In [ ]:
```
import numpy as np
from scipy.integrate import quad
import scipy.stats as stats
import pandas as pd
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
```

Econometrics Final Project - Monte-Carlo Replication of Stock and Watson (2008)

Author: Jiacheng Li
Date: April 20, 2022

This notebook replicates the Monte-Carlo experiment of Stock and Watson (2008).

Part 1. Model

Consider the following fixed effect regression model.

$$Y_{it} = \alpha_i + \beta' X_{it} + u_{it}, \quad i = 1, \dots, n, \quad t = 1, \dots, T$$

First, we define the “Within”-transformed variables, i.e., from which the time deviation is subtracted, as

$$\tilde{X}_{it} = X_{it} - T^{-1} \sum_{s=1}^{T} X_{is}.$$

And suppose (X_{it}, u_{it}) follow the below assumptions.

Assumption 1. $\{(X_{it}, u_{it})\}_{t=1}^{T}$ are i.i.d. over $i = 1, \dots, n.$

Assumption 2. $\mathbb{E}\left[u_{it} | X_{it}, \dots, X_{iT}\right] = 0.$

Assumption 3. $Q_{\tilde{X}\tilde{X}} \equiv \mathbb{E}\left[\frac{1}{T} \sum_{t=1}^{T} \tilde{X}_{it} \tilde{X}_{it}'\right]$ is nonsingular.

Assumption 4. $\mathbb{E}\left[u_{it} u_{is} | X_{it}, \dots, X_{iT}\right] = 0$ for $t \neq s.$

Assumption 5. (X_{it}, u_{it}) is stationary and has absolutely summable cumulants up to order 12.

The fixed effect estimator is

$$\hat{\beta}_{\text{FE}} = \left(\sum_{i=1}^{n} \sum_{t=1}^{T} \tilde{X}_{it} \tilde{X}_{it}' \right)^{-1} \sum_{i=1}^{n} \sum_{t=1}^{T} \tilde{X}_{it} \tilde{Y}_{it}.$$

where $\hat{\tilde{u}}_{it}$ is the fixed effect regression residuals, i.e.,

$$\hat{\tilde{u}}_{it} = \hat{\tilde{Y}}_{it} - \hat{\beta}_{\text{FE}}' \tilde{X}_{it} = \tilde{u}_{it} - \left(\hat{\beta}_{\text{FE}} - \beta \right)' \tilde{X}_{it}.$$

The paper considers and compares the following three heteroskedasticity-robust (HR) covariance estimators for fixed effect panel data regression:

$$\begin{aligned} \hat{\Sigma}^{\text{HR-XS}} &= \frac{1}{nT - n - k} \sum_{i=1}^{n} \sum_{t=1}^{T} \tilde{X}_{it} \tilde{X}_{it}' \hat{\tilde{u}}_{it}^2 \\ \hat{\Sigma}^{\text{HR-FE}} &= \left(\frac{T-1}{T-2} \right) \left(\hat{\Sigma}^{\text{HR}-\text{XS}} - \frac{1}{T-1} \hat{B} \right) \\ \hat{\Sigma}^{\text{cluster}} &= \frac{1}{nT} \sum_{i=1}^{n} \left(\sum_{t=1}^{T} \tilde{X}_{it} \hat{\tilde{u}}_{it} \right) \left(\sum_{s=1}^{T} \tilde{X}_{is} \hat{\tilde{u}}_{is} \right)' \end{aligned}$$

where $\hat{B} = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{1}{T} \sum_{t=1}^{T} \tilde{X}_{it} \tilde{X}_{it}' \right) \left(\frac{1}{T-1} \sum_{s=1}^{T} \hat{\tilde{u}}_{is}^2 \right).$

Part 2. Monte Carlo Design

The benchmark Monte Carlo design as specified in section 2 of the paper is as follows:

$y_{it} = x_{it} \beta + u_{it}$

$x_{it} = \zeta_{it} + \theta \zeta_{it-1}, \quad \zeta_{it} \sim \text{i.i.d. } \mathrm{N}(0, 1), \quad t = 1, \dots, T,$

$u_{it} = \varepsilon_{it} + \theta \varepsilon_{it-1}, \quad \varepsilon_{it} \mid x_i \sim \text{i.n.i.d. } \mathrm{N}\left(0, \sigma_{it}^2\right), \quad \sigma_{it}^2 = \lambda \left(0.1 + x_{it}^2\right)^{\kappa}, t = 1, \dots, T,$

where $\kappa = 1, -1$ and λ is chosen so that $Var(\epsilon_{it}) = 1.$

Deriving λ_1, λ_2 in the Design

Next, we find λ_1, λ_2 for $\kappa = 1, -1.$

Here,

$$Var\left(\epsilon_{it}\right) = \mathbb{E}\left[Var\left(\epsilon_{it} | x_i\right)\right] + Var\left(\mathbb{E}\left[\epsilon_{it} | x_i\right]\right) = \mathbb{E}\left[Var\left(\epsilon_{it} | x_i\right)\right] = \lambda \mathbb{E}\left[\left(0.1 + x_{it}^2\right)^{\kappa}\right] = 1.$$

Since $\theta = 0, x_{it} = \zeta_{it} \sim \mathrm{N}\left(0, 1\right)$ and $\mathbb{E}\left[x_{it}^2\right] = Var\left(x_{it}\right) = 1.$

When $\kappa = 1,$

$$\lambda \mathbb{E}\left[\left(0.1 + x_{it}^2\right)\right] = \lambda \left(0.1 + \mathbb{E}\left[x_{it}^2\right]\right) = 1 \implies \lambda = \frac{1}{0.1 + 1} = \frac{10}{11}.$$

When $\kappa = -1,$

$$\lambda \mathbb{E}\left[\left(0.1 + x_{it}^2\right)^{-1}\right] = 1$$

where

$$\mathbb{E}\left[\frac{1}{\left(0.1 + x_{it}^2\right)}\right] = \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} \frac{1}{\left(0.1 + x_{it}^2\right)} e^{-\frac{x^2}{2}} dx, \quad x_{it} \sim \mathrm{N}\left(0, 1\right)$$

Observe the identity $\frac{1}{S} = \int_0^{\infty} e^{-tS} dt.$

This allows us to write

$$\mathbb{E}\left[\frac{1}{\left(0.1 + x_{it}^2\right)}\right] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \int_0^{\infty} e^{-t\left(0.1 + x_{it}^2\right)} e^{-\frac{x^2}{2}} dt dx$$

Since $e^{-t\left(0.1 + x_{it}^2\right)} e^{-\frac{x^2}{2}} > 0$, we can apply Fubini theorem, which yields,

$$\begin{aligned} \mathbb{E}\left[\frac{1}{\left(0.1 + x_{it}^2\right)}\right] &= \frac{1}{\sqrt{2\pi}} \int_0^{\infty} \int_{-\infty}^{\infty} e^{-t\left(0.1 + x_{it}^2\right)} e^{-\frac{x^2}{2}} dx dt \\ &= \frac{1}{\sqrt{2\pi}} \int_0^{\infty} e^{-0.1t} \left(\int_{-\infty}^{\infty} e^{-\left(t + \frac{1}{2}\right) x_{it}^2} dx \right) dt \\ &= \int_0^{\infty} e^{-0.1t} (1 + 2t)^{-\frac{1}{2}} dt \end{aligned}$$

In [ ]:
```
# define lambda1
lambda1 = 1.1

# evaluate integral
def f(x):
    return np.exp(-1*x)*(1+2*x)**(-1/2)
integ, err = quad(f, 0, np.inf)

# get lambda2
lambda2 = 1/integ
```

This can be evaluted numerically and we get

$$\lambda = \frac{1}{\int_0^{\infty} e^{-0.1t} (1 + 2t)^{-\frac{1}{2}} dt} \simeq 1.525135$$

Monte Carlo Experiments

As there is only one regressor and one regressand, it is easy to define both X and Y as a $N \times T$ matrix.

The procedure for conducting Monte Carlo experiments is as follows:

1. For each (n, T), make M Monte Carlo draws with the data-generating process parametrized as indicated above, for each $\kappa = 1, -1.$
2. Compute the three estimators: $\hat{\Sigma}^{\text{HR-XS}}$, $\hat{\Sigma}^{\text{HR-FE}}$, and $\hat{\Sigma}^{\text{cluster}}$.
3. Compute the Bias relative to the True β and MSE relative to the infeasible estimator:

$$\hat{\Sigma^{inf}} = (nT)^{-1} \sum_{i=1}^{n} \sum_{t=1}^{T} \tilde{X}_{it}^2 u_{it}^2.$$

4. Compute the rates at which the null hypothesis, $\beta = \beta_0$ is rejected, using a two-sided test at a 10\% critical level. Here we compute the t-statistic using each suggested variance estimator. For $\hat{\Sigma}^{\text{HR-XS}}$, $\hat{\Sigma}^{\text{HR-FE}}$, the critical value comes from a normal distribution; and for $\hat{\Sigma}^{\text{cluster}}$, it follows the $\sqrt{\frac{n}{n-1}} t_{n-1}$ distribution.
5. Summarize the results.

Note: To simplify data generating and processing in a programming language, since the benchmark and follow-up Monte Carlo designs all only contain one single regressor, the panel data structure with double indexes allow us to store each variable in a matrix.

I will create $(X, Y, u, \zeta, \epsilon, \sigma)$ that are two-dimensional matrices where (Z_{it}) represents the i'th row, j's column element, which is the observation of individual i in $t = j$.

First, I build the following functions that compute the Fixed Effect estimator and the proposed three types of HR variance covariance estimators.

In [ ]:
```
# within operator
def within(Z):
    """
    Computes the resulting matrix by taking the deviation from the time average
    """
    # get dimension
    (N,T) = Z.shape

    # first broadcast
    time_mean = 1/T * Z.sum(axis=1, keepdims=True)
    bc = np.tile(
        time_mean, reps=(1,T)
    )

    # then within
    return Z - bc


# create functions to calculate FE estimator and HR covariance
def Fixed_Effect(X,Y):
    """
    Computes the fixed effect estimator given Y and X
    """
    # get dimension
    (N,T) = X.shape

    # within transformation
    X_tild, Y_tild = within(X), within(Y)

    # compute FE estimator
    beta_FE = np.sum(X_tild*Y_tild) / np.sum(X_tild**2)

    # compute residual
    res_FE = Y_tild - beta_FE*X_tild
    return beta_FE, res_FE


def HR_XS(res_FE, X_tild):
    """
    Computes the HR-XS heteroskedasticity-robust (HR) covariance estimator
    """
    # get dimension
    (N,T) = X_tild.shape

    # compute HR-XS vcov_hat
    vcov_hat = 1/(N*T-N-1) * np.sum((X_tild**2)*(res_FE**2))
    return vcov_hat


def HR_FE(res_FE, X_tild):
    """
    Computes the HR-FE heteroskedasticity-robust (HR) covariance estimator
    """
    # get dimension
    (N,T) = X_tild.shape

    # get HR_XS vcov_hat
    HR_XS_hat = HR_XS(res_FE, X_tild)

    # calculate B_hat
    B_hat1 = 1/T * np.sum(X_tild**2, axis=1, keepdims=True)
    B_hat2 = 1/(T-1) * np.sum(res_FE**2, axis=1, keepdims=True)

    B_hat = 1/N * np.sum(B_hat1 * B_hat2, axis=0)

    # compute HR-FE vcov_hat
    vcov_hat = ((T-1)/(T-2)) * (HR_XS_hat-1/(T-1)*B_hat)
    return vcov_hat


def cluster(res_FE, X_tild):
    """
    Computes the clustered covariance estimator
    """
    # get dimension
    (N,T) = X_tild.shape

    # this yields a N by 1 vector
    T_sum = np.sum(X_tild*res_FE, axis=1)

    # compute clustered vcov_hat
    vcov_hat = 1/(N*T) * np.sum(T_sum**2)
    return vcov_hat
```

Next, I write a function that generates a Monte Carlo draw for a given (n, T, κ) tuple.

Notice that the true variance is given by

$$\begin{aligned} \Sigma &= \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\tilde{X}_{it} \tilde{X}_{it}' u_{it}^2\right] \\ &= \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\tilde{X}_{it} \tilde{X}_{it}' \mathbb{E}\left[u_{it}^2 | X_{it}\right]\right] \\ &= \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\tilde{X}_{it}^2 \sigma_{it}^2\right] \\ &= \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\left(X_{it} - \frac{1}{T} \sum_{s=1}^{T} X_{is} \right)^2 \lambda \left(0.1 + X_{it}^2\right)^{\kappa}\right] \end{aligned}$$

since in our design X_{it} is a scalar. This can be estimated below using a large sample to avoid analytical technicalities. That is, we separately simulate a $(X, \tilde{X}, \sigma)$ tuple with a very large n and appeal to the Law of Large Numbers to evalute the expectation through approximation. This is achieved by the following program.

In [ ]:
```
def var_true(kappa, T):
    """
    Computes the true variance of beta using large sample approximation
    """
    # create matrix
    N = 10000
    X = np.random.normal(loc=0, scale=1, size=[N,T])
    X_tild = within(X)

    # define kappa
    if kappa==1:
        Lambda = lambda1
    elif kappa==-1:
        Lambda = lambda2

    # compute variance by appealing to LLN
    To_sum = X_tild**2 * Lambda * (0.1 + X**2)**kappa
    var = 1/T * 1/N * np.sum(To_sum)

    return var
```

And we propose the infeasible estimator

$$\hat{\Sigma}^{\text{inf}} = \frac{1}{nT} \sum_{i=1}^{n} \sum_{t=1}^{T} \tilde{X}_{it}^2 u_{it}^2$$

where u_{it} comes from the true data-generating process.

For each Monte Carlo draw, we compute:

- the Bias relateive to the true Σ of each estimator,
- the Mean Squared Error (MSE) relative to the infeasible estimator, and
- the rejection rates under the null hypothesis of the two-sided test of $\beta = \beta_0$ based on the t-statistic computed using the indicated variance estimator and the 10% asymptotic critical value.

Note: using $\hat{\Sigma}^{\text{HR-XS}}$ and $\hat{\Sigma}^{\text{HR-FE}}$, the critical value is from the standard normal distribution, using $\hat{\Sigma}^{\text{cluster}}$, it is from the $\sqrt{\frac{n}{n-1}} t_{n-1}$ distribution.

Below I create functions that

- compute the variance of ϵ_{it} in the DGP conditional on x_{it};
- compute the true variance and infeasible estimator respectively;
- compute the mean bias and MSE;
- create a hypothesis testing procedure that takes a given variance estimator as input and computes the rejection rate under the null hypothesis and proposed criterion and test statistic.

For hypothesis testing, recall that

$$\sqrt{nT} \left(\hat{\beta}_{\text{FE}} - \beta \right) \xrightarrow{d} \mathrm{N}\left(0, Q_{\tilde{X}\tilde{X}}^{-1} \Sigma Q_{\tilde{X}\tilde{X}}^{-1}\right)$$

where $\quad \Sigma = \frac{1}{T} \sum_{t=1}^{T} E\left(\tilde{X}_{it} \tilde{X}_{it}' u_{it}^2\right)$. This implies

$$\hat{\beta}_{FE} \sim N\left(\beta, \frac{1}{nT} Q_{\tilde{X}\tilde{X}}^{-1} \Sigma Q_{\tilde{X}\tilde{X}}^{-1}\right)$$

And,

$$\frac{\hat{\beta}_{FE} - \beta}{\sqrt{\frac{1}{nT} Q_{\tilde{X}\tilde{X}}^{-1} \Sigma Q_{\tilde{X}\tilde{X}}^{-1}}} \sim N\left(0, 1\right)$$

where we replace Q, Σ by the sample counterpart and estimators to evaluate rejection size.

In [ ]:
```
def var_eps(Lambda, kappa, x):
    """
    Computes the variance of eps | x
    """
    return Lambda*(0.1+x**2)**kappa


def var_inf(u, X_tild):
    """
    Compute the proposed infeasible variance estimator
    """
    # get dimension
    (N,T) = X_tild.shape

    mat = (X_tild**2) * (u**2)
    return 1/(N*T)*np.sum(mat)


def mean_bias(cov_hat, cov, N):
    """
    Compute the mean bias of estimated variance
    """
    return np.sum(cov_hat - cov) / N


def RMSE(cov_hat, cov, N):
    """
    Compute the RMSE of estimated variance
    """
    return np.sqrt(np.square(cov_hat - cov).sum() / N)


def hypo_test(
    var_hat, X_tild, beta_hat, beta,
    est_name    # need estimator name to specify asymptotic distributions
    ):
    """
    Conduct hypothesis testing for each estimator based on its asymptotic distribution
    """
    # get dimension
    (N,T) = X_tild.shape

    Q_hat = 1/(N*T) * np.sum(X_tild**2)

    # compute t-statistic
    t_stat = (beta_hat-beta) / np.sqrt(
        1/(N*T) * (1/Q_hat) * var_hat * (1/Q_hat)
    )

    # compute critical value at 10%
    if est_name=='HR-XS':
        crit_val = stats.norm.ppf(1-0.1/2)

    elif est_name=='HR-FE':
        crit_val = stats.norm.ppf(1-0.1/2)

    elif est_name=='cluster':
        crit_val = stats.t.ppf(
            q=1-0.1/2,
            df=N-1,
            scale=np.sqrt(N/(N-1))  # rescaled t
            )

    # reject or not
    if abs(t_stat) >= crit_val:
        reject = 1
    else:
        reject = 0

    return reject
```

Finally, I can pack them into a function that generates Monte Carlo draws and calculates Bias, MSE, and rejection size.

The draw is taken as follows:

1. draw $x_{it} = \zeta_{it} \sim N(0, 1)$;
2. draw $u_{it} = \epsilon_{it} \sim N(0, \sigma_{it}^2), \sigma_{it}^2 = \lambda(0.1 + x_{it}^2)^{\kappa}$;
3. compute $y_{it} = \beta x_{it} + u_{it}$.

These can be obtained through element-wise matrix operation since all variables are in matrices of the same size to improve efficiency.

In [ ]:
```
def MC_draw(N, T, kappa, M):
    """
    Generate a Monte Carlo draw for a given sample size N, time T, and kappa.
    For the replication, impose beta = theta = 0.
    """

    # since these are relative to the infeasible ones contingent on each sample
    total_bias = np.zeros([3,1])

    # sum of squared error of each estimator and infeasible estimator
    SE_est = np.zeros([3,1])
    SE_inf = 0

    # num of rejections
    rej_count = np.zeros([3,1])

    # compute true variance
    true_var = var_true(kappa, T)

    # draw M times for each design
    for num in range(1,M):

        # a matrix X of N(0,1) realizations of size N by T
        X = np.random.normal(
            loc=0, scale=1, size=(N,T)
            )

        # define kappa
        if kappa==1:
            Lambda = lambda1
        elif kappa==-1:
            Lambda = lambda2

        # a matrix X of N(0,sigma^2) realizations of size N by T
        U = np.random.normal(
            # mean and standard deviation are matrices because conditional on X
            loc=np.zeros([N,T]),
            scale=np.sqrt(Lambda * (0.1 + X**2) ** kappa),
            size=(N,T)
            )

        # define beta
        beta = 0

        # define Y
        Y = beta*X + U

        # NEXT, calculate FE estimator
        X_tild = within(X)
        Y_tild = within(Y)
        beta_FE, res_FE = Fixed_Effect(X,Y)

        # compute three types of HR variance estimator
        HR_XS_hat = HR_XS(res_FE, X_tild)
        HR_FE_hat = HR_FE(res_FE, X_tild)
        cluster_hat = cluster(res_FE, X_tild)

        # compute infeasible estimators
        inf_var = var_inf(U, X_tild)

        # cumulative bias and RMSE
        total_bias[0] += HR_XS_hat-true_var
        total_bias[1] += HR_FE_hat-true_var
        total_bias[2] += cluster_hat-true_var

        SE_est[0] += (HR_XS_hat-true_var)**2
        SE_est[1] += (HR_FE_hat-true_var)**2
        SE_est[2] += (cluster_hat-true_var)**2

        SE_inf += (inf_var - true_var)**2

        # rejection count
        rej_XS = hypo_test(HR_XS_hat, X_tild, beta_FE, beta, est_name='HR-XS')
        rej_FE = hypo_test(HR_FE_hat, X_tild, beta_FE, beta, est_name='HR-FE')
        rej_cluster = hypo_test(cluster_hat, X_tild, beta_FE, beta, est_name='cluster')

        rej_count[0] += rej_XS
        rej_count[1] += rej_FE
        rej_count[2] += rej_cluster

    # calculate total mean bias
    bias = total_bias / M
    bias_ratio = bias/true_var

    # MSE ratio
    MSE_est = SE_est / M
    MSE_inf = SE_inf / M
    MSE_ratio = MSE_est / MSE_inf

    # rejection rate
    rej_rate = rej_count/M

    return bias_ratio, MSE_ratio, rej_rate
```

The Monte Carlo experiment is executed in the following block of the codes. I replicate Table 1 in the Stock and Watson paper.

In [ ]:
```
# define design parameters
Ns = [20, 100, 500]
Ts = [5, 10, 20, 50]
kappas = [-1, 1]

NumDraws = 50000

# define arrays to store outcomes
output = np.zeros([4*6, 3*3+3])

row = 0
# loop
for i in range(len(Ns)):
    for j in range(len(Ts)):
        for k in range(len(kappas)):
            N = Ns[i]
            T = Ts[j]
            kappa = kappas[k]

            # perform Monte Carlo draws
            (bias_ratio, MSE_ratio, rej_rate) = MC_draw(
                N=N, T=T, kappa=kappa, M=NumDraws
                )

            # define labels
            order = np.array([kappa, T, N], ndmin=2).T
            temp = np.concatenate([
                order, bias_ratio, MSE_ratio, rej_rate
            ])

            # store
            output[row,:] = np.squeeze(temp)
            row += 1

table = pd.DataFrame(data=output)
```

```
<ipython-input-153-5b5c7b102320>:38: RuntimeWarning: invalid value encountered in reciprocal
  scale=np.sqrt(Lambda * (0.1 + X**2) ** kappa),
<ipython-input-152-1d23eab67d2b>:69: RuntimeWarning: invalid value encountered in sqrt
  t_stat = (beta_hat-beta) / np.sqrt(
```

In [ ]:
```
# sort and beautify table of outcomes
colnames = ['kappa', 'T', 'n',
    'Bias ratio: HR-XS', 'Bias ratio: HR-FE', 'Bias ratio: cluster',
    'MSE ratio: HR-XS', 'MSE ratio: HR-FE', 'MSE ratio: cluster',
    'Rej size: HR-XS', 'Rej size: HR-FE', 'Rej size: cluster',
    ]

# change column names and indexes
table.columns = colnames

# set integer to indexes
table[['kappa', 'T', 'n']] = table[['kappa', 'T', 'n']].astype(int)
table = table.sort_values(by=['kappa', 'T', 'n'],ascending=[False, True, True]).set_index(['kappa'
table.columns = pd.MultiIndex.from_product([
    ['Bias Relative to True', 'MSE Relative to Infeasible', 'Size (Nominal Level 10%)'], ['HR-XS',
    ])

# print
table
```

Out[ ]:

			Bias Relative to True			MSE Relative to Infeasible			Size (Nominal Level 10%)		
			HR-XS	HR-FE	cluster	HR-XS	HR-FE	cluster	HR-XS	HR-FE	cluster
kappa	**T**	**n**									
1	**5**	**20**	-0.167018	-0.065950	-0.109483	0.701211	0.864549	0.994899	0.15294	0.13296	0.12620
		100	-0.120409	-0.011068	-0.020611	0.936569	1.014448	1.196657	0.12908	0.10782	0.10666
		500	-0.097951	0.014668	0.013002	1.411187	1.055551	1.258072	0.12408	0.10180	0.10220
	10	**20**	-0.086640	-0.027604	-0.073949	0.825495	0.925673	1.379785	0.12490	0.11422	0.10864
		100	-0.053785	0.007884	-0.002088	0.928237	1.000293	1.478599	0.11298	0.10212	0.10100
		500	-0.075081	-0.014710	-0.016728	1.682807	1.030170	1.553747	0.11190	0.10110	0.10120
	20	**20**	-0.054336	-0.023669	-0.072848	0.920910	0.967471	2.013986	0.11354	0.10840	0.10674
		100	-0.028093	0.003573	-0.006545	0.967727	1.001680	2.104807	0.10798	0.10242	0.10266
		500	-0.036313	-0.004893	-0.006970	1.320987	1.012897	2.142078	0.10706	0.10176	0.10196
	50	**20**	-0.019969	-0.007276	-0.057544	0.965326	0.986577	3.796865	0.10510	0.10302	0.10248
		100	-0.015444	-0.002676	-0.013090	1.005910	1.002983	3.921963	0.10140	0.09924	0.09964
		500	-0.011885	0.000932	-0.001254	1.078416	1.004479	3.957466	0.10290	0.10082	0.10024
-1	**5**	**20**	0.333925	0.025212	-0.029188	3.270491	1.721855	1.989092	0.06080	0.10756	0.09296
		100	0.306204	-0.002298	-0.012990	8.326519	1.656508	2.029251	0.06006	0.10210	0.09988
		500	0.324765	0.009970	0.008042	37.177129	1.626602	2.026086	0.06068	0.10220	0.10080
	10	**20**	0.241587	0.009209	-0.044411	3.904450	1.550604	4.434379	0.06604	0.09922	0.09564
		100	0.236490	0.003100	-0.006956	12.582775	1.506074	4.589099	0.06728	0.09872	0.09786
		500	0.230873	-0.001936	-0.004140	54.281702	1.502253	4.667522	0.06772	0.09904	0.09898
	20	**20**	0.149439	0.006165	-0.044966	3.725885	1.377615	10.766868	0.07656	0.09846	0.09886
		100	0.139187	-0.003329	-0.013287	11.456702	1.362160	10.933831	0.07782	0.09938	0.09974
		500	0.145593	0.002108	0.000028	55.548518	1.364820	10.880381	0.07846	0.09898	0.09908
	50	**20**	0.067084	0.002064	-0.046242	2.684599	1.209185	31.977461	0.08838	0.09868	0.09900
		100	0.065462	0.000332	-0.009816	8.132601	1.199180	32.411497	0.08832	0.09864	0.09922
		500	0.066681	0.001454	-0.000865	36.540262	1.195063	32.382748	0.08880	0.09832	0.09796

Discussion

Indeed, the above exercise replicates the Monte Carlo report of Stock and Watson (2008) and produces results of the same magnitude and directions. This also verifies my codes.

The bias of the usually used $\hat{\Sigma}^{\text{HR-XS}}$ estimator can have a large and persistent bias that persists even if n increases. And the large bias can produce a large MSE. But in some cases its MSE is smaller than the MSE of the infeasible estimator when n and T are small. This illustrates a bias-variance trade-off.

The proposed bias correction estimator $\hat{\Sigma}^{\text{HR-FE}}$ works - the relative bias in all cases are small and in most cases its MSE is very close to the one of the infeasible estimator.

The ratio of the MSE of the cluster estimator as expected do not converge to 1 as n increases for a fixed T. The rejection results also confirm the conjecture that variance estimators with less bias would produce better size for the hypothesis test $\beta = 0$. When $\hat{\Sigma}^{\text{HR-XS}}$ is biased up, the test rejects too little, and when $\hat{\Sigma}^{\text{HR-XS}}$ is biased down, the test rejects too often. When T is small, the magnitude of distortion can be quite considerable.

Reference

Stock, J.H. and Watson, M.W. (2008), Heteroskedasticity-Robust Standard Errors for Fixed Effects Panel Data Regression. Econometrica, 76: 155-174.