
    galvao_estimated_betas.columns = factors

    asset = assets[rng]

    fig, axes = plt.subplots(3, 2, figsize = (15, 10))
    axes = np.ravel(axes)
    for i, factor in enumerate(factors):
        recreated_arr = beta_loading.loc[asset].T[factor].values
        galvao_arr = galvao_estimated_betas[factor].values
        min_len = min(len(recreated_arr), len(galvao_arr))

        recreated_arr = recreated_arr[:min_len].astype(float)
        galvao_arr    = galvao_arr[:min_len].astype(float)

        axes[i].plot(recreated_arr, label='recreated')
        axes[i].plot(galvao_arr, label='original')
```

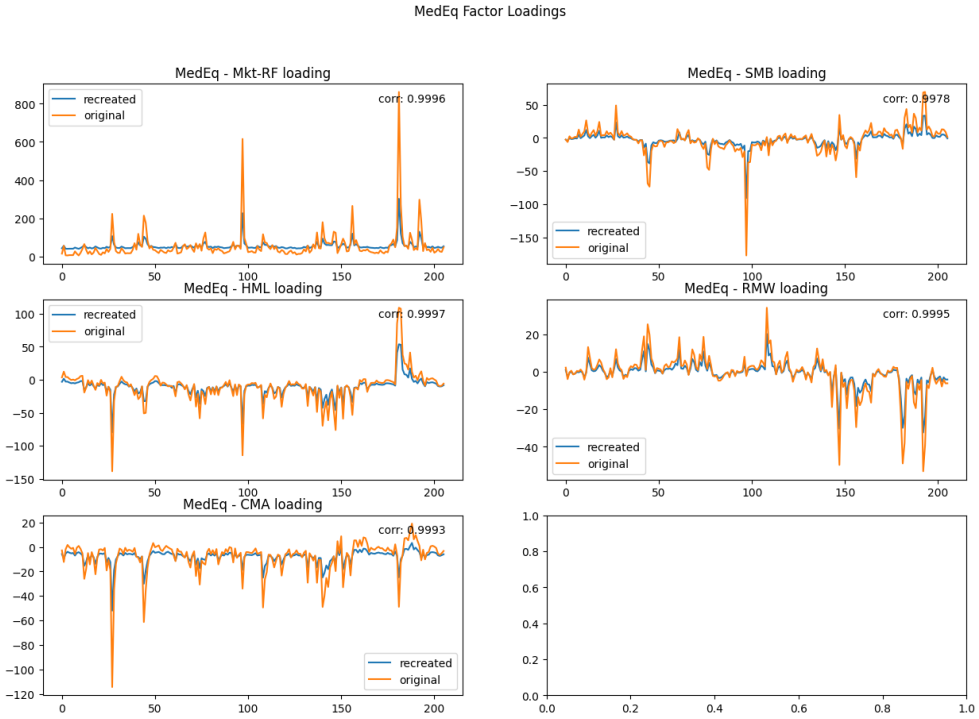
```
        recreated_arr = np.asarray(recreated_arr, dtype=float)
        galvao_arr    = np.asarray(galvao_arr, dtype=float)

        corr = np.corrcoef(recreated_arr, galvao_arr)[0, 1]
        axes[i].text(
            0.8, 0.9,
            f"corr: {corr:.4f}",
            transform = axes[i].transAxes,
        )

        axes[i].legend()
        axes[i].set_title(f"{asset} - {factor} loading")
    fig.suptitle(f"{asset} Factor Loadings")
    plt.show()
```




Telcm Factor Loadings
Telcm - Mkt-RF loading
recreated
original
corr: 0.9999
Telcm - SMB loading
corr: 0.9996
Telcm - HML loading
corr: 1.0000
Telcm - RMW loading
corr: 1.0000
Telcm - CMA loading
corr: 0.9999


Clths Factor Loadings
Clths - Mkt-RF loading
recreated
original
corr: 0.9997
Clths - SMB loading
recreated
original
corr: 0.9955
Clths - HML loading
recreated
original
corr: 0.9999
Clths - RMW loading
recreated
original
corr: 0.9998
Clths - CMA loading
recreated
original
corr: 0.9994


Fun Factor Loadings
Fun - Mkt-RF loading
recreated
original
corr: 0.9996
Fun - SMB loading
recreated
original
corr: 0.9966
Fun - HML loading
recreated
original
corr: 0.9999
Fun - RMW loading
recreated
original
corr: 0.9999
Fun - CMA loading
recreated
original
corr: 0.9994



As a final check, we sample all 47 industries and all 5 factors, and compute correlation for each of the 235 factor loadings.

[10]:
```
corr_df = pd.DataFrame(
    index = assets,
    columns = factors
)
for rng in range(47):
    galvao_estimated_betas = pd.read_csv(f"../gmo-files/omegareg{rng + 1}.txt",
                    sep=r"\s+",
                    header=None)
    galvao_estimated_betas.columns = factors
    asset = assets[rng]
    for i, factor in enumerate(factors):
        recreated_arr = beta_loading.loc[asset].T[factor].values
        galvao_arr = galvao_estimated_betas[factor].values
        min_len = min(len(recreated_arr), len(galvao_arr))
        recreated_arr = recreated_arr[:min_len].astype(float)
        galvao_arr    = galvao_arr[:min_len].astype(float)
        corr = np.corrcoef(recreated_arr, galvao_arr)[0, 1]
        corr_df.loc[asset, factor] = corr
```

```
corr_df
```

[10]:

	Mkt-RF	SMB	HML	RMW	CMA
Agric	0.999567	0.983181	0.999695	0.99985	0.99928
Food	0.999763	0.997571	0.999783	0.999493	0.999491
Soda	0.999783	0.999295	0.999822	0.999621	0.999576
Beer	0.999574	0.994752	0.99978	0.999852	0.99884
Smoke	0.999736	0.999549	0.999402	0.999349	0.999224
Toys	0.999617	0.995785	0.999898	0.999834	0.999364
Fun	0.999625	0.996554	0.999913	0.999914	0.999411
Books	0.999854	0.997026	0.999914	0.999753	0.99961
Hshld	0.999651	0.999656	0.999402	0.99916	0.999419
Clths	0.999723	0.995539	0.999886	0.999774	0.999352
MedEq	0.999614	0.997811	0.999664	0.999511	0.999325
Drugs	0.999699	0.999531	0.999533	0.999346	0.999181
Chems	0.99978	0.998965	0.999699	0.99958	0.999367
Rubbr	0.999662	0.992089	0.999944	0.999864	0.999184
Txtls	0.99971	0.992501	0.99996	0.999876	0.999267
BldMt	0.999812	0.995168	0.999933	0.999798	0.999365
Cnstr	0.999757	0.993609	0.999937	0.999897	0.999613
Steel	0.999887	0.998155	0.99995	0.999958	0.999834
FabPr	0.999748	0.993327	0.999954	0.999933	0.999334
Mach	0.999813	0.995988	0.999885	0.999846	0.999717
ElcEq	0.999889	0.998191	0.999959	0.999946	0.999902
Autos	0.999871	0.99897	0.999914	0.999845	0.999871
Aero	0.999621	0.999025	0.999545	0.9995	0.998883
Ships	0.999509	0.997592	0.999663	0.999658	0.998928
Guns	0.999465	0.99883	0.999349	0.998671	0.99893
Gold	0.999831	0.998684	0.999574	0.998804	0.999737
Mines	0.99991	0.998093	0.999945	0.999834	0.999785
Coal	0.99989	0.996182	0.999942	0.999888	0.999861
Oil	0.999897	0.999347	0.999777	0.999629	0.999686
Util	0.999919	0.998231	0.999909	0.999868	0.999607
Telcm	0.999922	0.999602	0.999964	0.999953	0.999907
PerSv	0.999407	0.989068	0.999864	0.999848	0.999274
BusSv	0.999636	0.988335	0.999924	0.999896	0.999398
Hardw	0.999848	0.999784	0.999915	0.999926	0.999961
Chips	0.999682	0.99777	0.99992	0.999942	0.999919
LabEq	0.999613	0.995699	0.99986	0.999913	0.999741
Paper	0.999807	0.995968	0.999781	0.999533	0.999318
Boxes	0.999828	0.999584	0.999785	0.999628	0.999747
Trans	0.999567	0.993846	0.999848	0.999749	0.999098
Whlsl	0.999585	0.987305	0.999907	0.999842	0.999364
Rtail	0.999733	0.996232	0.999763	0.999307	0.999496
Meals	0.999088	0.994654	0.999521	0.999292	0.998722
Banks	0.999883	0.998357	0.999973	0.999891	0.999817

```
Insur  0.999834   0.99638  0.999924  0.999829  0.999498
RlEst  0.999602  0.985248  0.999954  0.999904  0.999029
Fin    0.999915    0.9973  0.999977  0.999942  0.999897
Other  0.999713  0.991885  0.999973  0.999934  0.999688
```

[ ]:
```
latex_table = corr_df.round(3).to_latex(
    index=True,
    header=True,
    float_format="%.3f",
    escape=False
)

print(latex_table)
```

03_factor_risk_premia

December 18, 2025

1 Estimation of factor risk premia

This section of the study was not included in the online appendix or in the replication code / data.

Galvao et al discussed estimating factor risk premia under time-varying beta loadings, unobserved factors, and cross-correlation.

The authors propose a test statistic and justify its asymptotic properties both theoretically and through a MC study.

The estimation of factor risk premia requires solving a constrained non-convex optimization problem, to which the authors propose a iterative approach.

In this notebook, we attempt to solve the same optimization problem using 1) a penalty-based unconstrained optimization approach and 2) the iterative approach.

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
from utils.utils import iterative_convergence, penalty_based_minimization
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]

def calculate_factor_loading(
```

1

```
    input_df: pd.DataFrame,
    factors: list[str],
    assets: list[str],
) -> tuple[pd.DataFrame, pd.DataFrame]:
    """
    Given DataFrame of (non-excess) asset returns
    and factor returns,
    returns panel data of factor loadings

    Args:
        input_df (pd.DataFrame): DataFrame indexed on date,
            with column names corresponding to assets
        factors (list[str]): list of factors
        assets (list[str]): list of risky assets

    Returns:
        pd.DataFrame: panel data of factor loadings
        pd.DataFrame: modified returns dataframe with excess returns
    """

    assert type(input_df.index) == pd.DatetimeIndex, "input_df has wrong index"
    for factor in factors:
        assert factor in input_df.columns, f"missing factor {factor}"
    for asset in assets:
        assert asset in input_df.columns, f"missing asset {asset}"
    assert "RF" in input_df.columns, f"Missing risk free"

    input_df.sort_index(inplace=True)
    N = len(assets)
    K = len(factors)
    input_df['Quarter'] = input_df.index.to_period("Q")
    T = input_df['Quarter'].nunique()

    for col in assets:
        input_df[col] = input_df[col] - input_df["RF"]

    cols = list(assets) + list(factors)

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

    quarters = sorted(input_df['Quarter'].unique())
    for i, quarter in enumerate(quarters):
        returns = (
            input_df.loc[
                input_df['Quarter'] == quarter,
                cols
            ]
```

```
            .values
        )
        Omega_hat_t = returns.T @ returns
        realized_covariance_matrices[:, :, i] = Omega_hat_t[:N, N:N+K]

    beta_loading = pd.DataFrame(
        index = pd.MultiIndex.from_product([assets, factors]),
        columns = input_df['Quarter'].unique(),
    )

    for i, asset in enumerate(assets):
        for j, factor in enumerate(factors):
            omega_i_j_series = realized_covariance_matrices[i, j, :]
            Y = omega_i_j_series[1:]
            X = (
                np.column_stack([
                    np.ones(len(Y)),
                    omega_i_j_series[:-1]
                ])
            )
            b = np.linalg.lstsq(X, Y, rcond=None)[0]
            delta0, delta1 = b
            beta_loading.loc[(asset, factor)] = delta0 + delta1 *␣
 ↪omega_i_j_series

    return beta_loading, input_df

beta_loading, returns_df = calculate_factor_loading(data, factors=factors,␣
 ↪assets=assets)
```

[3]:
```
x = np.arange(10)
(x+1).prod() - 1
```

[3]:
```
np.int64(3628799)
```

[4]:
```
excess_returns = (
    returns_df
    .groupby("Quarter")
    .sum()
    [assets]
    .T
    .values
)
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
```

1.2 Approach 1 - Softer constraint using penalty

[5]:
```
def penalty_based_minimization(
    beta_hat: np.array,
    excess_returns: np.array,
    N: int,
    K: int,
    R: int,
    T: int,
    lam: float = 1.0,
    lr: float= 1e-2,
    n_iter: int = 2000,
    device: str = "cpu",
    seed: int = 0
) -> tuple[np.array, np.array, np.array, np.array]:
    """
    Solves unconstrained version of equation (24)
    With penalty

    Args:
        beta_hat (np.array): estimated beta loadings, N * K * T
        excess_returns (np.array): excess returns, N * T
        N (int): number of assets
        K (int): number of observed assets
        R (int): number of unobserved factors
        T (int): number of time periods
        lam (float): penalty weight on deviation from identity
        lr (float): learning rate
        n_iter (int): number of iterations
        device (str): cpu
        seed (int): for reproducibility

    Returns:
        tuple[np.array, np.array, np.array]:
            eta: N * (1 + K)
            G: T * R
            beta^*: N * R
            objective: np.array of dimensions num_iter
```

```
    """

    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})"

    torch.manual_seed(seed)

    beta_hat_t = torch.from_numpy(beta_hat).float().to(device)
    beta_hat_t = beta_hat_t.permute(0, 2, 1)  #(N, T, K)
    r = torch.from_numpy(excess_returns).float().to(device) #(N, T)

    ones = torch.ones((N, T, 1), device=device)
    X = torch.cat([ones, beta_hat_t], dim=2) # (N, T, 1 + K)

    # parameters of optimization problem
    eta = torch.nn.Parameter(torch.zeros(N, 1 + K, device=device))
    beta_star = torch.nn.Parameter(torch.zeros(N, R, device=device))
    G = torch.nn.Parameter(torch.zeros(T, R, device=device))

    torch.nn.init.normal_(eta, mean=0.0, std=0.1)
    torch.nn.init.normal_(beta_star, mean=0.0, std=0.1)
    torch.nn.init.normal_(G, mean=0.0, std=0.1)

    optimizer = torch.optim.Adam([eta, beta_star, G], lr=lr)
    I_R = torch.eye(R, device=device)

    objective = np.empty(shape=(n_iter))

    for it in range(n_iter):
        optimizer.zero_grad()
        obs_part = (X * eta[:, None, :]).sum(dim=2)
        latent_part = (G @ beta_star.t()).t()
        pred = obs_part + latent_part
        mse_loss = torch.mean((r - pred) ** 2)
        GTG = G.t() @ G / T
        penalty = torch.norm(GTG - I_R, p='fro')**2
        loss = mse_loss + lam * penalty
        loss.backward()
        optimizer.step()

        objective[it] = mse_loss.item()
```

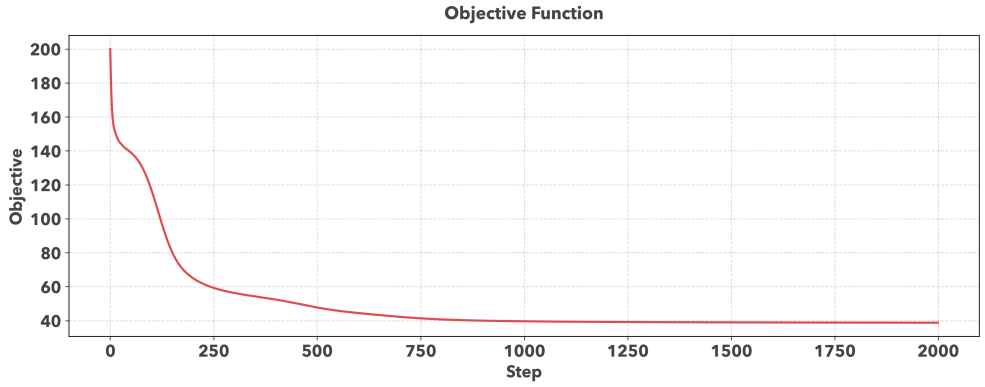
```
        if (it + 1) % 100 == 0:
            log_str = (
                f"Iter {it + 1}/{n_iter}, "
                f"objective={mse_loss.item():.6f}, "
                f"loss={loss.item():.6f}, "
                f"pen={penalty.item():.6f}"
            )
            print(log_str)

    eta_np = eta.detach().cpu().numpy()
    G_np = G.detach().cpu().numpy()
    beta_star_np = beta_star.detach().cpu().numpy()

    return eta_np, G_np, beta_star_np, objective
```

Example usage

[6]:
```
eta1, G1, bstar1, objective = penalty_based_minimization(
    beta_hat_np,
    excess_returns,
    N = 47,
    K = 5,
    R = 3,
    T = 206,
    lam=10
)
```

```
Iter 100/2000, objective=117.313988, loss=117.652817, pen=0.033883
Iter 200/2000, objective=65.169830, loss=68.735237, pen=0.356541
Iter 300/2000, objective=56.320801, loss=58.484299, pen=0.216350
Iter 400/2000, objective=52.429527, loss=53.664684, pen=0.123516
Iter 500/2000, objective=47.771122, loss=48.539043, pen=0.076792
Iter 600/2000, objective=44.462936, loss=44.950157, pen=0.048722
Iter 700/2000, objective=42.243813, loss=42.564178, pen=0.032037
Iter 800/2000, objective=40.720718, loss=40.935093, pen=0.021437
Iter 900/2000, objective=39.984112, loss=40.126289, pen=0.014218
Iter 1000/2000, objective=39.611584, loss=39.707634, pen=0.009605
Iter 1100/2000, objective=39.383282, loss=39.450108, pen=0.006683
Iter 1200/2000, objective=39.225346, loss=39.273193, pen=0.004785
Iter 1300/2000, objective=39.109459, loss=39.144539, pen=0.003508
Iter 1400/2000, objective=39.022053, loss=39.048252, pen=0.002620
Iter 1500/2000, objective=38.954044, loss=38.973904, pen=0.001986
Iter 1600/2000, objective=38.900913, loss=38.916149, pen=0.001524
Iter 1700/2000, objective=38.858635, loss=38.870445, pen=0.001181
Iter 1800/2000, objective=38.824543, loss=38.833782, pen=0.000924
Iter 1900/2000, objective=38.796738, loss=38.804024, pen=0.000729
Iter 2000/2000, objective=38.773815, loss=38.779610, pen=0.000579
```

[7]:
```
plt.rcParams['font.family'] = 'Avenir Next'
plt.rcParams['font.weight'] = 200
plt.rcParams['font.size'] = 15
plt.rcParams['axes.titlesize'] = 16
plt.rcParams['axes.labelsize'] = 15
plt.rcParams['axes.titleweight'] = 200
plt.rcParams['axes.labelweight'] = 200
plt.rcParams['figure.dpi'] = 300
curve_color = '#DD4C4F'

plt.figure(figsize=(15, 5), facecolor='#ffffff')

plt.plot(objective, color=curve_color, linewidth=2)
plt.title('Objective Function', pad=15, color='#444444')
plt.xlabel('Step', color='#444444')
plt.ylabel('Objective', color='#444444')
plt.grid(True, alpha=0.2, color='#444444', linestyle='--')
plt.tick_params(colors='#444444')
```



1.3 Approach 2 - Iteration till convergence

[8]:
```
def iterative_convergence(
    beta_hat: np.array,
    excess_returns: np.array,
    N: int,
    K: int,
    R: int,
    T: int,
    rtol: float = 1e-05,
    atol: float = 1e-08,
    n_iter: int = 2000,
```

```
    seed: int = 0,
) -> tuple[np.array, np.array, np.array]:
    """
    Solves constrained optimization by iterating
    until convergence

    Args:
        beta_hat (np.array): estimated beta loadings, N * K * T
        excess_returns (np.array): excess returns, N * T
        N (int): number of assets
        K (int): number of observed assets
        R (int): number of unobserved factors
        T (int): number of time periods
        rtol (float): relative tolerance for convergence, refer to
            numpy.allclose documentation
        atol (float): absolute tolerance for convgernce
        n_iter (int): number of iterations
        seed (int): for reproducibility

    Returns:
        tuple[np.array, np.array, np.array, np.array]:
            eta: N * (1 + K)
            G: T * R
            beta^*: N * R
            objective: n_iter
    """

    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})"

    np.random.seed(seed)

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

    eta = np.random.normal(0, 0.1, size = (N, 1 + K))
    beta_star = np.random.normal(0, 0.1, size = (N, R))
    G = np.random.normal(0, 0.1, size = (T, R))

    objective = np.empty(shape=(n_iter))
```

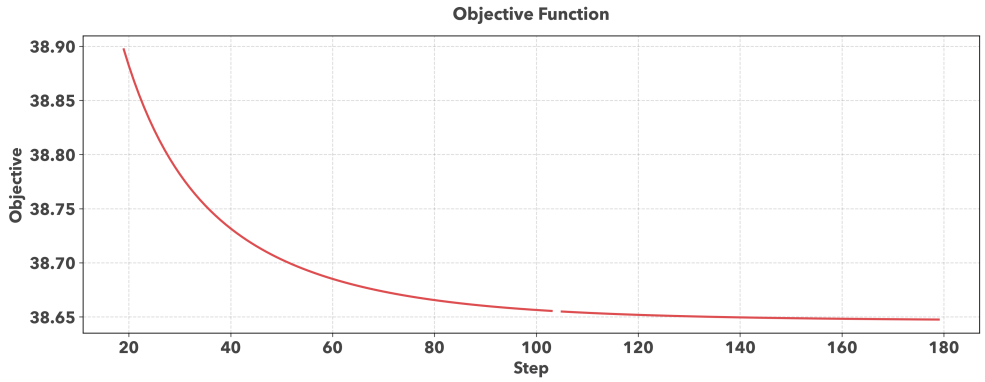
```
for i in range(N):
    Xi = X[i]               # (T, 1+K)
    ri = r[i]               # (T,)

    # (X'X)^{-1} X'r
    A = Xi.T @ Xi           # (1+K, 1+K)
    b = Xi.T @ ri           # (1+K,)

    A = A + 1e-8 * np.eye(1 + K)
    eta[i] = np.linalg.solve(A, b)

# U_i = r_i - X_i  _i
U = np.zeros((T, N))
for i in range(N):
    Xi = X[i]               # (T, 1+K)
    ri = r[i]               # (T,)
    ui = ri - Xi @ eta[i]   # (T,)
    U[:, i] = ui

# (1/NT) \sum_{i=1}^{N} (r_i - X eta) (r - X eta)'
S = (U @ U.T) / (N * T)      # (T, T)
eigvals, eigvecs = np.linalg.eigh(S)
G = eigvecs[:, -R:] * np.sqrt(T)            # (T, R)
I_T = np.eye(T)

for it in range(n_iter):
    G_old = G.copy()

    # M_G = I - G(G'G)^{-1}G'
    GtG = G.T @ G                    # (R, R)
    GtG_inv = np.linalg.inv(GtG)  # (R, R)
    Proj_G = G @ GtG_inv @ G.T       # (T, T)
    M_G = I_T - Proj_G

    # update eta and beta_star given G
    for i in range(N):
        Xi = X[i]               # (T, 1+K)
        ri = r[i]               # (T,)

        # eta = (X' M_G X)^{-1} X' M_G r
        A = Xi.T @ M_G @ Xi      # (1+K, 1+K)
        b = Xi.T @ M_G @ ri      # (1+K,)

        A = A + 1e-8 * np.eye(1 + K)
        eta[i] = np.linalg.solve(A, b)
```

```
            # calculate new residuals and beta_star
            vi = ri - Xi @ eta[i]              # (T,)
            beta_star[i] = GtG_inv @ (G.T @ vi)  # (R,)

        # update G
        U = np.zeros((T, N))
        for i in range(N):
            Xi = X[i]                   # (T, 1+K)
            ri = r[i]                   # (T,)
            ui = ri - Xi @ eta[i]
            U[:, i] = ui

        S = (U @ U.T) / (N * T)
        eigvals, eigvecs = np.linalg.eigh(S)
        G = eigvecs[:, -R:] * np.sqrt(T)

        # objective value
        U = np.zeros((T, N))
        for i in range(N):
            Xi = X[i]
            ri = r[i]
            U[:, i] = ri - Xi @ eta[i]

        # term_i[t] = u_i[t] - G[t] @ beta_star[i]
        loss_matrix = np.zeros((T, N))
        for i in range(N):
            loss_matrix[:, i] = U[:, i] - G @ beta_star[i]

        obj_value = np.mean(loss_matrix ** 2)

        objective[it] = obj_value.item()

        if (it + 1) % 100 == 0:
            loss = np.linalg.norm(G - G_old, ord="fro")
            log_str = (
                f"Iter {it + 1}/{n_iter}, "
                f"frobenius_norm(G - G_old)={loss:.4f}, "
                f"objective={obj_value:.6f}"
            )
            print(log_str)

        if np.allclose(G, G_old, rtol=rtol, atol=atol):
            print(f"Converged at iteration {it+1}")
            break

    return eta, G, beta_star, objective
```

[9]:
```
eta2, G2, bstar2, objective2 = iterative_convergence(
    beta_hat_np,
    excess_returns,
    N = 47,
    K = 5,
    R = 3,
    T = 206,
    n_iter=200
)
```

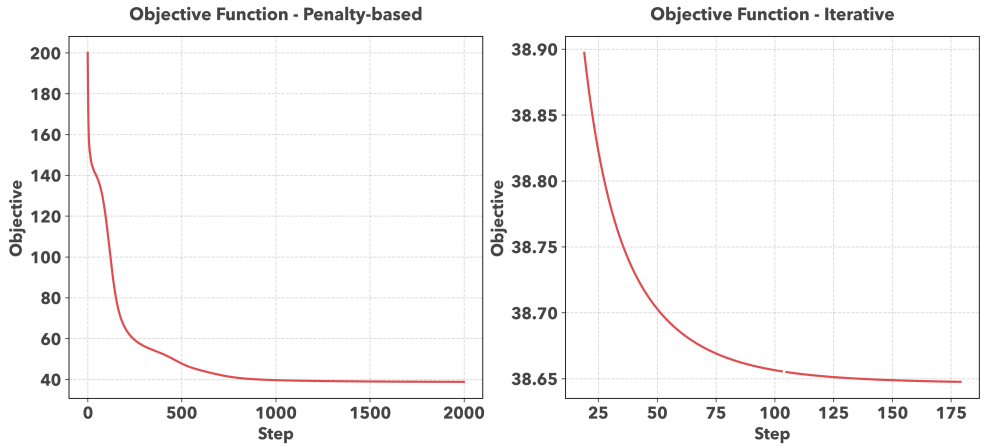
```
Iter 100/200, frobenius_norm(G - G_old)=0.0251, objective=38.656692
Iter 200/200, frobenius_norm(G - G_old)=0.0121, objective=38.647078
```

[10]:
```
plt.rcParams['font.family'] = 'Avenir Next'
plt.rcParams['font.weight'] = 200
plt.rcParams['font.size'] = 15
plt.rcParams['axes.titlesize'] = 16
plt.rcParams['axes.labelsize'] = 15
plt.rcParams['axes.titleweight'] = 200
plt.rcParams['axes.labelweight'] = 200
plt.rcParams['figure.dpi'] = 300
curve_color = '#DD4C4F'

plt.figure(figsize=(15, 5), facecolor='#ffffff')

plt.plot(
    np.where(
        (objective2 < np.nanpercentile(objective2, 10)) |
        (objective2 > np.nanpercentile(objective2, 90)),
        np.nan,
        objective2
    ),
    color=curve_color,
    linewidth=2
)
plt.title('Objective Function', pad=15, color='#444444')
plt.xlabel('Step', color='#444444')
plt.ylabel('Objective', color='#444444')
plt.grid(True, alpha=0.2, color='#444444', linestyle='--')
plt.tick_params(colors='#444444')
```



[39]:
```
plt.rcParams['font.family'] = 'Avenir Next'
plt.rcParams['font.weight'] = 200
plt.rcParams['font.size'] = 15
plt.rcParams['axes.titlesize'] = 16
plt.rcParams['axes.labelsize'] = 15
plt.rcParams['axes.titleweight'] = 200
plt.rcParams['axes.labelweight'] = 200
plt.rcParams['figure.dpi'] = 300
curve_color = '#DD4C4F'

fig, axes = plt.subplots(1, 2, figsize = (15, 6))
axes = np.ravel(axes)


axes[0].plot(objective, color=curve_color, linewidth=2)
axes[0].set_title('Objective Function - Penalty-based', pad=15, color='#444444')
axes[0].set_xlabel('Step', color='#444444')
axes[0].set_ylabel('Objective', color='#444444')
axes[0].grid(True, alpha=0.2, color='#444444', linestyle='--')
axes[0].tick_params(colors='#444444')


axes[1].plot(
    np.where(
        (objective2 < np.nanpercentile(objective2, 10)) |
        (objective2 > np.nanpercentile(objective2, 90)),
        np.nan,
        objective2
    ),
    color=curve_color,
    linewidth=2
)
```

```
axes[1].set_title('Objective Function - Iterative', pad=15, color='#444444')
axes[1].set_xlabel('Step', color='#444444')
axes[1].set_ylabel('Objective', color='#444444')
axes[1].grid(True, alpha=0.2, color='#444444', linestyle='--')
axes[1].tick_params(colors='#444444')
```



1.4 Validation of estimate

[17]:
```
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 the Galvao et al. model structure.
    Returns:
        beta_true      : (N, K, T)
        r              : (N, T)
        realized_cov   : (N, K, T)
        residuals      : (N, K, T)
        G_true         : (T, R)
        beta_star_true : (N, R)
        lambda_true    : (N, K)
    """
    rng = np.random.default_rng(seed)
```

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

    lambda_true = np.zeros((N, K))
    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))

    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_cov = beta_true + sigma_u * rng.normal(0, 1, size=(N, K, T))
    residuals = sigma_u * rng.normal(0, 1, size=(N, K, T))

    r = np.zeros((N, T))
    for i in range(N):
        for t in range(T):
            r[i, t] = (
                lambda_true[i] @ beta_true[i, :, t]
                + beta_star_true[i] @ G_true[t]
                + sigma_eps * rng.normal()
            )

    return beta_true, r, realized_cov, residuals, G_true, beta_star_true,␣
↪lambda_true
```

[ ]:
```
def build_true_eta(lambda_true):
    """
    lambda_true: (N, K)
    returns eta_true: (N, K+1)
    """
    N, K = lambda_true.shape
    eta_true = np.zeros((N, K+1))
    eta_true[:, 1:] = lambda_true
    return eta_true

def mc_compare_estimators(
    N=10, K=3, R=1, T=200,
    n_rep=50,
    heterogeneity_strength=0.5,
    seed=123,
    verbose=False
):
    rng = np.random.default_rng(seed)
```

```
eta_diff = []
rmse_pen_list = []
rmse_iter_list = []
obj_pen_final = []
obj_iter_final = []
obj_pen_all = []
obj_iter_all = []

for rep in range(n_rep):
    rep_seed = rng.integers(1_000_000_000)

    (
        beta_true,
        r,
        realized_cov,
        residuals,
        G_true,
        beta_star_true,
        lambda_true
    ) = simulate_dgp(
        N=N, K=K, R=R, T=T,
        heterogeneity_strength=heterogeneity_strength,
        seed=rep_seed
    )

    eta_true = build_true_eta(lambda_true)
    beta_hat = beta_true

    eta_pen, G_pen, beta_star_pen, obj_pen = penalty_based_minimization(
        beta_hat, r, N, K, R, T,
        lam=1.0, lr=1e-2, n_iter=1000, seed=rep_seed
    )
    obj_pen_all.append(obj_pen)

    eta_iter, G_iter, beta_star_iter, obj_iter = iterative_convergence(
        beta_hat, r, N, K, R, T,
        n_iter=1000, seed=rep_seed
    )
    obj_iter_all.append(obj_iter)

    diff = np.linalg.norm(eta_pen - eta_iter) / np.sqrt(N*(K+1))
    eta_diff.append(diff)

    rmse_pen = np.linalg.norm(eta_pen - eta_true) / np.sqrt(N*(K+1))
    rmse_iter = np.linalg.norm(eta_iter - eta_true) / np.sqrt(N*(K+1))
```

```
        rmse_pen_list.append(rmse_pen)
        rmse_iter_list.append(rmse_iter)

        obj_pen_final.append(obj_pen[-1])
        obj_iter_final.append(obj_iter[-1])

        if verbose:
            print(
                f"[{rep+1}/{n_rep}] diff={diff:.4f}, "
                f"RMSE_pen={rmse_pen:.4f}, RMSE_iter={rmse_iter:.4f}"
            )

    obj_pen_all = np.vstack(obj_pen_all)
    obj_iter_all = np.vstack(obj_iter_all)

    return {
        "eta_diff": np.array(eta_diff),
        "rmse_pen": np.array(rmse_pen_list),
        "rmse_iter": np.array(rmse_iter_list),
        "obj_pen_final": np.array(obj_pen_final),
        "obj_iter_final": np.array(obj_iter_final),
        "obj_pen_all": obj_pen_all,
        "obj_iter_all": obj_iter_all,
    }
```

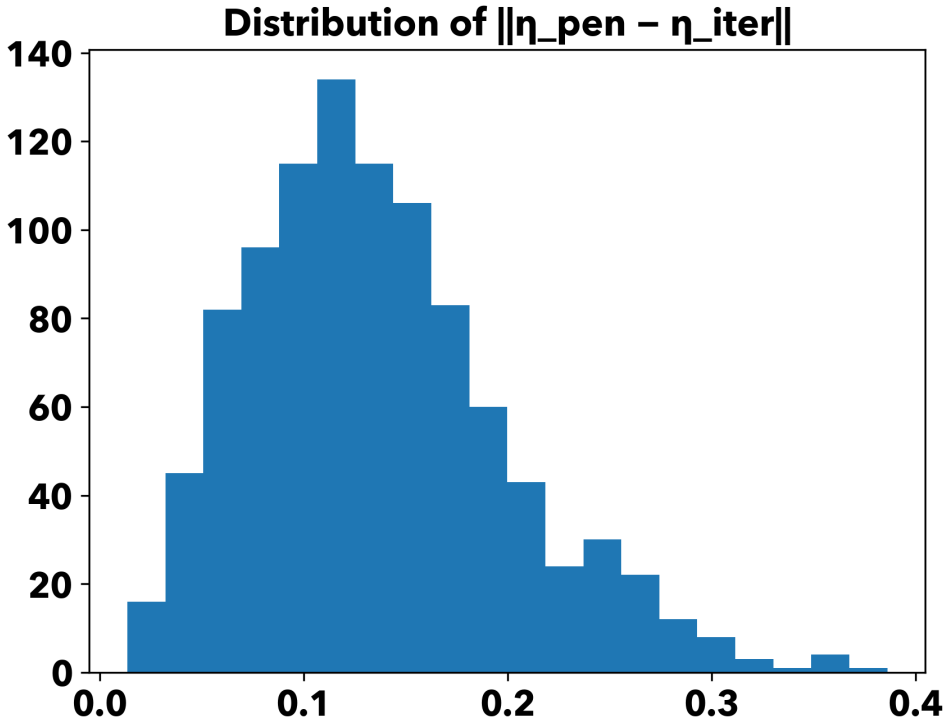
[43]:
```
def summarize_mc_results(results):
    print("Mean || _pen -  _iter||:", results["eta_diff"].mean())
    print("Std  || _pen -  _iter||:", results["eta_diff"].std())

    print("\nRMSE vs true  :")
    print("Penalty estimator mean RMSE:", results["rmse_pen"].mean())
    print("Iterative estimator mean RMSE:", results["rmse_iter"].mean())

    plt.hist(results["eta_diff"], bins=20)
    plt.title("Distribution of || _pen -  _iter||")
    plt.show()

    plt.hist(results["rmse_pen"], alpha=0.6, bins=20, label="Penalty")
    plt.hist(results["rmse_iter"], alpha=0.6, bins=20, label="Iterative")
    plt.legend()
    plt.title("RMSE( vs  _true)")
    plt.show()
```

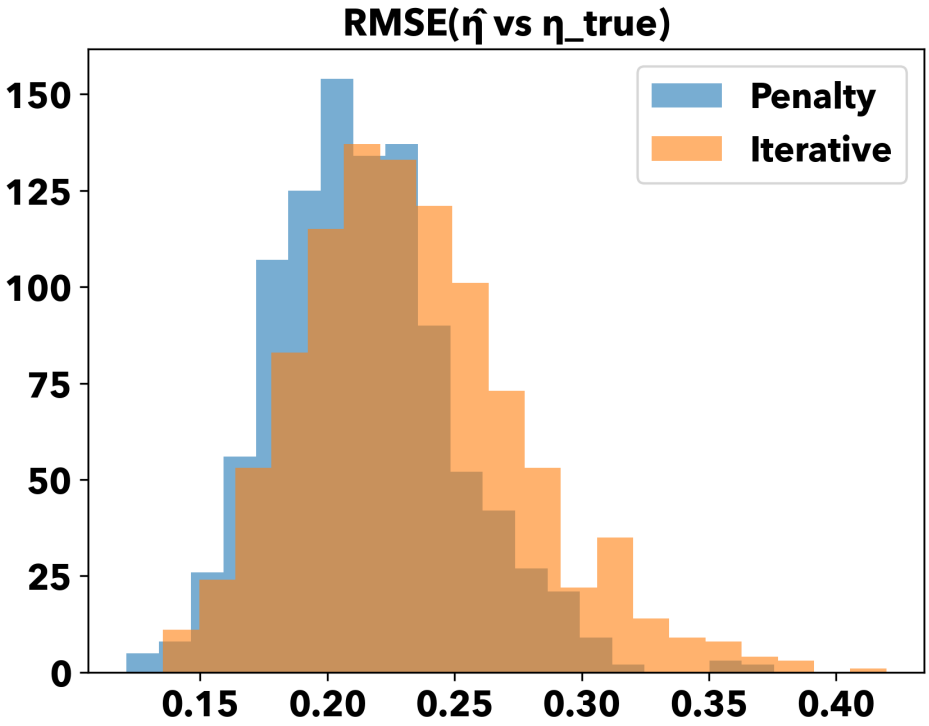
[ ]:
```
results = mc_compare_estimators(
    N=10, K=3, R=1, T=200,
    n_rep=1000,
    heterogeneity_strength=0.5,
```

```
    verbose=True
)
```

[46]:
```
summarize_mc_results(results)
```

```
Mean || _pen -  _iter||: 0.13549610740681772
Std  || _pen -  _iter||: 0.06303714782188485

RMSE vs true  :
Penalty estimator mean RMSE: 0.21490165579445744
Iterative estimator mean RMSE: 0.23338957450207307
```





[47]:
```
mean_diff = results["eta_diff"].mean()
std_diff = results["eta_diff"].std()

rmse_pen = results["rmse_pen"].mean()
rmse_iter = results["rmse_iter"].mean()

summary_df = pd.DataFrame({
    "Metric": [
        "Mean || _pen -  _iter||",
        "Std  || _pen -  _iter||",
        "RMSE (Penalty Estimator)",
        "RMSE (Iterative Estimator)"
    ],
    "Value": [
        mean_diff,
        std_diff,
        rmse_pen,
        rmse_iter
    ]
})
```

```
summary_df
```

[47]:

	Metric	Value
0	Mean || _pen - _iter||	0.135496
1	Std || _pen - _iter||	0.063037
2	RMSE (Penalty Estimator)	0.214902
3	RMSE (Iterative Estimator)	0.233390

[ ]:
```
latex_table = summary_df.round(3).to_latex(
    index=False,
    float_format="%.3f",
    caption="Monte Carlo Comparison of Penalty-Based and Iterative Estimators",
    label="tab:mc_eta_comparison",
    escape=False
)

print(latex_table)
```

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,
```

```
) -> 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)

# 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
```

```
    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
```

```
        - 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]:
```
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(
```

```
    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
```

05_monte_carlo

December 18, 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
from scipy.stats import kstest
import scipy.stats as stats
import seaborn as sns
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)

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

    alpha = np.zeros(N)
    lambda_true = np.zeros((N, K))

    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))

    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_cov = beta_true + sigma_u * rng.normal(0,1,size=(N,K,T))

    residuals = sigma_u * rng.normal(0,1,size=(N,K,T))

    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, lambda_true
```

[3]:
```
def run_mc(N, K, R, T, n_rep=200, verbose=False, heterogeneity_strength = 0):
    """
    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)
    """

    gamma_a_lambda = np.empty_like(np.arange(n_rep))
    gamma_a = np.empty_like(np.arange(n_rep))
    gamma_lambda = np.empty_like(np.arange(n_rep))
```

```
    for rep in range(n_rep):
        if verbose and (rep % 100) == 0:
            print(f"Processing {rep} out of {n_rep}")
        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=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,
        )

        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)

        gamma_a_lambda[rep] = full
        gamma_a[rep] = inta
        gamma_lambda[rep] = slope

    return {
        "gamma_a_lambda": gamma_a_lambda,
        "gamma_a": gamma_a,
        "gamma_lambda": gamma_lambda,
    }
```

[4]:
```
alpha = 0.05
n_rep = 1000
print("SIZE TEST RESULTS:")

results = run_mc(N=20, K=3, R=1, T=200, n_rep=n_rep, verbose= True)
```

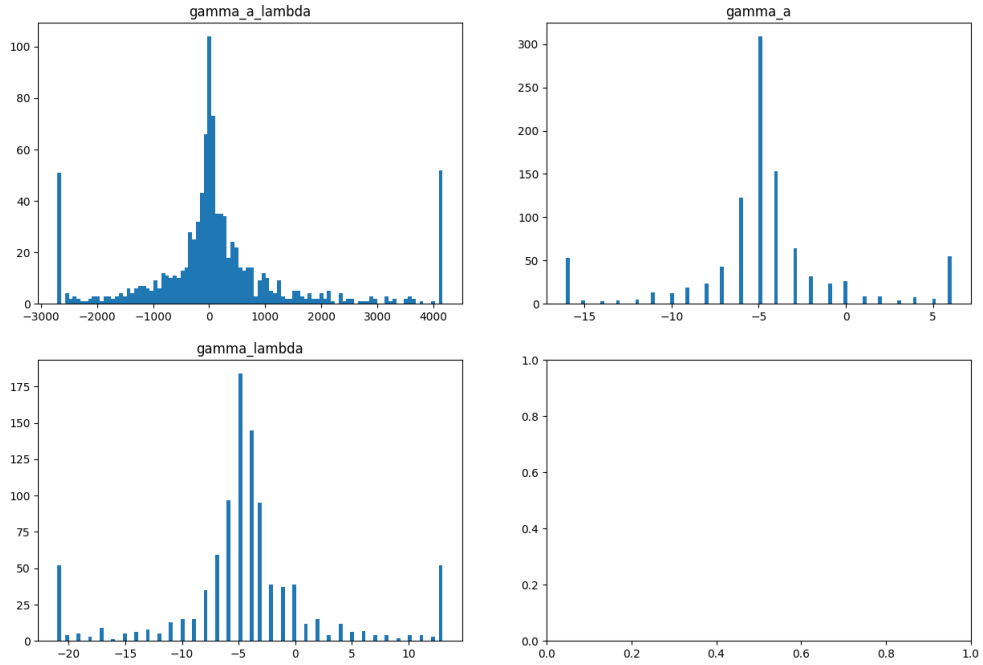
```
SIZE TEST RESULTS:
Processing 0 out of 1000
Processing 100 out of 1000
Processing 200 out of 1000
```

```
Processing 300 out of 1000
Processing 400 out of 1000
Processing 500 out of 1000
Processing 600 out of 1000
Processing 700 out of 1000
Processing 800 out of 1000
Processing 900 out of 1000
```

[5]:
```
critical = norm.ppf(1 - alpha/2)
print(f"Type I error rate - joint: ")
print(f"{(np.abs(results['gamma_a_lambda']) > critical).sum() / n_rep:.4f}")

print(f"Type I error rate - intercept: ")
print(f"{(np.abs(results['gamma_a']) > critical).sum() / n_rep:.4f}")

print(f"Type I error rate - slope: ")
print(f"{(np.abs(results['gamma_lambda']) > critical).sum() / n_rep:.4f}")
```

```
Type I error rate - joint:
0.9960
Type I error rate - intercept:
0.9420
Type I error rate - slope:
0.9120
```

[6]:
```
fig, axes = plt.subplots(2, 2, figsize = (15, 10))
axes = np.ravel(axes)

for i, test_statistic in enumerate(['gamma_a_lambda', 'gamma_a',␣
 ↪'gamma_lambda']):
    axes[i].hist(
        utils.clean(results[test_statistic]),
        bins = 100,
    )
    axes[i].set_title(test_statistic)
```

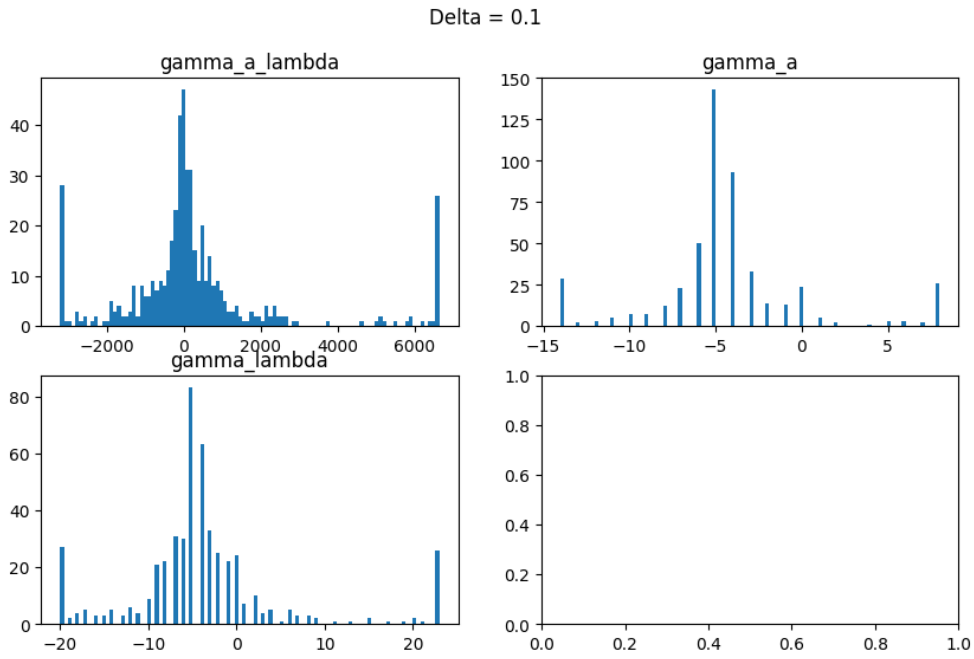


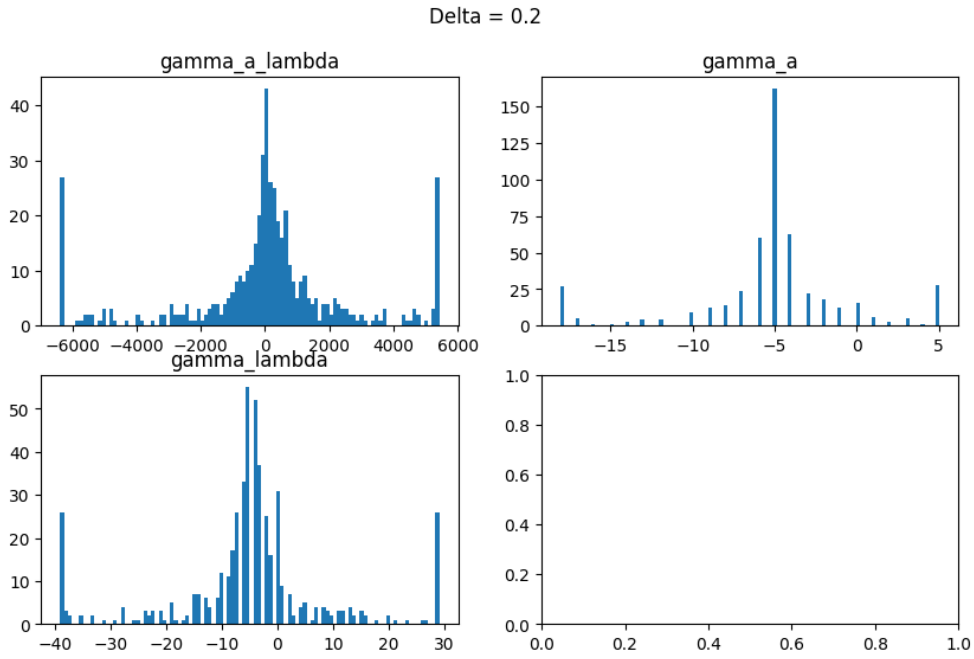
[7]:
```
power_results = {}
for delta in [0.1, 0.2, 0.5, 1.0]:

    print("=====================================================================")
    print(f"POWER for heterogeneity_strength={delta}:")

    local_results = run_mc(N=20, K=3, R=1, T=200,
                    heterogeneity_strength=delta,
                    n_rep=500,
                    verbose = True)

    power_results[delta]= local_results

    critical = norm.ppf(1 - alpha/2)
    print(f"Power - joint")
    print(f"{(np.abs(results['gamma_a_lambda']) > critical).sum() / n_rep:.4f}")

    print(f"Power - intercept")
    print(f"{(np.abs(results['gamma_a']) > critical).sum() / n_rep:.4f}")

    print(f"Power - slope")
    print(f"{(np.abs(results['gamma_lambda']) > critical).sum() / n_rep:.4f}")
    print("=====================================================================")
```

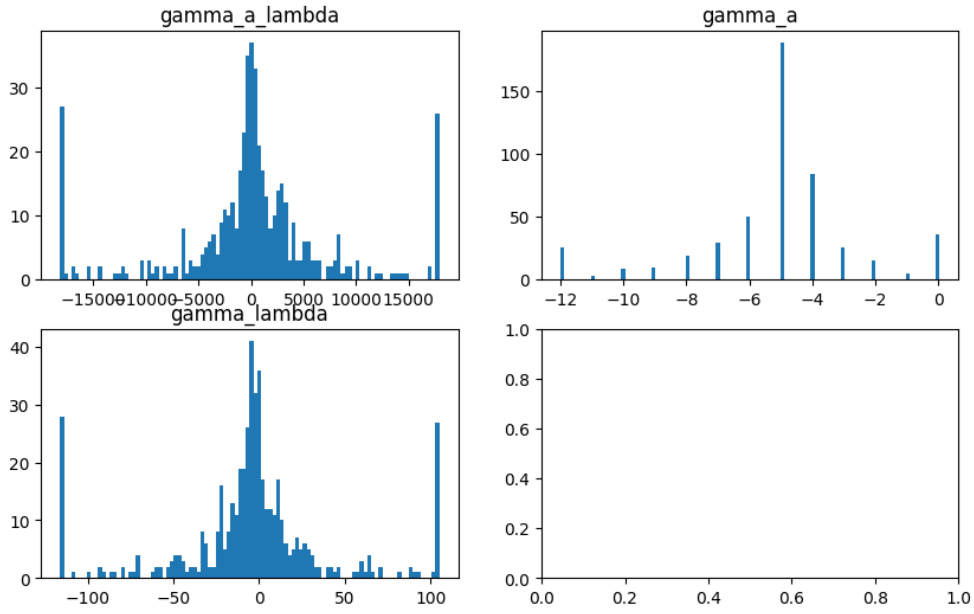
```
======================================================================
POWER for heterogeneity_strength=0.1:
Processing 0 out of 500
Processing 100 out of 500
Processing 200 out of 500
Processing 300 out of 500
Processing 400 out of 500
Power - joint
0.9960
Power - intercept
0.9420
Power - slope
0.9120
======================================================================
======================================================================
POWER for heterogeneity_strength=0.2:
Processing 0 out of 500
Processing 100 out of 500
Processing 200 out of 500
Processing 300 out of 500
Processing 400 out of 500
Power - joint
0.9960
Power - intercept
0.9420
Power - slope
0.9120
======================================================================
======================================================================
POWER for heterogeneity_strength=0.5:
Processing 0 out of 500
Processing 100 out of 500
Processing 200 out of 500
Processing 300 out of 500
Processing 400 out of 500
Power - joint
0.9960
Power - intercept
0.9420
Power - slope
0.9120
======================================================================
======================================================================
POWER for heterogeneity_strength=1.0:
Processing 0 out of 500
Processing 100 out of 500
```

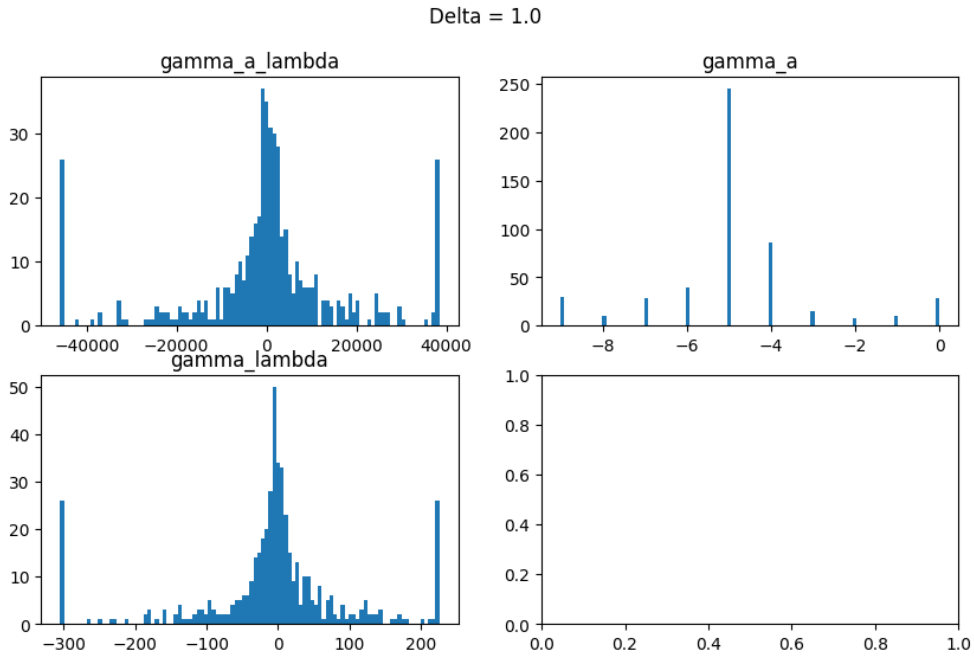
```
Processing 200 out of 500
Processing 300 out of 500
Processing 400 out of 500
Power - joint
0.9960
Power - intercept
0.9420
Power - slope
0.9120
========================================================================
```

[8]:
```
for delta in [0.1, 0.2, 0.5, 1.0]:
    fig, axes = plt.subplots(2, 2, figsize = (10, 6))
    axes = np.ravel(axes)

    for i, test_statistic in enumerate(['gamma_a_lambda', 'gamma_a',␣
↪'gamma_lambda']):
        axes[i].hist(
            utils.clean(power_results[delta][test_statistic]),
            bins = 100,
        )
        axes[i].set_title(test_statistic)
    fig.suptitle(f"Delta = {delta}")
```




Delta = 0.2
gamma_a_lambda
gamma_a
gamma_lambda


Delta = 0.5
gamma_a_lambda
gamma_a
gamma_lambda



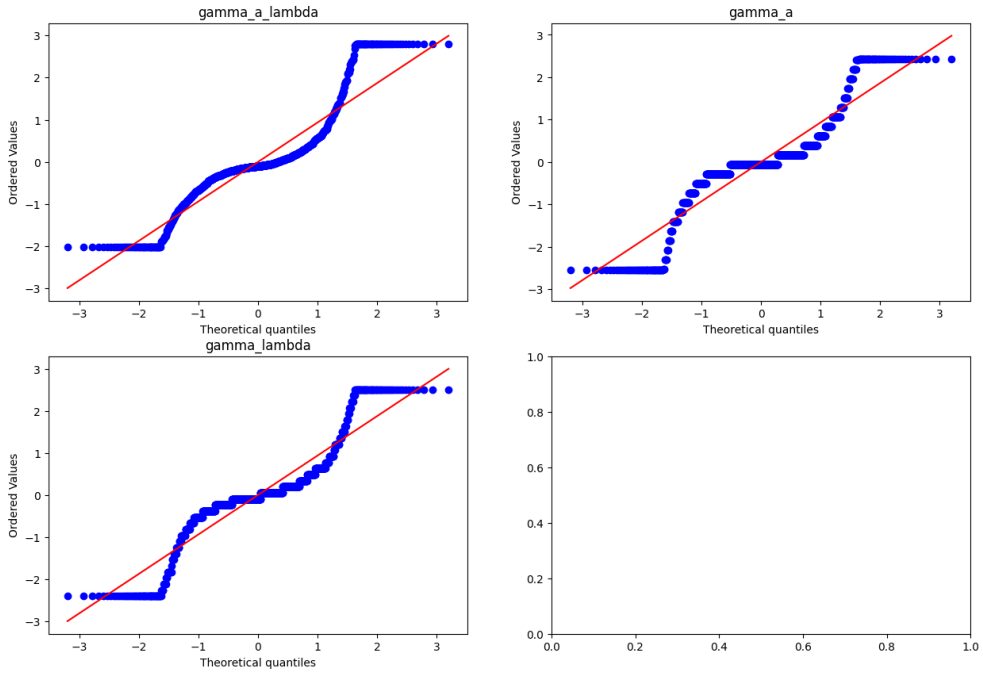
We examine the distribution of the test statistic under null

[9]:
```
print("MC parameters")
print("N=20")
print("K=3")
print("R=1")
print("T=200")
for k, v in results.items():
    print(f"==========" * 5)
    print(k)
    print(f"mean: {v.mean()}")
    print(f"std: {v.std()}")
    print(f"min: {v.min()}")
    print(f"max: {v.max()}")
```

```
MC parameters
N=20
K=3
R=1
T=200
==================================================
gamma_a_lambda
mean: 420.043
std: 24151.585248657095
```

```
min: -507281
max: 422577
==================================================
gamma_a
mean: -8.414
std: 75.15964744462285
min: -1923
max: 109
==================================================
gamma_lambda
mean: -0.58
std: 168.72552148385856
min: -1709
max: 4320
```

[10]:
```
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
axes = np.ravel(axes)

for i, test_statistic in enumerate(['gamma_a_lambda', 'gamma_a',␣
 ↪'gamma_lambda']):
    vals = utils.clean(results[test_statistic])
    z_vals = (vals- vals.mean()) / vals.std()
    stats.probplot(z_vals, dist="norm", plot=axes[i])
    axes[i].set_title(test_statistic)
```



1.1 Asymptotic properties of estimator

[ ]:
```python
def run_mc_eta_properties(
    N=20, K=3, R=1, T=200,
    n_rep=200,
    heterogeneity_strength=0.5,
    sigma_u=0.2,
    sigma_eps=0.5,
    sigma_g=1.0,
    seed=0
):
    """
    Monte Carlo experiment for sampling properties of the eta estimator.
    Returns:
        eta_hats: (n_rep, N, K+1)
        avars:    (n_rep, N*(K+1), N*(K+1))
        lambda_true: (N, K)
    """

    rng = np.random.default_rng(seed)

    p = N * (K+1)

    eta_hats = np.zeros((n_rep, N, K+1))
    avars    = np.zeros((n_rep, p, p))
    lambda_store = None  # to save true lambda

    for rep in range(n_rep):
        if(rep % 10) == 0:
            print(f"Processing rep {rep}")
        rep_seed = rng.integers(1, 1_000_000_000)

        (
            beta_true,          # (N, K, T)
            r,                  # (N, T)
            realized_cov,       # (N, K, T)
            residuals,          # (N, K, T)
            G_true,             # (T, R)
            beta_star_true,     # (N, R)
            lambda_true         # (N, K)
        ) = simulate_dgp(
            N=N, K=K, R=R, T=T,
            heterogeneity_strength=heterogeneity_strength,
            sigma_u=sigma_u,
            sigma_eps=sigma_eps,
```

```
            sigma_g=sigma_g,
            seed=rep_seed
        )

        if lambda_store is None:
            lambda_store = lambda_true.copy()

        beta_hat = beta_true

        # Estimate eta
        eta_hat, G_hat, beta_star_hat, objvals = utils.iterative_convergence(
            beta_hat=beta_hat,
            excess_returns=r,
            N=N, K=K, R=R, T=T,
            n_iter=2000
        )

        # Estimate AVAR
        avar_hat = 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
        )

        eta_hats[rep] = eta_hat
        avars[rep] = avar_hat

    return eta_hats, avars, lambda_store

def analyze_eta_bias_variance(eta_hats, lambda_true, N, K, T):
    """
    eta_hats: (n_rep, N, K+1)
    lambda_true: (N, K)
    """
    n_rep = eta_hats.shape[0]
    Kp1 = K+1
    p = N*Kp1

    # build eta_true array
    eta_true = np.zeros((N, K+1))
    eta_true[:,0] = 0.0
    eta_true[:,1:] = lambda_true
```

```
    eta_true_vec = eta_true.reshape(p)

    # empirical mean
    eta_bar = eta_hats.mean(axis=0)
    eta_bias = eta_bar - eta_true

    print("\n==== BIAS OF ETA ESTIMATOR ====")
    print("Max abs bias:", np.max(np.abs(eta_bias)))
    print("Mean abs bias:", np.mean(np.abs(eta_bias)))

    eta_vec = eta_hats.reshape(n_rep, p)
    eta_centered = eta_vec - eta_true_vec

    emp_cov = (eta_centered.T @ eta_centered) / (n_rep - 1)

    return eta_true, eta_true_vec, emp_cov

def compare_empirical_to_avar(emp_cov, avars, T, N, K):
    Kp1 = K + 1
    p = N*Kp1

    avar_mean = avars.mean(axis=0)
    emp_var = np.diag(emp_cov)
    theo_var = np.diag(avar_mean) / T  # theoretical scaled variance

    print("\n==== VARIANCE COMPARISON ====")
    print("Empirical variance (first 10):", emp_var[:10])
    print("Theoretical variance (first 10):", theo_var[:10])

    return emp_var, theo_var


def check_z_normality(eta_hats, avars, eta_true_vec, T, param_index):
    """
    param_index: index in vec(eta) to test (0 to p-1)
    """

    n_rep = eta_hats.shape[0]
    p = eta_true_vec.shape[0]

    z_vals = []

    for rep in range(n_rep):
        eta_vec = eta_hats[rep].reshape(p)
        avar_hat = avars[rep]
```

```
        se = np.sqrt(avar_hat[param_index, param_index] / T)
        z = (eta_vec[param_index] - eta_true_vec[param_index]) / se

        z_vals.append(z)

    z_vals = np.array(z_vals)

    print("\n==== NORMALITY CHECK ====")
    print("mean(z):", z_vals.mean())
    print("std(z):", z_vals.std())
    print("KS test vs N(0,1):", kstest(z_vals, 'norm'))

    return z_vals
```

[ ]:
```
def coverage_probability(eta_hats, avars, eta_true_vec, param_index, T, alpha=0.
↪05):
    zcrit = norm.ppf(1 - alpha/2)

    n_rep = eta_hats.shape[0]
    p = len(eta_true_vec)
    covered = 0

    for rep in range(n_rep):
        eta_vec = eta_hats[rep].reshape(p)
        avar_hat = avars[rep]
        se = np.sqrt(avar_hat[param_index, param_index] / T)

        ci_low  = eta_vec[param_index] - zcrit*se
        ci_high = eta_vec[param_index] + zcrit*se

        if (eta_true_vec[param_index] >= ci_low) and (eta_true_vec[param_index]␣
↪<= ci_high):
            covered += 1

    coverage = covered / n_rep
    print("\n==== COVERAGE ====")
    print(f"Coverage probability: {coverage:.3f} (target = 0.95)")
    return coverage
```

[ ]:
```
eta_hats, avars, lambda_true = run_mc_eta_properties(
    N=20, K=3, R=1, T=200,
    heterogeneity_strength=0.5,
    n_rep=500
)

eta_true, eta_true_vec, emp_cov = analyze_eta_bias_variance(
    eta_hats, lambda_true, N=20, K=3, T=200
```

```
)

emp_var, theo_var = compare_empirical_to_avar(
    emp_cov, avars, T=200, N=20, K=3
)

Kp1 = 3 + 1
param_index = 0*Kp1 + 1  # (asset 0, slope 1)

z_vals = check_z_normality(
    eta_hats, avars, eta_true_vec, T=200, param_index=param_index
)

coverage = coverage_probability(
    eta_hats, avars, eta_true_vec, param_index=param_index, T=200
)
```

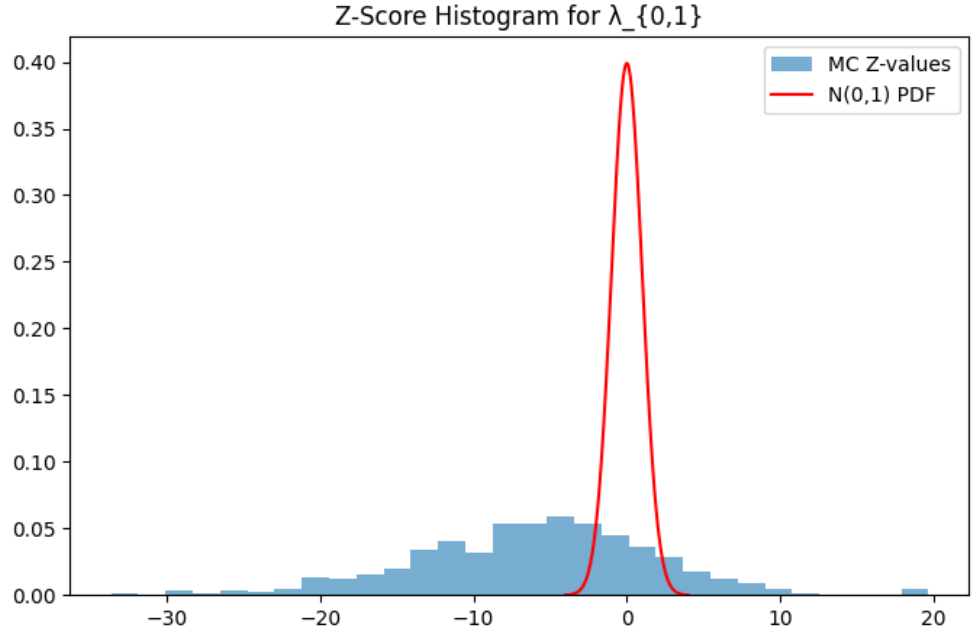
```
==== BIAS OF ETA ESTIMATOR ====
Max abs bias: 1.6593719669770914
Mean abs bias: 0.3159582355355777

==== VARIANCE COMPARISON ====
Empirical variance (first 10): [0.09689915 0.50258872 0.34710584 0.27115956
0.11882595 0.58212828
 1.82794002 0.42457894 0.1129175  0.3213619 ]
Theoretical variance (first 10): [0.00713551 0.00715691 0.00715691 0.00715691
0.00714484 0.00715691
 0.00714225 0.00715691 0.00714669 0.00715691]

==== NORMALITY CHECK ====
mean(z): -5.956843693296206
std(z): 7.766782441554891
KS test vs N(0,1): KstestResult(statistic=np.float64(0.6739886744018414),
pvalue=np.float64(4.9638594541501135e-225),
statistic_location=np.float64(-1.7278080080687428), statistic_sign=np.int8(1))

==== COVERAGE ====
Coverage probability: 0.168 (target = 0.95)
```

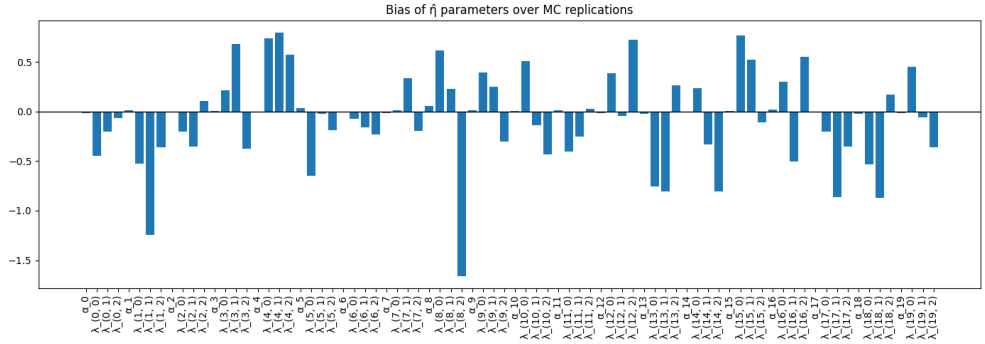
[ ]:
```
def param_labels(N, K):
    labels = []
    for i in range(N):
        labels.append(f" _{i}")
        for k in range(K):
            labels.append(f" _{i,k}")
    return labels
```

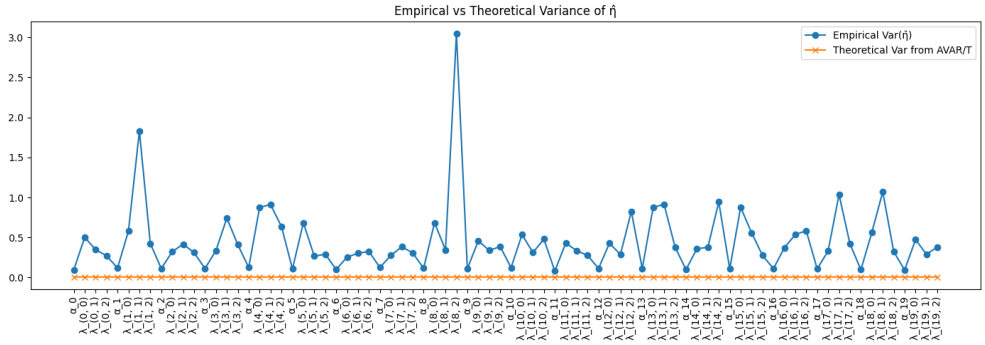
```python
def plot_eta_bias(eta_hats, eta_true, N, K):
    labels = param_labels(N, K)
    p = len(labels)

    eta_mean = eta_hats.mean(axis=0).reshape(-1)
    eta_true_vec = eta_true.reshape(-1)

    bias = eta_mean - eta_true_vec

    plt.figure(figsize=(14,5))
    plt.bar(range(p), bias)
    plt.axhline(0, color="black", linewidth=1)
    plt.xticks(range(p), labels, rotation=90)
    plt.title("Bias of ^ parameters over MC replications")
    plt.tight_layout()
    plt.show()

def plot_variance_comparison(emp_cov, avars, T, N, K):
    labels = param_labels(N, K)
    p = len(labels)

    emp_var = np.diag(emp_cov)
    avar_mean = avars.mean(axis=0)
    theo_var = np.diag(avar_mean) / T

    plt.figure(figsize=(14,5))
    plt.plot(emp_var, label="Empirical Var(^)", marker='o')
    plt.plot(theo_var, label="Theoretical Var from AVAR/T", marker='x')
    plt.xticks(range(p), labels, rotation=90)
    plt.legend()
    plt.title("Empirical vs Theoretical Variance of ^")
    plt.tight_layout()
    plt.show()

def plot_z_histogram(z_vals, param_label):
    plt.figure(figsize=(8,5))
    plt.hist(z_vals, bins=30, density=True, alpha=0.6, label="MC Z-values")

    x = np.linspace(-4,4,200)
    plt.plot(x, norm.pdf(x), color="red", label="N(0,1) PDF")

    plt.title(f"Z-Score Histogram for {param_label}")
    plt.legend()
    plt.show()

def plot_qq(z_vals, param_label):
```

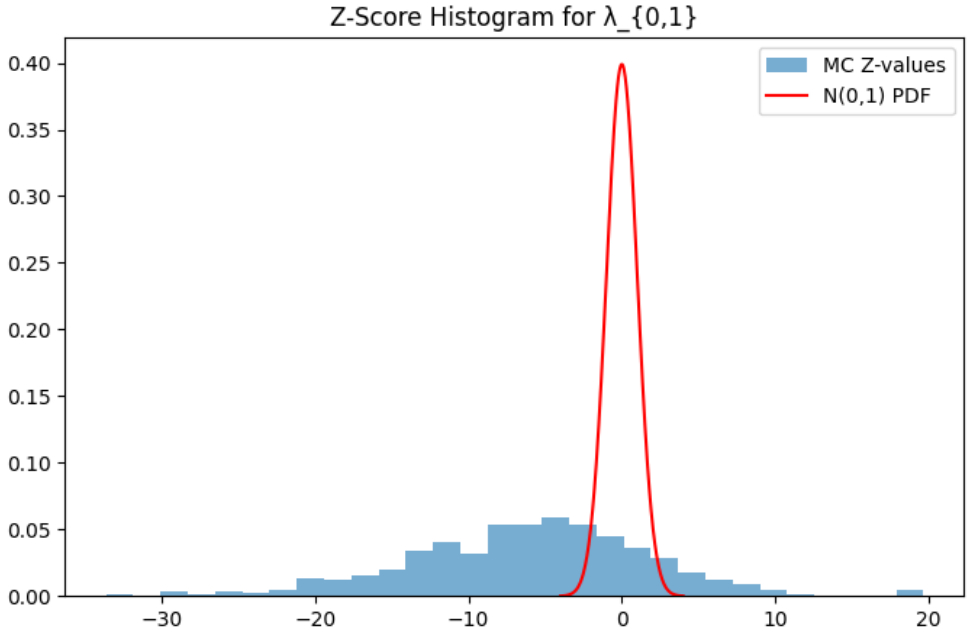
```
    plt.figure(figsize=(6,6))
    stats.probplot(z_vals, dist="norm", plot=plt)
    plt.title(f"QQ Plot for Z-scores: {param_label}")
    plt.show()

def plot_coverage(coverage, param_label):
    plt.figure(figsize=(5,4))
    plt.bar([0], [coverage], width=0.4)
    plt.axhline(0.95, color="red", linestyle="--", label="95% target")
    plt.ylim(0,1)
    plt.xticks([0], [param_label])
    plt.ylabel("Coverage probability")
    plt.legend()
    plt.title("Empirical 95% CI Coverage")
    plt.show()

def plot_eta_heatmap(eta_hats, eta_true):
    error = eta_hats.mean(axis=0) - eta_true  # (N, K+1)

    plt.figure(figsize=(10,6))
    sns.heatmap(np.abs(error), cmap="magma", annot=False)
    plt.title("Mean Absolute Error Heatmap of ^")
    plt.xlabel("Parameter ( ,  1,..., K)")
    plt.ylabel("Industry index")
    plt.show()

def plot_eta_boxplots(eta_hats, eta_true, N, K):
    labels = param_labels(N, K)
    p = len(labels)

    plt.figure(figsize=(15,6))
    plt.boxplot(eta_hats.reshape(eta_hats.shape[0], -1), labels=labels)
    plt.axhline(0, color="black")
    plt.title("Distribution of ^ Across Monte Carlo Replications")
    plt.xticks(rotation=90)
    plt.tight_layout()
    plt.show()
```

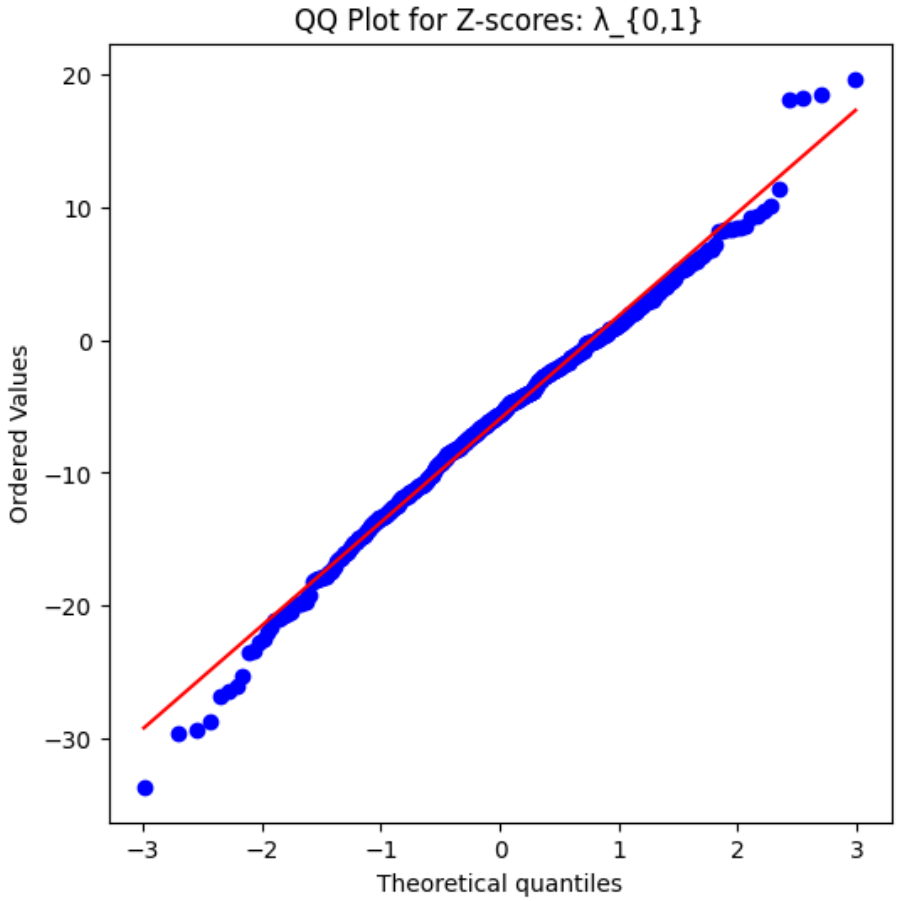
[ ]:
```
param_label = " _{0,1}"    # Adjust depending on param_index
plot_z_histogram(z_vals, param_label)
plot_eta_bias(eta_hats, eta_true, N=20, K=3)
plot_variance_comparison(emp_cov, avars, T=200, N=20, K=3)
plot_z_histogram(z_vals, param_label)
plot_qq(z_vals, param_label)
plot_coverage(coverage, param_label)
plot_eta_heatmap(eta_hats, eta_true)
plot_eta_boxplots(eta_hats, eta_true, N=20, K=3)
```

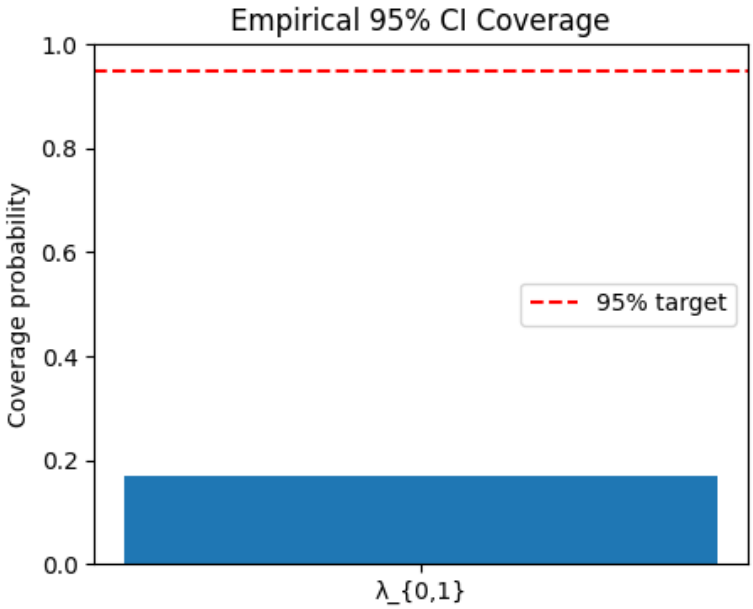

Z-Score Histogram for λ_{0,1}
MC Z-values
N(0,1) PDF
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−30
−20
−10
0
10
20

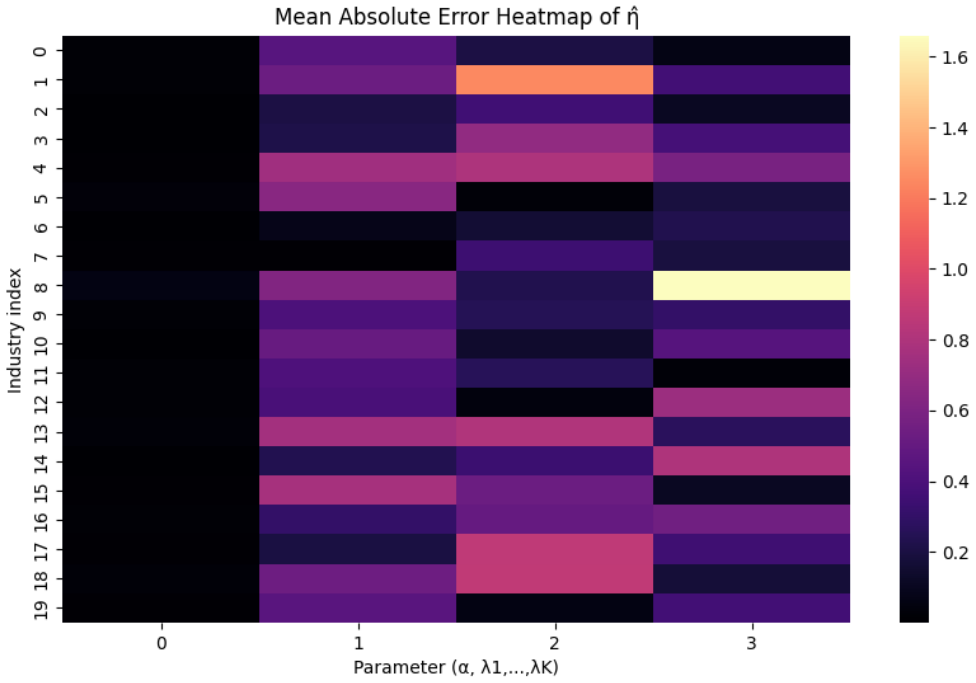

Bias of η̂ parameters over MC replications
0.5
0.0
−0.5
−1.0
−1.5


Empirical vs Theoretical Variance of ĥ
Empirical Var(ĥ)
Theoretical Var from AVAR/T
3.0
2.5
2.0
1.5
1.0
0.5
0.0

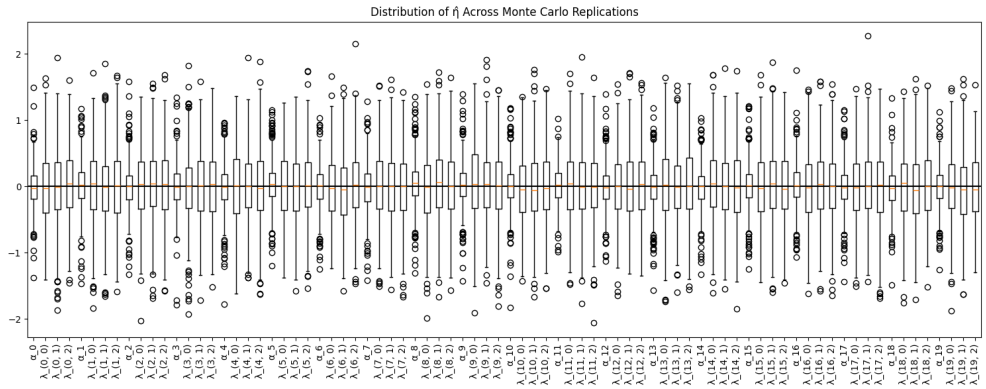

Z-Score Histogram for λ_{0,1}
MC Z-values
N(0,1) PDF
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−30
−20
−10
0
10
20


QQ Plot for Z-scores: λ_{0,1}
Ordered Values
Theoretical quantiles
20
10
0
−10
−20
−30
−3
−2
−1
0
1
2
3


Empirical 95% CI Coverage
Coverage probability
1.0
0.8
0.6
0.4
0.2
0.0
95% target
λ_{0,1}


Mean Absolute Error Heatmap of η̂
Industry index
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0
1
2
3
Parameter (α, λ1,...,λK)
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2

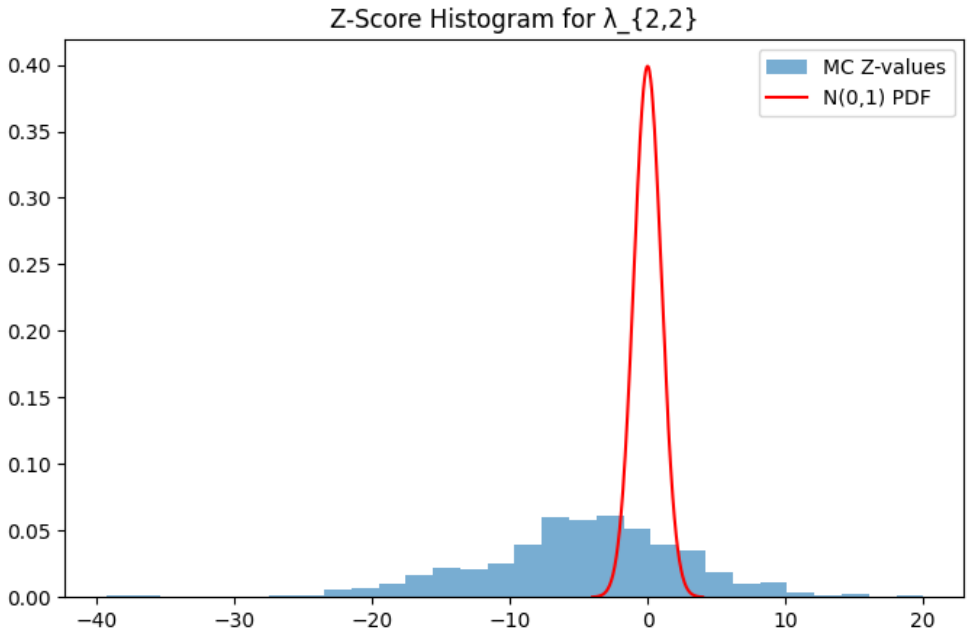
```
/var/folders/sp/cw_2m19j25xbvgpdjz48gclh0000gn/T/ipykernel_22666/4028456446.py:8
6: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been
renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be
dropped in 3.11.
  plt.boxplot(eta_hats.reshape(eta_hats.shape[0], -1), labels=labels)
```

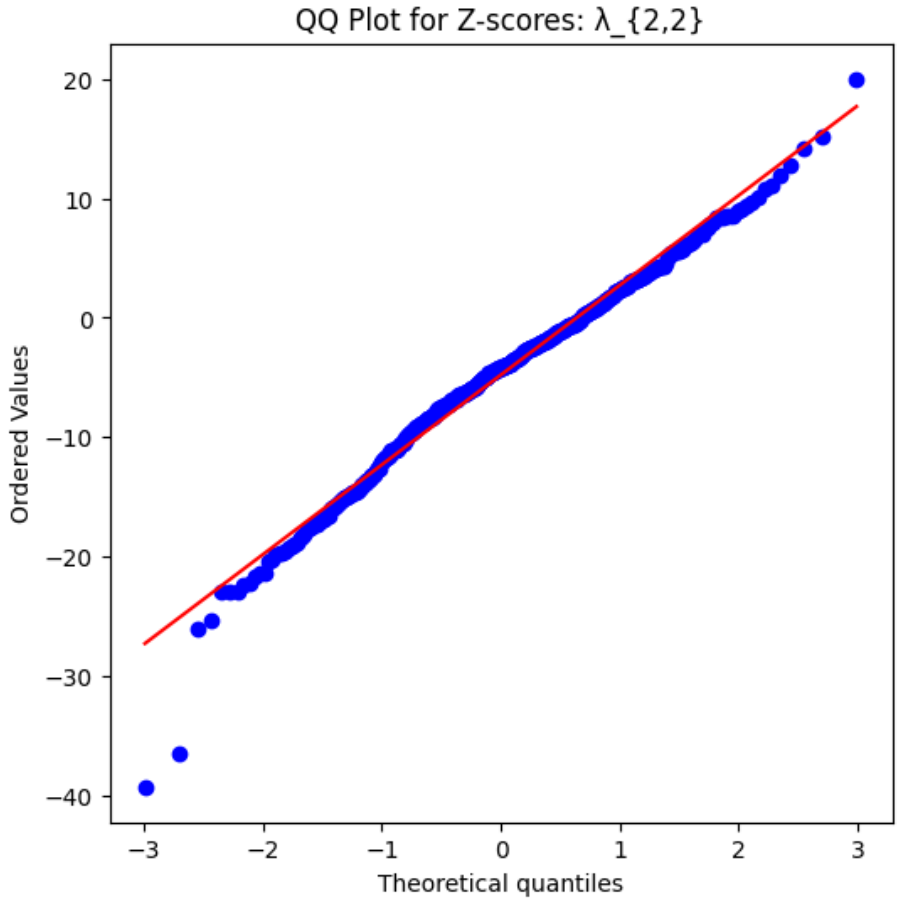


[ ]:
```python
# pick asset i and slope index j
i = 2    # asset 0
j = 2    # slope for factor 1 (assuming j=1..K)

def get_param_index(i, j, K):
    """
    i = industry index (0..N-1)
    j = 0 for intercept, 1..K for slopes
        j = 0 → intercept _i
        j >= 1 → slope _{i,j-1}

    K = number of factors
    returns index in vec(eta)
    """
    Kp1 = K + 1
    return i * Kp1 + j

param_index = get_param_index(i, j, K=3)
param_label = f" _{{{i},{j}}}"

z_vals = check_z_normality(
    eta_hats, avars, eta_true_vec, T=200, param_index=param_index
)

coverage_probability(
    eta_hats, avars, eta_true_vec, param_index=param_index, T=200
```

```
)

plot_z_histogram(z_vals, param_label)
plot_qq(z_vals, param_label)
```

```
==== NORMALITY CHECK ====
mean(z): -4.804067060910951
std(z): 7.551418289678208
KS test vs N(0,1): KstestResult(statistic=np.float64(0.6242362685243527),
pvalue=np.float64(1.0800520439165965e-188),
statistic_location=np.float64(-1.8842773985696188), statistic_sign=np.int8(1))

==== COVERAGE ====
Coverage probability: 0.184 (target = 0.95)
```



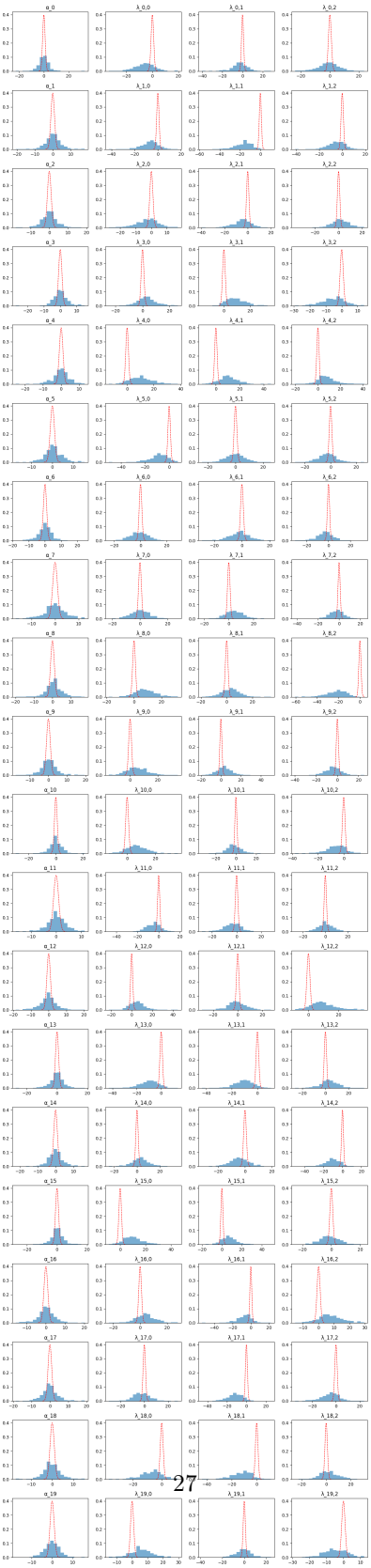


[ ]:
```
def plot_all_z_histograms(eta_hats, avars, eta_true_vec, N, K, T, bins=20):
    """
    Creates a grid of z-score histograms:
    Shape: N rows × (K+1) columns.
    """

    p = N * (K+1)
    n_rep = eta_hats.shape[0]

    fig, axes = plt.subplots(N, K+1, figsize=(3*(K+1), 2.5*N), squeeze=False)

    for i in range(N):
        for j in range(K+1):
            ax = axes[i, j]
```

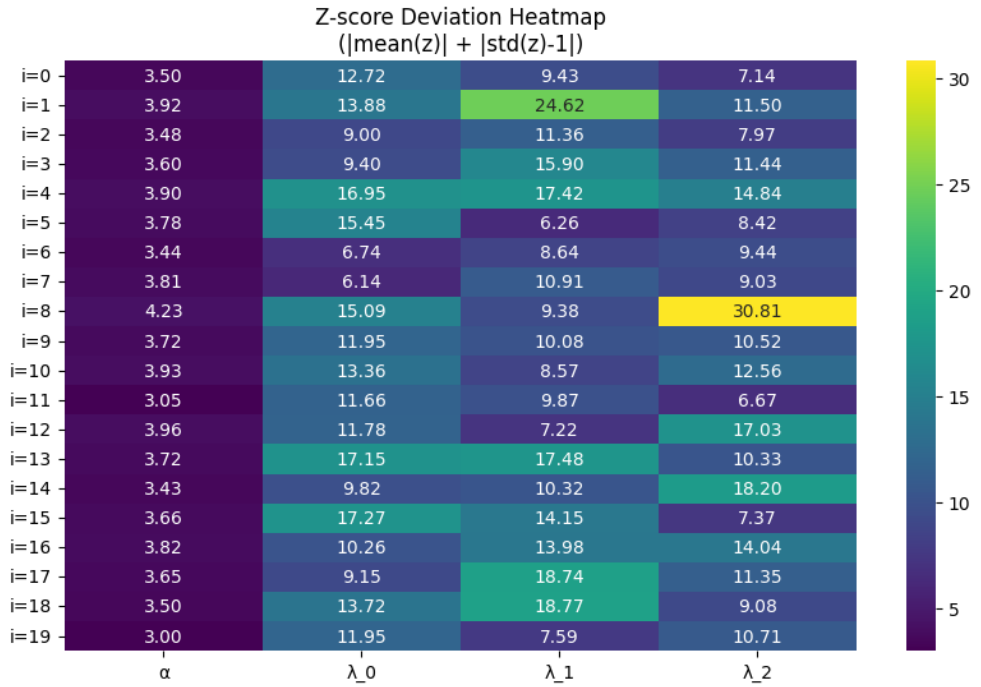
```
            param_index = get_param_index(i, j, K)

            # compute z-values for this parameter
            z_vals = []
            for rep in range(n_rep):
                eta_vec = eta_hats[rep].reshape(p)
                avar_hat = avars[rep]
                se = np.sqrt(avar_hat[param_index, param_index] / T)
                z = (eta_vec[param_index] - eta_true_vec[param_index]) / se
                z_vals.append(z)
            z_vals = np.array(z_vals)

            # histogram
            ax.hist(z_vals, bins=bins, density=True, alpha=0.6)

            # normal pdf
            x = np.linspace(-4,4,200)
            ax.plot(x, norm.pdf(x), "r--", linewidth=1)

            # label
            if j == 0:
                ax.set_title(f" _{i}")
            else:
                ax.set_title(f" _{i},{j-1}")

    plt.tight_layout()
    plt.show()

def compute_z_matrix(eta_hats, avars, eta_true_vec, N, K, T):
    """
    Returns an N x (K+1) matrix of z-score deviation:
       deviation = |mean(z)| + |std(z) - 1|
    Larger numbers = worse normal fit.
    """
    p = N * (K+1)
    n_rep = eta_hats.shape[0]

    deviation = np.zeros((N, K+1))

    for i in range(N):
        for j in range(K+1):
            param_index = get_param_index(i, j, K)

            # compute z-values
            z_vals = []
            for rep in range(n_rep):
                eta_vec = eta_hats[rep].reshape(p)
```

```
                avar_hat = avars[rep]
                se = np.sqrt(avar_hat[param_index, param_index] / T)
                z = (eta_vec[param_index] - eta_true_vec[param_index]) / se
                z_vals.append(z)
            z_vals = np.array(z_vals)

            # deviation from N(0,1)
            deviation[i,j] = abs(z_vals.mean()) + abs(z_vals.std() - 1)

    return deviation

def plot_z_heatmap(eta_hats, avars, eta_true_vec, N, K, T):
    deviation = compute_z_matrix(eta_hats, avars, eta_true_vec, N, K, T)

    plt.figure(figsize=(10,6))
    sns.heatmap(
        deviation,
        cmap="viridis",
        xticklabels=[f" "] + [f" _{j}" for j in range(K)],
        yticklabels=[f"i={i}" for i in range(N)],
        annot=True,
        fmt=".2f"
    )
    plt.title("Z-score Deviation Heatmap\n(|mean(z)| + |std(z)-1|)")
    plt.show()
```

[ ]:
```
plot_all_z_histograms(
    eta_hats, avars, eta_true_vec,
    N=20, K=3, T=200
)
```


α_0
λ_0,0
λ_0,1
λ_0,2
α_1
λ_1,0
λ_1,1
λ_1,2
α_2
λ_2,0
λ_2,1
λ_2,2
α_3
λ_3,0
λ_3,1
λ_3,2
α_4
λ_4,0
λ_4,1
λ_4,2
α_5
λ_5,0
λ_5,1
λ_5,2
α_6
λ_6,0
λ_6,1
λ_6,2
α_7
λ_7,0
λ_7,1
λ_7,2
α_8
λ_8,0
λ_8,1
λ_8,2
α_9
λ_9,0
λ_9,1
λ_9,2
α_10
λ_10,0
λ_10,1
λ_10,2
α_11
λ_11,0
λ_11,1
λ_11,2
α_12
λ_12,0
λ_12,1
λ_12,2
α_13
λ_13,0
λ_13,1
λ_13,2
α_14
λ_14,0
λ_14,1
λ_14,2
α_15
λ_15,0
λ_15,1
λ_15,2
α_16
λ_16,0
λ_16,1
λ_16,2
α_17
λ_17,0
λ_17,1
λ_17,2
α_18
λ_18,0
λ_18,1
λ_18,2
α_19
λ_19,0
λ_19,1
λ_19,2

[ ]:
```
plot_z_heatmap(
    eta_hats, avars, eta_true_vec,
    N=20, K=3, T=200,
)
```



06_empirical_homogeneity_test

December 18, 2025

1 Empirical Homogeneity Test

1.1 Notebook Setup

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
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_extended.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")
```

1.2 Set up empirical testing parameters

[3]:
```
factor_options = [
    ['Mkt-RF'],
    ['Mkt-RF', 'SMB', 'HML'],
    ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'],
]
R_options = [1, 2, 5]
sample_period_options = [
    ('1963-01-01', '2025-12-31'),
    ('1963-01-01', '1983-01-01'),
    ('1973-01-01', '1993-01-01'),
```

```
    ('1983-01-01', '2003-01-01'),
    ('1993-01-01', '2013-01-01'),
    ('2003-01-01', '2023-01-01'),
]

results = pd.DataFrame(
    index=pd.MultiIndex.from_product([
        list(map(tuple, factor_options)),   # convert lists → tuples
        R_options,
        sample_period_options
    ]),
    columns=['gamma_a_lam', 'gamma_a', 'gamma_lam']
)


print(f"Total combinations: {results.shape[0]}")
counter = 0

for factors in factor_options:
    K = len(factors)
    for R in R_options:
        for sample_period in sample_period_options:
            print(f"Processing {counter}/{results.shape[0]}: {factors} - {R} -␣
↪{sample_period}")
            data_slice = data.loc[
                (data.index > sample_period[0]) &
                (data.index < sample_period[1])
            ]
            beta_loading, returns_df, realized_covariance, residuals = utils.
↪calculate_factor_loading(
                data_slice,
                factors=factors,
                assets=assets
            )

            excess_returns = returns_df.groupby("Quarter").sum()[assets].T.
↪values
            industries = beta_loading.index.get_level_values(0).unique().
↪tolist()
            factors_names = 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


            eta, G, beta_star, objective = utils.iterative_convergence(
                beta_hat_np,
                excess_returns,
                N = N,
                K = K,
                R = R,
                T = T,
                n_iter=2000
            )

            avar = utils.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 = N,
                K = K,
                R = R,
                T = T,
            )


            gamma_a_lambda = utils.full_homogeneity_test(
                eta = eta,
                avar = avar,
                N = N,
                K = K,
                T = T
            )

            gamma_a = utils.intercept_homogeneity_test(
                eta = eta,
                avar = avar,
                N = N,
                K = K,
                T = T
            )
```

```
            gamma_lambda = utils.slope_homogeneity_test(
                eta = eta,
                avar = avar,
                N = N,
                K = K,
                T = T
            )
            print(f"Test statistics")
            print(f"gamma_a_lam: {gamma_a_lambda}")
            print(f"gamma_a: {gamma_a}")
            print(f"gamma_lam: {gamma_lambda}")

            results.loc[(
                tuple(factors), R, sample_period
            )] = np.asarray([
                gamma_a_lambda,
                gamma_a,
                gamma_lambda
            ])
            counter += 1
            print(f"============================")
```

```
Total combinations: 54
Processing 0/54: ['Mkt-RF'] - 1 - ('1963-01-01', '2025-12-31')
Test statistics
gamma_a_lam: 44.021375176126504
gamma_a: -4.700491211791159
gamma_lam: -4.793114003302496
============================
Processing 1/54: ['Mkt-RF'] - 1 - ('1963-01-01', '1983-01-01')
Test statistics
gamma_a_lam: -19.45507836193608
gamma_a: -5.290005147715778
gamma_lam: -4.82127703591286
============================
Processing 2/54: ['Mkt-RF'] - 1 - ('1973-01-01', '1993-01-01')
Test statistics
gamma_a_lam: 38.51023758818772
gamma_a: -4.780716850584511
gamma_lam: -4.7891599721430245
============================
Processing 3/54: ['Mkt-RF'] - 1 - ('1983-01-01', '2003-01-01')
Test statistics
gamma_a_lam: -29.992553862296607
gamma_a: -4.8616347938768305
gamma_lam: -4.796865405543951
============================
```

```
Processing 4/54: ['Mkt-RF'] - 1 - ('1993-01-01', '2013-01-01')
Test statistics
gamma_a_lam: -11.007995129966412
gamma_a: -5.291507502318423
gamma_lam: -4.738953249229063
============================
Processing 5/54: ['Mkt-RF'] - 1 - ('2003-01-01', '2023-01-01')
Test statistics
gamma_a_lam: 24.688723708717475
gamma_a: -3.749937797792885
gamma_lam: -4.3028255952482075
============================
Processing 6/54: ['Mkt-RF'] - 2 - ('1963-01-01', '2025-12-31')
Test statistics
gamma_a_lam: -2.6732672426974027
gamma_a: -5.339212629468527
gamma_lam: -4.798137830129114
============================
Processing 7/54: ['Mkt-RF'] - 2 - ('1963-01-01', '1983-01-01')
Test statistics
gamma_a_lam: 88.12448718838729
gamma_a: 0.48587747852576674
gamma_lam: -2.3129955443075003
============================
Processing 8/54: ['Mkt-RF'] - 2 - ('1973-01-01', '1993-01-01')
Test statistics
gamma_a_lam: -42.373442941233144
gamma_a: -4.8611064892949125
gamma_lam: -4.798696771105146
============================
Processing 9/54: ['Mkt-RF'] - 2 - ('1983-01-01', '2003-01-01')
Test statistics
gamma_a_lam: -66.67978493809103
gamma_a: -4.304672737445764
gamma_lam: -5.388708326488772
============================
Processing 10/54: ['Mkt-RF'] - 2 - ('1993-01-01', '2013-01-01')
Test statistics
gamma_a_lam: -52.79965093886
gamma_a: -6.252624253692823
gamma_lam: -5.062956245535808
============================
Processing 11/54: ['Mkt-RF'] - 2 - ('2003-01-01', '2023-01-01')
Test statistics
gamma_a_lam: -4.784644320688956
gamma_a: -5.196610265700589
gamma_lam: -4.767763418798012
============================
```

```
Processing 12/54: ['Mkt-RF'] - 5 - ('1963-01-01', '2025-12-31')
Test statistics
gamma_a_lam: 2438.235754570473
gamma_a: -4.408409913623353
gamma_lam: 10.27941437482029
============================
Processing 13/54: ['Mkt-RF'] - 5 - ('1963-01-01', '1983-01-01')
Test statistics
gamma_a_lam: -54.07376914753523
gamma_a: -2.3270995295242027
gamma_lam: -2.9256391278549447
============================
Processing 14/54: ['Mkt-RF'] - 5 - ('1973-01-01', '1993-01-01')
Test statistics
gamma_a_lam: 54.92341803552409
gamma_a: -4.614662623076495
gamma_lam: -4.375630327908701
============================
Processing 15/54: ['Mkt-RF'] - 5 - ('1983-01-01', '2003-01-01')
Test statistics
gamma_a_lam: -110.7462205971981
gamma_a: -0.6415817446507754
gamma_lam: -3.2707208769144454
============================
Processing 16/54: ['Mkt-RF'] - 5 - ('1993-01-01', '2013-01-01')
Test statistics
gamma_a_lam: 23007.892323800672
gamma_a: -4.420165690507534
gamma_lam: 405.0376607175024
============================
Processing 17/54: ['Mkt-RF'] - 5 - ('2003-01-01', '2023-01-01')
Test statistics
gamma_a_lam: 2327.3686235461323
gamma_a: -4.628706308110402
gamma_lam: 52.707782016738044
============================
Processing 18/54: ['Mkt-RF', 'SMB', 'HML'] - 1 - ('1963-01-01', '2025-12-31')
Test statistics
gamma_a_lam: -8.923047093232306
gamma_a: -8.342481795516516
gamma_lam: -8.316274429301354
============================
Processing 19/54: ['Mkt-RF', 'SMB', 'HML'] - 1 - ('1963-01-01', '1983-01-01')
Test statistics
gamma_a_lam: -5.314649047383824
gamma_a: -8.280847252572762
gamma_lam: -8.281782396994634
============================
```

```
Processing 20/54: ['Mkt-RF', 'SMB', 'HML'] - 1 - ('1973-01-01', '1993-01-01')
Test statistics
gamma_a_lam: -10.300182757377934
gamma_a: -8.308301059276616
gamma_lam: -8.302502751359713
============================
Processing 21/54: ['Mkt-RF', 'SMB', 'HML'] - 1 - ('1983-01-01', '2003-01-01')
Test statistics
gamma_a_lam: -2.406316386752981
gamma_a: -8.300168588980867
gamma_lam: -8.319993557702267
============================
Processing 22/54: ['Mkt-RF', 'SMB', 'HML'] - 1 - ('1993-01-01', '2013-01-01')
Test statistics
gamma_a_lam: -0.41990634528970106
gamma_a: -8.319842344640795
gamma_lam: -8.194226581906426
============================
Processing 23/54: ['Mkt-RF', 'SMB', 'HML'] - 1 - ('2003-01-01', '2023-01-01')
Test statistics
gamma_a_lam: -3.4937121635169968
gamma_a: -8.412641044776443
gamma_lam: -8.185507994135017
============================
Processing 24/54: ['Mkt-RF', 'SMB', 'HML'] - 2 - ('1963-01-01', '2025-12-31')
Test statistics
gamma_a_lam: -6.179766852180811
gamma_a: -8.332050591414788
gamma_lam: -8.304815827058258
============================
Processing 25/54: ['Mkt-RF', 'SMB', 'HML'] - 2 - ('1963-01-01', '1983-01-01')
Test statistics
gamma_a_lam: -5.485200270298585
gamma_a: -8.320074970812707
gamma_lam: -8.234234510709241
============================
Processing 26/54: ['Mkt-RF', 'SMB', 'HML'] - 2 - ('1973-01-01', '1993-01-01')
Test statistics
gamma_a_lam: -8.930289623322828
gamma_a: -8.307522529574804
gamma_lam: -8.296730946931552
============================
Processing 27/54: ['Mkt-RF', 'SMB', 'HML'] - 2 - ('1983-01-01', '2003-01-01')
Test statistics
gamma_a_lam: 28.525921920863002
gamma_a: -7.92538694894371
gamma_lam: -7.476400000084076
============================
```

```
Processing 28/54: ['Mkt-RF', 'SMB', 'HML'] - 2 - ('1993-01-01', '2013-01-01')
Test statistics
gamma_a_lam: -5.057213948893936
gamma_a: -8.306454288597754
gamma_lam: -8.205388413949166
============================
Processing 29/54: ['Mkt-RF', 'SMB', 'HML'] - 2 - ('2003-01-01', '2023-01-01')
Test statistics
gamma_a_lam: 6.245207755020384
gamma_a: -8.295843603336781
gamma_lam: -8.119037880321539
============================
Processing 30/54: ['Mkt-RF', 'SMB', 'HML'] - 5 - ('1963-01-01', '2025-12-31')
Test statistics
gamma_a_lam: 25.315847021418865
gamma_a: -8.303484886785856
gamma_lam: -8.153324480349433
============================
Processing 31/54: ['Mkt-RF', 'SMB', 'HML'] - 5 - ('1963-01-01', '1983-01-01')
Test statistics
gamma_a_lam: 80.82782726012826
gamma_a: -8.145131563688048
gamma_lam: -5.364330857647703
============================
Processing 32/54: ['Mkt-RF', 'SMB', 'HML'] - 5 - ('1973-01-01', '1993-01-01')
Test statistics
gamma_a_lam: -8.791553803265451
gamma_a: -8.306774051476598
gamma_lam: -8.293809428971358
============================
Processing 33/54: ['Mkt-RF', 'SMB', 'HML'] - 5 - ('1983-01-01', '2003-01-01')
Test statistics
gamma_a_lam: 160.97248409693034
gamma_a: -5.5733069604654455
gamma_lam: -8.02360459901131
============================
Processing 34/54: ['Mkt-RF', 'SMB', 'HML'] - 5 - ('1993-01-01', '2013-01-01')
Test statistics
gamma_a_lam: 20.58239283143639
gamma_a: -8.309451357234803
gamma_lam: -7.91183313085424
============================
Processing 35/54: ['Mkt-RF', 'SMB', 'HML'] - 5 - ('2003-01-01', '2023-01-01')
Test statistics
gamma_a_lam: 190.7316567042458
gamma_a: -8.28268522981492
gamma_lam: -5.35425478148479
============================
```

```
Processing 36/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 1 - ('1963-01-01',
'2025-12-31')
Test statistics
gamma_a_lam: -6.989102222335923
gamma_a: -10.729210072723074
gamma_lam: -10.71198401688972
============================
Processing 37/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 1 - ('1963-01-01',
'1983-01-01')
Test statistics
gamma_a_lam: -318.5416871338544
gamma_a: -10.740611204993483
gamma_lam: -14.80512922722086
============================
Processing 38/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 1 - ('1973-01-01',
'1993-01-01')
Test statistics
gamma_a_lam: -10.775980300456077
gamma_a: -10.723980282508062
gamma_lam: -10.711166523369874
============================
Processing 39/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 1 - ('1983-01-01',
'2003-01-01')
Test statistics
gamma_a_lam: -11.416336712717651
gamma_a: -10.721325132066054
gamma_lam: -10.704642047423233
============================
Processing 40/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 1 - ('1993-01-01',
'2013-01-01')
Test statistics
gamma_a_lam: -5.13661953183519
gamma_a: -10.729204576609552
gamma_lam: -10.63414572420469
============================
Processing 41/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 1 - ('2003-01-01',
'2023-01-01')
Test statistics
gamma_a_lam: 8.601292836020288
gamma_a: -10.724989530371124
gamma_lam: -10.439807841455991
============================
Processing 42/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 2 - ('1963-01-01',
'2025-12-31')
Test statistics
gamma_a_lam: -8.436648789139339
gamma_a: -10.723460705114645
gamma_lam: -10.694462133694998
```

```
==========================
Processing 43/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 2 - ('1963-01-01',
'1983-01-01')
Test statistics
gamma_a_lam: -7.207820064974214
gamma_a: -10.73009196773403
gamma_lam: -10.679977870666
==========================
Processing 44/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 2 - ('1973-01-01',
'1993-01-01')
Test statistics
gamma_a_lam: -11.08305022537954
gamma_a: -10.723819109374606
gamma_lam: -10.714400535178392
==========================
Processing 45/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 2 - ('1983-01-01',
'2003-01-01')
Test statistics
gamma_a_lam: -9.90601732155809
gamma_a: -10.723600199018447
gamma_lam: -10.700533788162488
==========================
Processing 46/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 2 - ('1993-01-01',
'2013-01-01')
Test statistics
gamma_a_lam: -1.924426667705066
gamma_a: -10.71666063535228
gamma_lam: -10.458018252513854
==========================
Processing 47/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 2 - ('2003-01-01',
'2023-01-01')
Test statistics
gamma_a_lam: 0.33669647749347487
gamma_a: -10.717627169786844
gamma_lam: -10.581157489857768
==========================
Processing 48/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 5 - ('1963-01-01',
'2025-12-31')
Test statistics
gamma_a_lam: -9.916105003169564
gamma_a: -10.713971677782427
gamma_lam: -10.705596837215989
==========================
Processing 49/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 5 - ('1963-01-01',
'1983-01-01')
Test statistics
gamma_a_lam: 1.3329101008414095
gamma_a: -10.592708095027165
```

```
gamma_lam: -10.59276717465792
============================
Processing 50/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 5 - ('1973-01-01',
'1993-01-01')
Test statistics
gamma_a_lam: -7.920070704109096
gamma_a: -10.723821766053717
gamma_lam: -10.670876394277588
============================
Processing 51/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 5 - ('1983-01-01',
'2003-01-01')
Test statistics
gamma_a_lam: 3.275619012570612
gamma_a: -10.556758491551385
gamma_lam: -10.589651287052614
============================
Processing 52/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 5 - ('1993-01-01',
'2013-01-01')
Test statistics
gamma_a_lam: 39.095190938146686
gamma_a: -10.723974414915215
gamma_lam: -10.027807855086415
============================
Processing 53/54: ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'] - 5 - ('2003-01-01',
'2023-01-01')
Test statistics
gamma_a_lam: 1.2406245003043173
gamma_a: -10.724461939758651
gamma_lam: -10.542239363092206
============================
```

[4]:
```
results
```

[4]:

				gamma_a_lam
(Mkt-RF,)	1	(1963-01-01, 2025-12-31)		44.021375
		(1963-01-01, 1983-01-01)		-19.455078
		(1973-01-01, 1993-01-01)		38.510238
		(1983-01-01, 2003-01-01)		-29.992554
		(1993-01-01, 2013-01-01)		-11.007995
		(2003-01-01, 2023-01-01)		24.688724
	2	(1963-01-01, 2025-12-31)		-2.673267
		(1963-01-01, 1983-01-01)		88.124487
		(1973-01-01, 1993-01-01)		-42.373443
		(1983-01-01, 2003-01-01)		-66.679785
		(1993-01-01, 2013-01-01)		-52.799651
		(2003-01-01, 2023-01-01)		-4.784644
	5	(1963-01-01, 2025-12-31)		2438.235755

		(1963-01-01, 1983-01-01)	-54.073769
		(1973-01-01, 1993-01-01)	54.923418
		(1983-01-01, 2003-01-01)	-110.746221
		(1993-01-01, 2013-01-01)	23007.892324
		(2003-01-01, 2023-01-01)	2327.368624
(Mkt-RF, SMB, HML)	1	(1963-01-01, 2025-12-31)	-8.923047
		(1963-01-01, 1983-01-01)	-5.314649
		(1973-01-01, 1993-01-01)	-10.300183
		(1983-01-01, 2003-01-01)	-2.406316
		(1993-01-01, 2013-01-01)	-0.419906
		(2003-01-01, 2023-01-01)	-3.493712
	2	(1963-01-01, 2025-12-31)	-6.179767
		(1963-01-01, 1983-01-01)	-5.4852
		(1973-01-01, 1993-01-01)	-8.93029
		(1983-01-01, 2003-01-01)	28.525922
		(1993-01-01, 2013-01-01)	-5.057214
		(2003-01-01, 2023-01-01)	6.245208
	5	(1963-01-01, 2025-12-31)	25.315847
		(1963-01-01, 1983-01-01)	80.827827
		(1973-01-01, 1993-01-01)	-8.791554
		(1983-01-01, 2003-01-01)	160.972484
		(1993-01-01, 2013-01-01)	20.582393
		(2003-01-01, 2023-01-01)	190.731657
(Mkt-RF, SMB, HML, RMW, CMA)	1	(1963-01-01, 2025-12-31)	-6.989102
		(1963-01-01, 1983-01-01)	-318.541687
		(1973-01-01, 1993-01-01)	-10.77598
		(1983-01-01, 2003-01-01)	-11.416337
		(1993-01-01, 2013-01-01)	-5.13662
		(2003-01-01, 2023-01-01)	8.601293
	2	(1963-01-01, 2025-12-31)	-8.436649
		(1963-01-01, 1983-01-01)	-7.20782
		(1973-01-01, 1993-01-01)	-11.08305
		(1983-01-01, 2003-01-01)	-9.906017
		(1993-01-01, 2013-01-01)	-1.924427
		(2003-01-01, 2023-01-01)	0.336696
	5	(1963-01-01, 2025-12-31)	-9.916105
		(1963-01-01, 1983-01-01)	1.33291
		(1973-01-01, 1993-01-01)	-7.920071
		(1983-01-01, 2003-01-01)	3.275619
		(1993-01-01, 2013-01-01)	39.095191
		(2003-01-01, 2023-01-01)	1.240625

			gamma_a	gamma_lam
(Mkt-RF,)	1	(1963-01-01, 2025-12-31)	-4.700491	-4.793114
		(1963-01-01, 1983-01-01)	-5.290005	-4.821277
		(1973-01-01, 1993-01-01)	-4.780717	-4.78916
		(1983-01-01, 2003-01-01)	-4.861635	-4.796865

		(1993-01-01, 2013-01-01)	-5.291508	-4.738953
		(2003-01-01, 2023-01-01)	-3.749938	-4.302826
	2	(1963-01-01, 2025-12-31)	-5.339213	-4.798138
		(1963-01-01, 1983-01-01)	0.485877	-2.312996
		(1973-01-01, 1993-01-01)	-4.861106	-4.798697
		(1983-01-01, 2003-01-01)	-4.304673	-5.388708
		(1993-01-01, 2013-01-01)	-6.252624	-5.062956
		(2003-01-01, 2023-01-01)	-5.19661	-4.767763
	5	(1963-01-01, 2025-12-31)	-4.40841	10.279414
		(1963-01-01, 1983-01-01)	-2.3271	-2.925639
		(1973-01-01, 1993-01-01)	-4.614663	-4.37563
		(1983-01-01, 2003-01-01)	-0.641582	-3.270721
		(1993-01-01, 2013-01-01)	-4.420166	405.037661
		(2003-01-01, 2023-01-01)	-4.628706	52.707782
(Mkt-RF, SMB, HML)	1	(1963-01-01, 2025-12-31)	-8.342482	-8.316274
		(1963-01-01, 1983-01-01)	-8.280847	-8.281782
		(1973-01-01, 1993-01-01)	-8.308301	-8.302503
		(1983-01-01, 2003-01-01)	-8.300169	-8.319994
		(1993-01-01, 2013-01-01)	-8.319842	-8.194227
		(2003-01-01, 2023-01-01)	-8.412641	-8.185508
	2	(1963-01-01, 2025-12-31)	-8.332051	-8.304816
		(1963-01-01, 1983-01-01)	-8.320075	-8.234235
		(1973-01-01, 1993-01-01)	-8.307523	-8.296731
		(1983-01-01, 2003-01-01)	-7.925387	-7.4764
		(1993-01-01, 2013-01-01)	-8.306454	-8.205388
		(2003-01-01, 2023-01-01)	-8.295844	-8.119038
	5	(1963-01-01, 2025-12-31)	-8.303485	-8.153324
		(1963-01-01, 1983-01-01)	-8.145132	-5.364331
		(1973-01-01, 1993-01-01)	-8.306774	-8.293809
		(1983-01-01, 2003-01-01)	-5.573307	-8.023605
		(1993-01-01, 2013-01-01)	-8.309451	-7.911833
		(2003-01-01, 2023-01-01)	-8.282685	-5.354255
(Mkt-RF, SMB, HML, RMW, CMA)	1	(1963-01-01, 2025-12-31)	-10.72921	-10.711984
		(1963-01-01, 1983-01-01)	-10.740611	-14.805129
		(1973-01-01, 1993-01-01)	-10.72398	-10.711167
		(1983-01-01, 2003-01-01)	-10.721325	-10.704642
		(1993-01-01, 2013-01-01)	-10.729205	-10.634146
		(2003-01-01, 2023-01-01)	-10.72499	-10.439808
	2	(1963-01-01, 2025-12-31)	-10.723461	-10.694462
		(1963-01-01, 1983-01-01)	-10.730092	-10.679978
		(1973-01-01, 1993-01-01)	-10.723819	-10.714401
		(1983-01-01, 2003-01-01)	-10.7236	-10.700534
		(1993-01-01, 2013-01-01)	-10.716661	-10.458018
		(2003-01-01, 2023-01-01)	-10.717627	-10.581157
	5	(1963-01-01, 2025-12-31)	-10.713972	-10.705597
		(1963-01-01, 1983-01-01)	-10.592708	-10.592767
		(1973-01-01, 1993-01-01)	-10.723822	-10.670876

```
                                    (1983-01-01, 2003-01-01) -10.556758  -10.589651
                                    (1993-01-01, 2013-01-01) -10.723974  -10.027808
                                    (2003-01-01, 2023-01-01) -10.724462  -10.542239
```

[5]:
```python
def clean_results_index(results):

    fac_idx = results.index.get_level_values(0)
    R_idx   = results.index.get_level_values(1)
    t_idx   = results.index.get_level_values(2)

    fac_new = [len(x) for x in fac_idx]

    t_new = [
        f"{str(start)[:4]}-{str(end)[:4]}"
        for start, end in t_idx
    ]

    new_index = pd.MultiIndex.from_arrays(
        [fac_new, R_idx, t_new],
        names=["K", "R", "Period"]
    )

    cleaned = results.copy()
    cleaned.index = new_index
    return cleaned

cleaned_results = clean_results_index(results)
cleaned_results
```

[5]:

K	R	Period	gamma_a_lam	gamma_a	gamma_lam
1	1	1963-2025	44.021375	-4.700491	-4.793114
		1963-1983	-19.455078	-5.290005	-4.821277
		1973-1993	38.510238	-4.780717	-4.78916
		1983-2003	-29.992554	-4.861635	-4.796865
		1993-2013	-11.007995	-5.291508	-4.738953
		2003-2023	24.688724	-3.749938	-4.302826
	2	1963-2025	-2.673267	-5.339213	-4.798138
		1963-1983	88.124487	0.485877	-2.312996
		1973-1993	-42.373443	-4.861106	-4.798697
		1983-2003	-66.679785	-4.304673	-5.388708
		1993-2013	-52.799651	-6.252624	-5.062956
		2003-2023	-4.784644	-5.19661	-4.767763
	5	1963-2025	2438.235755	-4.40841	10.279414
		1963-1983	-54.073769	-2.3271	-2.925639
		1973-1993	54.923418	-4.614663	-4.37563
		1983-2003	-110.746221	-0.641582	-3.270721

```
      1993-2013  23007.892324  -4.420166  405.037661
      2003-2023   2327.368624  -4.628706   52.707782
  3 1 1963-2025     -8.923047  -8.342482   -8.316274
      1963-1983     -5.314649  -8.280847   -8.281782
      1973-1993    -10.300183  -8.308301   -8.302503
      1983-2003     -2.406316  -8.300169   -8.319994
      1993-2013     -0.419906  -8.319842   -8.194227
      2003-2023     -3.493712  -8.412641   -8.185508
    2 1963-2025     -6.179767  -8.332051   -8.304816
      1963-1983       -5.4852  -8.320075   -8.234235
      1973-1993      -8.93029  -8.307523   -8.296731
      1983-2003     28.525922  -7.925387     -7.4764
      1993-2013     -5.057214  -8.306454   -8.205388
      2003-2023      6.245208  -8.295844   -8.119038
    5 1963-2025     25.315847  -8.303485   -8.153324
      1963-1983     80.827827  -8.145132   -5.364331
      1973-1993     -8.791554  -8.306774   -8.293809
      1983-2003    160.972484  -5.573307   -8.023605
      1993-2013     20.582393  -8.309451   -7.911833
      2003-2023    190.731657  -8.282685   -5.354255
  5 1 1963-2025     -6.989102  -10.72921  -10.711984
      1963-1983   -318.541687 -10.740611  -14.805129
      1973-1993     -10.77598  -10.72398  -10.711167
      1983-2003    -11.416337 -10.721325  -10.704642
      1993-2013      -5.13662 -10.729205  -10.634146
      2003-2023      8.601293  -10.72499  -10.439808
    2 1963-2025     -8.436649 -10.723461  -10.694462
      1963-1983      -7.20782 -10.730092  -10.679978
      1973-1993     -11.08305 -10.723819  -10.714401
      1983-2003     -9.906017   -10.7236  -10.700534
      1993-2013     -1.924427 -10.716661  -10.458018
      2003-2023      0.336696 -10.717627  -10.581157
    5 1963-2025     -9.916105 -10.713972  -10.705597
      1963-1983       1.33291 -10.592708  -10.592767
      1973-1993     -7.920071 -10.723822  -10.670876
      1983-2003      3.275619 -10.556758  -10.589651
      1993-2013     39.095191 -10.723974  -10.027808
      2003-2023      1.240625 -10.724462  -10.542239
```

[6]:
```
table_gamma_a_lam = cleaned_results["gamma_a_lam"].unstack(level=[0,1])
table_gamma_a     = cleaned_results["gamma_a"].unstack(level=[0,1])
table_gamma_lam   = cleaned_results["gamma_lam"].unstack(level=[0,1])
```

[7]:
```
def add_p_values(table):
    table_numeric = table.apply(pd.to_numeric, errors="coerce")

    periods = table_numeric.index
```

```
    columns = table_numeric.columns

    new_rows = []
    new_index = []

    for period in periods:
        stats = table_numeric.loc[period].values.astype(float)

        new_rows.append(stats)
        new_index.append((period, "$\gamma$"))

        pvals = 2 * (1 - norm.cdf(np.abs(stats)))
        new_rows.append(pvals)
        new_index.append((period, "$p$"))

    multi_index = pd.MultiIndex.from_tuples(new_index, names=["Period", "Type"])
    new_table = pd.DataFrame(new_rows, index=multi_index, columns=columns)

    return new_table
```

[8]:
```
table_gamma_a_lam
```

[8]:
```
K                 1                                3             \
R                 1          2             5          1          2
Period
1963-1983 -19.455078  88.124487    -54.073769  -5.314649    -5.4852
1963-2025  44.021375  -2.673267   2438.235755  -8.923047  -6.179767
1973-1993  38.510238 -42.373443     54.923418 -10.300183   -8.93029
1983-2003 -29.992554 -66.679785   -110.746221  -2.406316  28.525922
1993-2013 -11.007995 -52.799651  23007.892324  -0.419906  -5.057214
2003-2023  24.688724  -4.784644   2327.368624  -3.493712   6.245208

K                              5
R                  5           1         2          5
Period
1963-1983   80.827827 -318.541687  -7.20782    1.33291
1963-2025   25.315847   -6.989102 -8.436649  -9.916105
1973-1993   -8.791554   -10.77598 -11.08305  -7.920071
1983-2003  160.972484  -11.416337 -9.906017   3.275619
1993-2013   20.582393    -5.13662 -1.924427  39.095191
2003-2023  190.731657    8.601293  0.336696   1.240625
```

[9]:
```
table_gamma_a_lam = add_p_values(table_gamma_a_lam)
table_gamma_a = add_p_values(table_gamma_a)
table_gamma_lam = add_p_values(table_gamma_lam)
```

[ ]:
```
latex_a_lam = table_gamma_a_lam.round(3).to_latex(
    multirow=True,
    multicolumn=True,
    index=True,
    escape=False,
    caption="Joint Homogeneity Test ($\\Gamma_{\\alpha,\\lambda}$) with␣
 ↪p-values",
    label="tab:gamma_a_lam_with_p",
    float_format="%.2f",
)

latex_a = table_gamma_a.round(3).to_latex(
    multirow=True,
    multicolumn=True,
    index=True,
    escape=False,
    caption="Intercept Homogeneity Test ($\\Gamma_{\\alpha}$) with p-values",
    label="tab:gamma_a",
    float_format="%.2f",
)

latex_lam = table_gamma_lam.round(3).to_latex(
    multirow=True,
    multicolumn=True,
    index=True,
    escape=False,
    caption="Slope Homogeneity Test ($\\Gamma_{\\lambda}$) with p-values",
    label="tab:gamma_lam",
    float_format="%.2f",
)

print(latex_a_lam)
print(latex_a)
print(latex_lam)
```

07_bootstrap

December 18, 2025

1 Bootstrapping

Due to some issues we noticed when replicating Galvao et al's asymptotic variance estimator and test statistic, we turn to bootstrapping to try to come up with bootstrap bias estimate, bootstrap standard errors, and an alternative Wald-type test.

1.1 Notebook setup

[63]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from scipy.stats import chi2
from scipy import stats
import torch
from tqdm import tqdm
import sys
sys.path.append('../')
from utils import utils
sys.executable
```

[63]:
```
'/Users/fanghema/Desktop/aaSTAT_5200/STAT_5200_final_project/env/bin/python'
```

[2]:
```
data = pd.read_csv(
    '../data/processed/data_extended.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
```

```
)

excess_returns = (
    returns_df
    .groupby("Quarter")
    .sum()
    [assets]
    .T
    .values
)
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]
R = 3

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
```

[2]: (47, 5, 250)

1.2 Using Jackknife

[ ]:
```
p = N * (K + 1)
jackknife_eta_estimates = np.zeros((T, p))

for t in range(T):
    if (t % 10) == 0:
        print(f"Processing {t} out of {T}")

    time_mask = np.ones(T, dtype=bool)
    time_mask[t] = False

    beta_hat_subset = beta_hat_np[:, :, time_mask]          # (N, K, T-1)
    excess_returns_subset = excess_returns[:, time_mask]    # (N, T-1)

    eta_jk, G_jk, beta_star_jk, objective_jk = utils.penalty_based_minimization(
        beta_hat=beta_hat_subset,
        excess_returns=excess_returns_subset,
        N=N,
```

```
        K=K,
        R=R,
        T=T-1,           # updated sample size
        n_iter=500,
    )

    jackknife_eta_estimates[t, :] = eta_jk.reshape(p)
```

Now, we do a full sample estimation.

[7]:
```
eta, G, beta_star, objective = utils.iterative_convergence(
    beta_hat=beta_hat_np,
    excess_returns=excess_returns,
    N=N,
    K=K,
    R=R,
    T=T,
    n_iter=2000,
    verbose=False
)

p = N * (K + 1)
eta_hat_full = eta.reshape(p)
```

[42]:
```
# Jackknife mean
eta_jk_mean = jackknife_eta_estimates.mean(axis=0)

# Jackknife bias
jk_bias = (T - 1) * (eta_jk_mean - eta_hat_full)

# Jackknife covariance
eta_centered = jackknife_eta_estimates - eta_jk_mean
jk_cov = ((T - 1) / T) * (eta_centered.T @ eta_centered)
```

[43]:
```
eta_centered.shape
```

[43]:
```
(250, 282)
```

[44]:
```
print(np.linalg.inv(jk_cov).min())
print(np.linalg.inv(jk_cov).max())
```

```
-1.863547211797388e+17
1.638447607959742e+17
```

[45]:
```
p = N * (K + 1)

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

```
d_vec = eta_centered.reshape(-1)
print("Typical |d|:", np.median(np.abs(d_vec)))

print("Diag(jk_cov) head:", np.diag(jk_cov)[:10])

rough_var = jackknife_eta_estimates.var(axis=0)[:10]
print("Naive var across jk replicates (first 10):", rough_var)
```

```
Typical |d|: 0.050935989443553834
Diag(jk_cov) head: [1.26641957e+01 9.11005233e-03 2.70422503e-02 5.87296361e-02
 7.69373618e-02 1.26184372e-01 5.24237287e+01 7.07177949e-02
 2.01213236e-01 4.07243362e-02]
Naive var across jk replicates (first 10): [5.08602237e-02 3.65865555e-05
1.08603415e-04 2.35861993e-04
 3.08985389e-04 5.06764545e-04 2.10537063e-01 2.84007209e-04
 8.08085283e-04 1.63551551e-04]
```

[40]:
```
def _flatten_eta(eta: np.ndarray) -> np.ndarray:
    """
    vec(eta) stacking assets one after another.
    eta: (N, K+1)
    returns: (N*(K+1),)
    """
    return eta.reshape(-1)


def wald_full_homogeneity_jackknife(
    eta: np.ndarray,
    jk_cov: np.ndarray,
    N: int,
    K: int
):
    """
    Wald-type test for joint homogeneity:
        H0: all intercepts equal AND all slopes equal across assets.
    Here we use centered eta_i (deviation from cross-sectional mean)
    and jackknife covariance of vec(eta).

    Returns:
        gamma_ad: standardized test statistic ~ N(0,1) under H0
        W: chi-square statistic ~  ²_q under H0
        p_value: p-value based on N(0,1) approximation for gamma_ad
    """
    p = N * (K + 1)
    assert eta.shape == (N, K + 1)
    assert jk_cov.shape == (p, p)
```

```
    # Center eta across assets
    eta_mean = eta.mean(axis=0)          # (K+1,)
    eta_centered = eta - eta_mean        # (N, K+1)
    d_vec = _flatten_eta(eta_centered)   # (p,)

    V = jk_cov
    V_inv = np.linalg.pinv(V)

    W = float(d_vec.T @ V_inv @ d_vec)

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

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

    p_val = 2 * (1 - norm.cdf(abs(gamma_ad)))

    return gamma_ad, W, p_val


def wald_intercept_homogeneity_jackknife(
    eta: np.ndarray,
    jk_cov: np.ndarray,
    N: int,
    K: int
):
    """
    Wald-type test for:
        H0: _i = 0  for all i (intercepts jointly zero).

    Uses jackknife covariance submatrix corresponding to intercepts.
    """
    p = N * (K + 1)
    assert eta.shape == (N, K + 1)
    assert jk_cov.shape == (p, p)

    alpha = eta[:, 0]                    # (N,)

    # indices of intercepts in vec(eta): 0, K+1, 2(K+1), ...
    alpha_idx = np.arange(0, p, K + 1)

    V_eta = jk_cov
    V_alpha = V_eta[np.ix_(alpha_idx, alpha_idx)]   # (N, N)
    V_alpha_inv = np.linalg.pinv(V_alpha)

    W = float(alpha.T @ V_alpha_inv @ alpha)

    q = N
```

```
    gamma_a = (W - q) / np.sqrt(2 * q)
    p_val = 2 * (1 - norm.cdf(abs(gamma_a)))

    return gamma_a, W, p_val


def wald_slope_homogeneity_jackknife(
    eta: np.ndarray,
    jk_cov: np.ndarray,
    N: int,
    K: int
):
    """
    Wald-type test for:
        H0: _i =    for all i  (all slope vectors equal across assets).

    We work with cross-sectionally centered slopes, so under H0
    the mean of vec(centered slopes) is approximately 0, and we
    use the jackknife covariance of those components.
    """
    p = N * (K + 1)
    assert eta.shape == (N, K + 1)
    assert jk_cov.shape == (p, p)

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

    slope_idx = []
    for i in range(N):
        for j in range(K):
            slope_idx.append(i * (K + 1) + 1 + j)
    slope_idx = np.array(slope_idx)

    V_eta = jk_cov
    V_lambda = V_eta[np.ix_(slope_idx, slope_idx)]   # (NK, NK)
    V_lambda_inv = np.linalg.pinv(V_lambda)

    W = float(slopes_vec.T @ V_lambda_inv @ slopes_vec)

    q = (N - 1) * K

    gamma_lambda = (W - q) / np.sqrt(2 * q)
    p_val = 2 * (1 - norm.cdf(abs(gamma_lambda)))

    return gamma_lambda, W, p_val
```

[41]:
```
gamma_a_lam_jk, W_a_lam_jk, _ = wald_full_homogeneity_jackknife(
    eta=eta,
    jk_cov = jk_cov,
    N = N,
    K = K
)

gamma_a_jk, W_a_jk, _= wald_intercept_homogeneity_jackknife(
    eta=eta,
    jk_cov = jk_cov,
    N = N,
    K = K
)

gamma_lam_jk, W_lam_jk, _= wald_slope_homogeneity_jackknife(
    eta=eta,
    jk_cov = jk_cov,
    N = N,
    K = K
)

print(gamma_a_lam_jk)
print(gamma_a_jk)
print(gamma_lam_jk)
```

```
1829.0568303859861
7.00146195513254
3411.160745721216
```

1.2.1 Using jackknife for full empirical test

[ ]:
```
factor_options = [
    ['Mkt-RF'],
    ['Mkt-RF', 'SMB', 'HML'],
    ['Mkt-RF', 'SMB', 'HML', 'RMW', 'CMA'],
]
R_options = [1, 2, 5]
sample_period_options = [
    ('1963-01-01', '2025-12-31'),
    ('1963-01-01', '1983-01-01'),
    ('1973-01-01', '1993-01-01'),
    ('1983-01-01', '2003-01-01'),
    ('1993-01-01', '2013-01-01'),
    ('2003-01-01', '2023-01-01'),
]

results = pd.DataFrame(
    index=pd.MultiIndex.from_product([
```

```
        list(map(tuple, factor_options)),   # convert lists → tuples
        R_options,
        sample_period_options
    ]),
    columns=['gamma_a_lam', 'gamma_a', 'gamma_lam']
)

print(f"Total combinations: {results.shape[0]}")
counter = 0

for factors in factor_options:
    K = len(factors)
    for R in R_options:
        for sample_period in sample_period_options:
            print(f"Processing {counter}/{results.shape[0]}: {factors} - {R} -␣
↪{sample_period}")
            data_slice = data.loc[
                (data.index > sample_period[0]) &
                (data.index < sample_period[1])
            ]
            beta_loading, returns_df, realized_covariance, residuals = utils.
↪calculate_factor_loading(
                data_slice,
                factors=factors,
                assets=assets
            )

            excess_returns = returns_df.groupby("Quarter").sum()[assets].T.
↪values
            industries = beta_loading.index.get_level_values(0).unique().
↪tolist()
            factors_names = 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
```

```
eta, G, beta_star, objective = utils.iterative_convergence(
    beta_hat_np,
    excess_returns,
    N = N,
    K = K,
    R = R,
    T = T,
    n_iter=2000
)

avar = utils.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 = N,
    K = K,
    R = R,
    T = T,
)


gamma_a_lambda = utils.full_homogeneity_test(
    eta = eta,
    avar = avar,
    N = N,
    K = K,
    T = T
)

gamma_a = utils.intercept_homogeneity_test(
    eta = eta,
    avar = avar,
    N = N,
    K = K,
    T = T
)

gamma_lambda = utils.slope_homogeneity_test(
    eta = eta,
    avar = avar,
    N = N,
    K = K,
    T = T
```

```
            )
            print(f"Test statistics")
            print(f"gamma_a_lam: {gamma_a_lambda}")
            print(f"gamma_a: {gamma_a}")
            print(f"gamma_lam: {gamma_lambda}")

            results.loc[(
                tuple(factors), R, sample_period
            )] = np.asarray([
                gamma_a_lambda,
                gamma_a,
                gamma_lambda
            ])
            counter += 1
            print(f"============================")
```

1.3 Testing for Type I error rate

[85]:
```
def run_mc_jackknife(
    N=20, K=3, R=1, T=200,
    MC_REPS=100,
    n_iter_est=400,
    seed=12345,
    verbose=False,
    heterogeneiy_strength = 0.0
):
    rng = np.random.default_rng(seed)

    # store statistics
    gamma_full_list = []
    gamma_a_list = []
    gamma_lam_list = []

    for rep in range(MC_REPS):
        rep_seed = rng.integers(1_000_000)

        # heterogeneity_strength = 0
        beta_true, r, realized_cov, residuals, G_true, beta_star_true,␣
↪lambda_true = utils.simulate_dgp(
            N=N, K=K, R=R, T=T,
            heterogeneity_strength=heterogeneiy_strength,
            seed=rep_seed
        )

        eta_hat, G_hat, beta_star_hat, obj = utils.iterative_convergence(
            beta_true,
            r,
```

```
        N=N, K=K, R=R, T=T,
        n_iter=n_iter_est,
        verbose=False
    )

    p = N * (K + 1)
    jk_eta = np.zeros((T, p))

    for t in range(T):
        mask = np.ones(T, dtype=bool)
        mask[t] = False

        beta_sub = beta_true[:, :, mask]
        r_sub = r[:, mask]

        eta_jk, _, _, _ = utils.iterative_convergence(
            beta_sub, r_sub,
            N=N, K=K, R=R, T=T-1,
            n_iter=n_iter_est,
            verbose=False
        )

        jk_eta[t, :] = eta_jk.reshape(p)

    jk_mean = jk_eta.mean(axis=0)

    diffs = jk_eta - jk_mean
    jk_cov = (T - 1) / T * (diffs.T @ diffs)

    gamma_full, _, _ = wald_full_homogeneity_jackknife(
        eta=eta_hat,
        jk_cov=jk_cov,
        N=N,
        K=K
    )
    gamma_a, _, _ = wald_intercept_homogeneity_jackknife(
        eta=eta_hat,
        jk_cov=jk_cov,
        N=N,
        K=K
    )
    gamma_lam, _, _ = wald_slope_homogeneity_jackknife(
        eta=eta_hat,
        jk_cov=jk_cov,
        N=N,
        K=K
    )
```

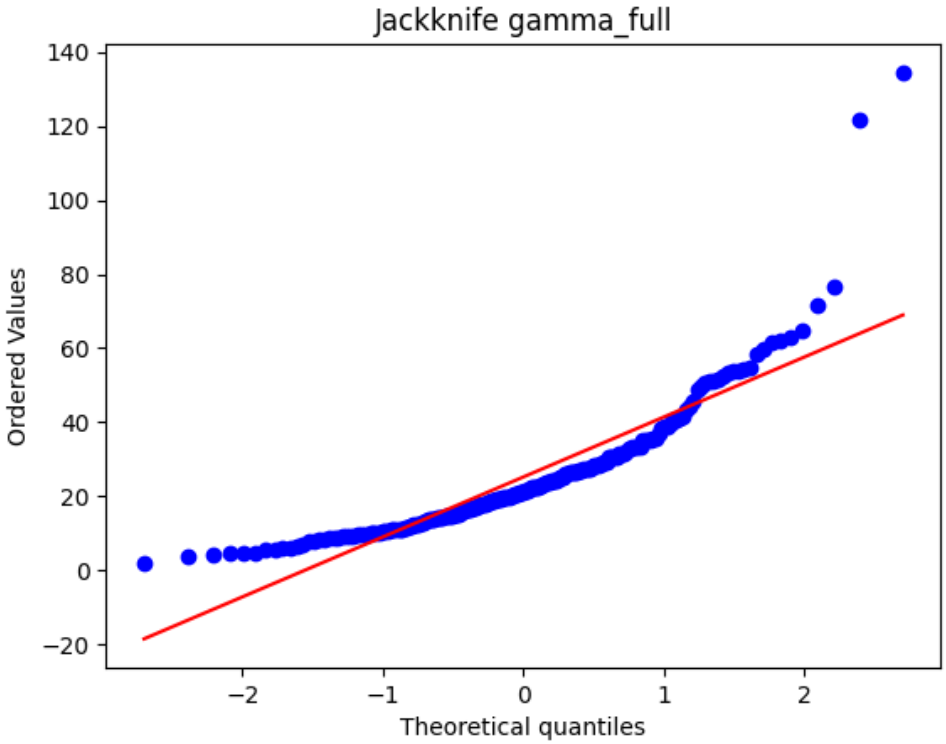
```
        gamma_full_list.append(gamma_full)
        gamma_a_list.append(gamma_a)
        gamma_lam_list.append(gamma_lam)

        if (rep+1) % 2 == 0 and verbose:
            print(f"MC rep {rep+1}/{MC_REPS} complete.")

    return np.array(gamma_full_list), np.array(gamma_a_list), np.
↪array(gamma_lam_list)
```

[ ]:
```
gamma_full, gamma_a, gamma_lam = run_mc_jackknife(
    N=20, K=3, R=1, T=200,
    MC_REPS=100,
    verbose=True
)

def type1_error(g):
    return np.mean(np.abs(g) > 1.96)
```

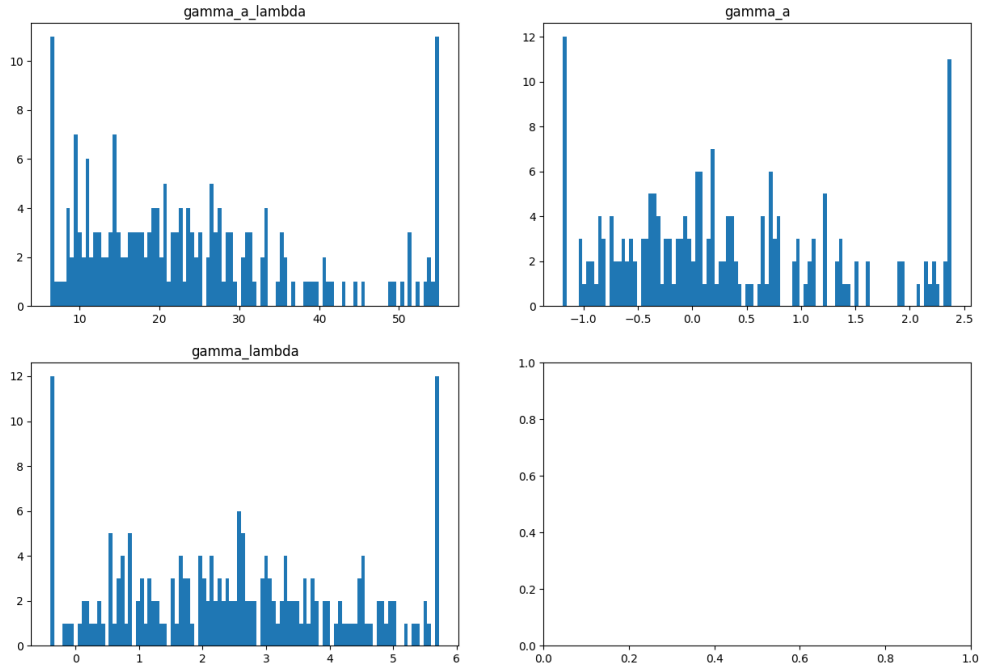
[79]:
```
print("Type-I error (full test):", type1_error(gamma_full))
print("Type-I error (alpha test):", type1_error(gamma_a))
print("Type-I error (lambda test):", type1_error(gamma_lam))
```

```
Type-I error (full test): 0.995
Type-I error (alpha test): 0.105
Type-I error (lambda test): 0.61
```

[60]:
```
stats.probplot(gamma_full, dist="norm", plot=plt)
plt.title("Jackknife gamma_full")
plt.show()
```



[58]:
```
fig, axes = plt.subplots(2, 2, figsize = (15, 10))
axes = np.ravel(axes)
labels = ['gamma_a_lambda', 'gamma_a', 'gamma_lambda']

for i, test_statistic in enumerate([gamma_full, gamma_a, gamma_lam]):
    axes[i].hist(
        utils.clean(test_statistic),
        bins = 100,
    )
    axes[i].set_title(labels[i])
```



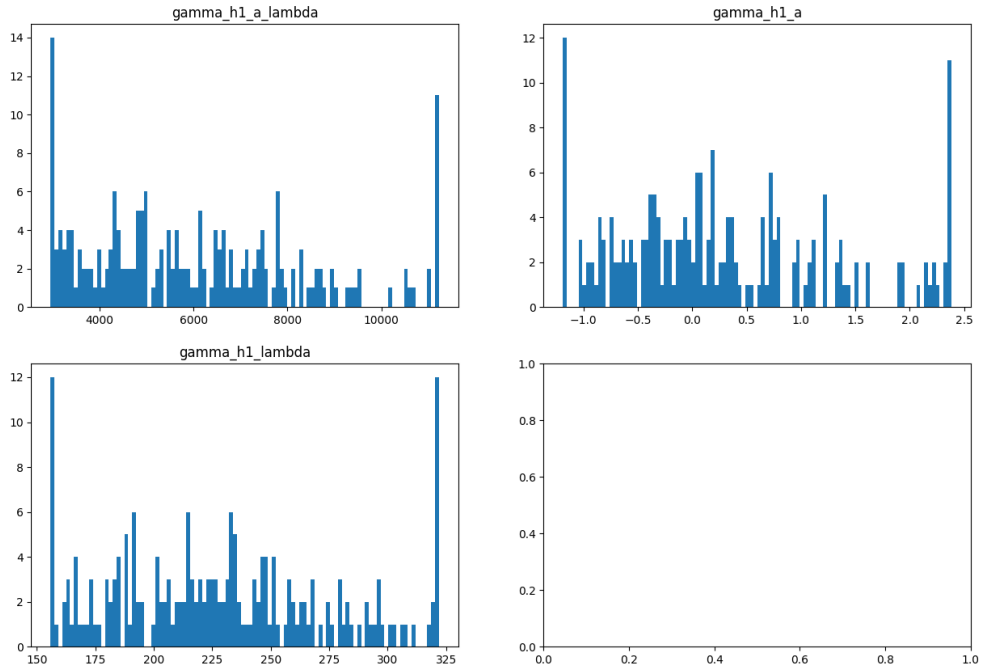
[87]:
```
gamma_h1_full, gamma_h1_a, gamma_h1_lam = run_mc_jackknife(
    N=20, K=3, R=1, T=200,
    MC_REPS=200,
    verbose=True,
    heterogeneiy_strength=1.0
)

def reject_rate(g):
    return np.mean(np.abs(g) > 1.96)


print("Power (full test):", reject_rate(gamma_h1_full))
print("Power (alpha test):", reject_rate(gamma_h1_a))
print("Power (lambda test):", reject_rate(gamma_h1_lam))
```

```
MC rep 2/200 complete.
MC rep 4/200 complete.
MC rep 6/200 complete.
MC rep 8/200 complete.
MC rep 10/200 complete.
MC rep 12/200 complete.
MC rep 14/200 complete.
MC rep 16/200 complete.
```

```
MC rep 18/200 complete.
MC rep 20/200 complete.
MC rep 22/200 complete.
MC rep 24/200 complete.
MC rep 26/200 complete.
MC rep 28/200 complete.
MC rep 30/200 complete.
MC rep 32/200 complete.
MC rep 34/200 complete.
MC rep 36/200 complete.
MC rep 38/200 complete.
MC rep 40/200 complete.
MC rep 42/200 complete.
MC rep 44/200 complete.
MC rep 46/200 complete.
MC rep 48/200 complete.
MC rep 50/200 complete.
MC rep 52/200 complete.
MC rep 54/200 complete.
MC rep 56/200 complete.
MC rep 58/200 complete.
MC rep 60/200 complete.
MC rep 62/200 complete.
MC rep 64/200 complete.
MC rep 66/200 complete.
MC rep 68/200 complete.
MC rep 70/200 complete.
MC rep 72/200 complete.
MC rep 74/200 complete.
MC rep 76/200 complete.
MC rep 78/200 complete.
MC rep 80/200 complete.
MC rep 82/200 complete.
MC rep 84/200 complete.
MC rep 86/200 complete.
MC rep 88/200 complete.
MC rep 90/200 complete.
MC rep 92/200 complete.
MC rep 94/200 complete.
MC rep 96/200 complete.
MC rep 98/200 complete.
MC rep 100/200 complete.
MC rep 102/200 complete.
MC rep 104/200 complete.
MC rep 106/200 complete.
MC rep 108/200 complete.
MC rep 110/200 complete.
MC rep 112/200 complete.
```

```
MC rep 114/200 complete.
MC rep 116/200 complete.
MC rep 118/200 complete.
MC rep 120/200 complete.
MC rep 122/200 complete.
MC rep 124/200 complete.
MC rep 126/200 complete.
MC rep 128/200 complete.
MC rep 130/200 complete.
MC rep 132/200 complete.
MC rep 134/200 complete.
MC rep 136/200 complete.
MC rep 138/200 complete.
MC rep 140/200 complete.
MC rep 142/200 complete.
MC rep 144/200 complete.
MC rep 146/200 complete.
MC rep 148/200 complete.
MC rep 150/200 complete.
MC rep 152/200 complete.
MC rep 154/200 complete.
MC rep 156/200 complete.
MC rep 158/200 complete.
MC rep 160/200 complete.
MC rep 162/200 complete.
MC rep 164/200 complete.
MC rep 166/200 complete.
MC rep 168/200 complete.
MC rep 170/200 complete.
MC rep 172/200 complete.
MC rep 174/200 complete.
MC rep 176/200 complete.
MC rep 178/200 complete.
MC rep 180/200 complete.
MC rep 182/200 complete.
MC rep 184/200 complete.
MC rep 186/200 complete.
MC rep 188/200 complete.
MC rep 190/200 complete.
MC rep 192/200 complete.
MC rep 194/200 complete.
MC rep 196/200 complete.
MC rep 198/200 complete.
MC rep 200/200 complete.
Power (full test): 1.0
Power (alpha test): 0.105
Power (lambda test): 1.0
```

[88]:
```
fig, axes = plt.subplots(2, 2, figsize = (15, 10))
axes = np.ravel(axes)
labels = ['gamma_h1_a_lambda', 'gamma_h1_a', 'gamma_h1_lambda']

for i, test_statistic in enumerate([gamma_h1_full, gamma_h1_a, gamma_h1_lam]):
    axes[i].hist(
        utils.clean(test_statistic),
        bins = 100,
    )
    axes[i].set_title(labels[i])
```



1.4 Jackknife Attempt #2

[76]:
```
def mc_null_jackknife_theoretical_avar_tests(
    N=20, K=3, R=1, T=200,
    MC_REPS=50,
    n_iter_est=400,
    seed=1234,
    heterogeneity_strength=0.0
):
    rng = np.random.default_rng(seed)
```

```
    gamma_full_obs      = np.zeros(MC_REPS)
    gamma_alpha_obs     = np.zeros(MC_REPS)
    gamma_lambda_obs    = np.zeros(MC_REPS)

    gamma_full_resample   = np.zeros((MC_REPS, T))
    gamma_alpha_resample  = np.zeros((MC_REPS, T))
    gamma_lambda_resample = np.zeros((MC_REPS, T))

    for mc in range(MC_REPS):
        print(f"Processing {mc}", end = '\r')
        rep_seed = rng.integers(1_000_000_000)

        # Simulate under the NULL
        beta_true, r, realized_cov, residuals, G_true, beta_star_true,␣
↪lambda_true = utils.simulate_dgp(
            N=N, K=K, R=R, T=T,
            heterogeneity_strength=heterogeneity_strength,
            seed=rep_seed
        )

        # FULL-SAMPLE ESTIMATE
        eta_full, G_full, beta_star_full, obj_full = utils.
↪iterative_convergence(
            beta_hat=beta_true,
            excess_returns=r,
            N=N, K=K, R=R, T=T,
            n_iter=n_iter_est,
            verbose=False
        )

        avar_full = utils.estimate_avar(
            beta_hat=beta_true,
            excess_returns=r,
            eta=eta_full,
            G=G_full,
            beta_star=beta_star_full,
            realized_covariance=realized_cov,
            residuals=residuals,
            N=N, K=K, R=R, T=T
        )

        # Full-sample test stats
        gamma_full_obs[mc] = utils.full_homogeneity_test(eta_full, avar_full,␣
↪N, K, T)
        gamma_alpha_obs[mc] = utils.intercept_homogeneity_test(eta_full,␣
↪avar_full, N, K, T)
```

```
        gamma_lambda_obs[mc] = utils.slope_homogeneity_test(eta_full,␣
↪avar_full, N, K, T)

        # jackknife resampling
        for t in range(T):

            mask = np.ones(T, dtype=bool)
            mask[t] = False

            beta_sub = beta_true[:, :, mask]    # (N, K, T-1)
            r_sub = r[:, mask]                  # (N, T-1)
            realized_cov_sub = realized_cov[:, :, mask]
            residuals_sub = residuals[:, :, mask]

            # Re-estimate parameters on the resampled dataset
            eta_jk, G_jk, beta_star_jk, obj_jk = utils.iterative_convergence(
                beta_hat=beta_sub,
                excess_returns=r_sub,
                N=N, K=K, R=R, T=T-1,
                n_iter=n_iter_est,
                verbose=False
            )

            avar_jk = utils.estimate_avar(
                beta_hat=beta_sub,
                excess_returns=r_sub,
                eta=eta_jk,
                G=G_jk,
                beta_star=beta_star_jk,
                realized_covariance=realized_cov_sub,
                residuals=residuals_sub,
                N=N, K=K, R=R, T=T-1
            )

            # Test statistics for leave-one-out
            gamma_full_resample[mc, t] = utils.full_homogeneity_test(eta_jk,␣
↪avar_jk, N, K, T-1)
            gamma_alpha_resample[mc, t] = utils.
↪intercept_homogeneity_test(eta_jk, avar_jk, N, K, T-1)
            gamma_lambda_resample[mc, t] = utils.slope_homogeneity_test(eta_jk,␣
↪avar_jk, N, K, T-1)

    return (
        gamma_full_obs, gamma_alpha_obs, gamma_lambda_obs,
        gamma_full_resample, gamma_alpha_resample, gamma_lambda_resample
    )
```

[69]:
```
gamma_full_obs, gamma_alpha_obs, gamma_lambda_obs, \
gamma_full_resample, gamma_alpha_resample, gamma_lambda_resample \
    = mc_null_jackknife_theoretical_avar_tests()
```

```
Size (full test): 0.1
Size (alpha test): 0.08
Size (lambda test): 0.06
```

[70]:
```
def size_from_resamples_two_sided(g_obs, g_res):
    """
    g_obs: shape (MC_REPS,)
    g_res: shape (MC_REPS, T) - each row is T resampled statistics
    """
    lower = np.percentile(g_res, 2.5, axis=1)   # per MC rep
    upper = np.percentile(g_res, 97.5, axis=1)

    reject = (g_obs < lower) | (g_obs > upper)

    return np.mean(reject)

print("Size (full test):", size_from_resamples_two_sided(gamma_full_obs,␣
 ↪gamma_full_resample))
print("Size (alpha test):", size_from_resamples_two_sided(gamma_alpha_obs,␣
 ↪gamma_alpha_resample))
print("Size (lambda test):", size_from_resamples_two_sided(gamma_lambda_obs,␣
 ↪gamma_lambda_resample))
```

```
Size (full test): 0.08
Size (alpha test): 0.06
Size (lambda test): 0.04
```

[81]:
```
gamma_h1_full_obs, gamma_h1_alpha_obs, gamma_h1_lambda_obs, \
gamma_h1_full_resample, gamma_h1_alpha_resample, gamma_h1_lambda_resample \
    = mc_null_jackknife_theoretical_avar_tests(
        heterogeneity_strength=10
    )
```

```
Processing 49
```

[82]:
```
print("When heterogeneity is 1.0")
print("Power (full test):", size_from_resamples_two_sided(gamma_h1_full_obs,␣
 ↪gamma_h1_full_resample))
print("Power (alpha test):", size_from_resamples_two_sided(gamma_h1_alpha_obs,␣
 ↪gamma_h1_alpha_resample))
print("Power (lambda test):",␣
 ↪size_from_resamples_two_sided(gamma_h1_lambda_obs, gamma_h1_lambda_resample))
```

```
When heterogeneity is 1.0
Power (full test): 0.02
```

```
Power (alpha test): 0.02
Power (lambda test): 0.02
```