Time-varying Minimum Variance Portfolio: A User's Manual

Qingliang Fan*[a], Ruike Wu[b], Yanrong Yang[c], and Wei Zhong[b]

[a]Department of Economics, The Chinese University of Hong Kong
[b]WISE & School of Economics, Xiamen University
[c]College of Business & Economics, The Australian National University

June 13, 2022

Abstract

In this document, we provide detailed step by step implementation procedure for the paper "Time-varying Minimum Variance Portfolio" (Fan et al. 2022). We first introduce some main functions including estimating time-varying factor loading, time-varying covariance matrix(also sparse residual covariance matrix estimation), rolling window procedure, and the two-step hypothesis test in the main paper. Further, we introduce some auxiliary functions, such as the tools for selecting values of various tuning parameters and some functions used in listed examples. Finally, we provide a large example to show a whole picture of our codes. All example codes could be found in the file *main_process.m* for users' convenience.

*Correspondence to: Department of Economics, 903, Esther Lee Building, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong. E-mail addresses: michaelqfan@cuhk.edu.hk (Q. Fan), wrkworld@163.com (R. Wu), wzhong@xmu.edu.cn (W. Zhong), yanrong.yang@anu.edu.au (Y. Yang).

Contents

1 Preliminary setting

Before running provided code, some preliminary setting need to be done for adapting to the local environment. We recommend that you put the provided folder 'TV-MVP' under the folder 'bin' in MATLAB.

(1). install cvx in matlab(we provide the cvx toolbox named "cvx.rar", the user only need to unzip the file, and run *cvx_setup.m* in MATLAB command window)

(2). install packages 'spcov', 'PDSCE' and 'R.matlab' locally in R.

(3). Open 'Rspcov.bat' in text, edit the first path to the location of local 'Rscript.exe', e.g. $F : /R - 3.6.1/bin/Rscript.exe$, and the user need to replace $F : /R - 3.6.1/bin/$ by their local path. And further edit the second path to the location of 'TV-MVP' folder, e.g. $F : /matlab/bin/TV - MVP/spcov\_test.R$, and the user need to replace $F : /matlab/bin/TV - MVP$ by their local path. We use 'bat' to run the code written by R and 'bat' file called by MATLAB.

(4). Open MATLAB, and change the working path to current folder, e.g. $F : /matlab/bin/TV - MVP/$

2 Main Functions

2.1 $Timevarying\_factor\_model.m$

Description: This function compute time-varying factor loading estimator and corresponding estimated factors discussed in section 3.1. For this function, there are some other outputs which are intermediate products used by hypothesis test function.

Usage:

$$[Factor\_loadings, Factor\_set] = Timevarying\_factor\_model(R, f\_number, Kernel\_set, h)$$

Input:

(1). R is $N \times T$ excess return matrix, N is dimension, T is sample size

(2). f_number is an integer, the number of factor.

(3). $Kernel\_set$ is pre-calculating kernel weighting matrix by using bandwidth h, see the following example to learn how to calculate, and the details of underlying kernel function can be referred to function $Kernel2$ in our "Other" section.

(4). h is bandwidth used in calculating $Kernel\_set$.

Output:

(1). $Factor\_loading$ is $T \times 1$ cell, each element of the cell is $f\_number \times N$ estimated time-varying factor loadings at each time point.

(2). $Factor\_set$ is a $f\_number \times T$ matrix, and each row is the estimated factors given in section 3.1 in the main paper.

1. Example: In this example, we show how to use Function *Timevarying_factor_model* to estimate factor loadings and factors nonparametrically. we generate virtual data by using smooth time-varying factor loading and residuals from multivariate normal distribution with cross-dependence covariance matrix. Dimension, sample size and ρ(control the departure from constant factor loading)are set to be 50, 250 and 0.7, respectively. The number of factors is set to 2, with AR(1)-type factors whose unconditional variances are both 1. During the estimating, the number of factors is estimated by BIC-type information criterion introduced in B.3 of our appendix, and rule of thumb bandwidth is used. For result, firstly, the estimated factor number K is exactly equal to 2. And the first estimated loading is shown in figure 1, one can observe that large variation is featured by factor loading estimator.

```
% Basic setting
clear; clc;
rng(2);% set the random seed
d = 50; T = 250; % set the dimension and sample size
rho = 0.7; % set the departure from the null.
h = (2.35/sqrt(12)) * T^(-0.2) * d^(-0.1); % set rule of thumb bandwidth

% generate data by using smooth time varying factor loading and cross-sectional depen-
dence % setting defined in appendix C.2.1
[Y, ~, s_cov] = data_generate(d, T, rho);

% Pre-calculating all kernel weights to increase computing speed.
Kernel_set = zeros(T, T); % Reserve space for storing kernel weights
for r = 1 : T
for t = 1 : T
    % see an introduction for Function Kernel2 in the 'Others' section
    k = (h^-1) * Kernel2(T, t, r, h);
    Kernel_set(t, r) = k^0.5;
end
end

% Using BIC-type information criterion to select the number of factors
% max number is set to 10, see more details of Function factor_number_selection in 'Others'
section
K = factor_number_selection(Y, 10);

% Estimate factor loading and corresponding factors
[Factor_loadings, Factor] = Timevarying_factor_model(Y, K, Kernel_set, h);

% plot the loading for first common factor in time series.
Fir_loading = [];
for i = 1 : T
    loading_temp = Factor_loadings{i};
    Fir_loading(i, :) = loading_temp(1, :);
end
plot(Fir_loading)
```

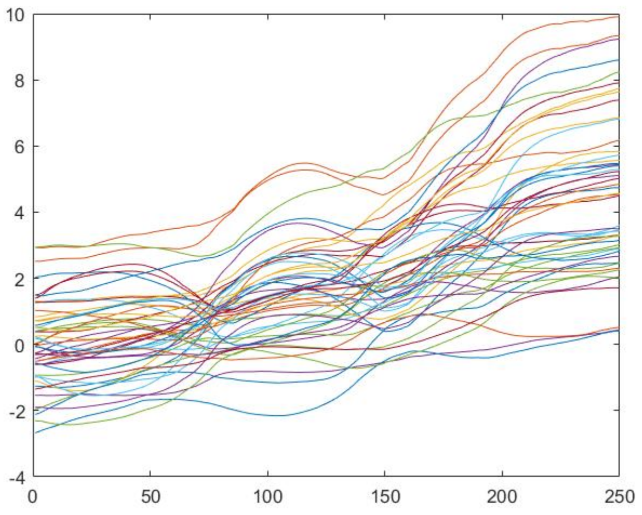

Figure 1: Time-varying factor loadings

2.2 $Tim\_cov.m$

Description: This function compute the proposed estimator for time-varying covariance matrix $\Sigma_{r,t}$ and the sparse estimator for residual covariance estimator Σ_e in section 2 our main article.

Usage:

$$[Sigma\_r, Sigma\_e, Residuals] = Time\_COV(R, PCV, f\_number, tau, h)$$

Input:

(1). R is $N \times T$ excess return matrix, N is dimension, T is sample size

(2). PCV is the sparse penalty coefficience for residual sparse matrix

(3). f_number is the number of factor

(4). tau is positive definite tunning parameter in (B.6)

(5). h is bandwidth parameter, default is rule of thumb $2.35/\sqrt(12)T^{-0.2}N^{-0.1}$

Output:

(1). $Sigma\_r$ is proposed estimated time-varying covariance matrix.

(2). $Sigma\_e$ is sparse residual covariance matrix estimator shown in (3.4) by using .

(3). $Residuals$ is residuals after estimating time-varying factor loading.

2. Example: Based on the setting in example 1, we further provide an example to show how to use Function Tim_cov. In this example, we use cross-validation method proposed in appendix B.1 to select the value for sparse tuning parameter λ.

We show the visualization of residual covariance matrix and its sparse estimator in figure 2. The left plot displays the true residual covariance matrix, and the right plot gives the estimated residual sparse covariance matrix, the degree of color reflects the size of element values. Obviously, we can observe the similar pattern between these two figures.

```
% Continue to the example 1.
tau = 0.001; % set the value for positive definite parameter.

% From example 1, obvious time-varying feature is shown in factor loadings
% hence, we first get the estimated residual filtered by estimated time-varying factor loadings.
[~, ~, Residuals] = Time_COV(Y, 0, K, tau)

% We use cross-validation method proposed in appendix B.1 for sparse tuning parameter
% The folds is set to 3, see more details of Function CV_for_spcov in 'Others' section
lambda = CV_for_spcov(Residuals, 3, tau);

% Estimate covariance matrix and residual covariance matrix.
[Sigma_r, Sigma_e] = Time_COV(Y, lambda, K, tau)

% Visual sparse matrix estimator
map = [1, 1, 0; 0, 0.8, 1; 0, 0.6, 1;
       0, 0.4, 1; 0, 0.2, 1; 0, 0, 1]; % define color map
subplot(1, 2, 1); imagesc(s_cov); colormap(map);
subplot(1, 2, 2); imagesc(Sigma_e); colormap(map); colorbar
```

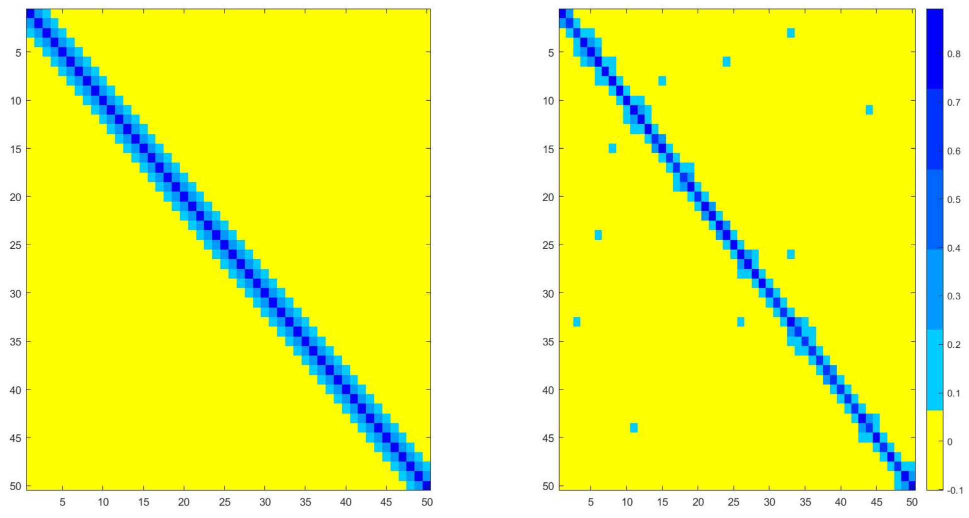

Figure 2: Spare residual covariance matrix(left) and its estimator(right)

2.3 *Rolling_window.m*

Description: Rolling window process described in empirical analysis for daily return data. Using cross-validations for sparse tunning parameter λ and kernel bandwidth in appendix B.1 and B.2, respectively. And using BIC-type information criterion for the number of factors in B.3. The parameters are updated annually.

Usage:

$$[info, Tv\_return, eq\_return] = Rolling\_window(Data\_matrix, history, persist, cur\_pos, end\_pos)$$

Input:

(1). *Data_matrix* is the whole excess return matrix with N rows, N is the number of assets, and the number of columns depends on the length of periods, e.g. length is 10340 for daily data from 01/1980 to 12/2019.

(2). *history* is the length of window.

(3). *persist* is the holding periods after each rebalancing.

(4). *cur_pos* is the position of start time point in *Data_matrix*.

(5). *end_pos* is the position of end time point in *Data_matrix*.

Output:

(1). *info* is a 1×6 row vector containing the Sharpe ratio, mean return and standard deviation of out of sample portfolios from 'TV-MVP' and '1/N', the information from left to right is Sharpe ratio of 'TV-MVP' and '1/N', mean excess return of 'TV-MVP' and '1/N', and standard deviation of 'TV-MVP' and '1/N'.

(2). *Tv_return* contains all out of sample realized excess returns of constructed portfolio based on 'TV-MVP'.

(3). *eq_return* contains all out of sample realized excess returns of constructed portfolio based on '1/N'.

3. Example: In this example, we conduct the rolling window procedure by using provided data $R0$, which is a 50×500 excess return matrix. We set the length of window be 250, the holding period of portfolio is 5, that is we conduct weekly rebalancing. The rolling window procedure continuous to run to the end of the sample. Actually, the data matrix is the largest 50 stocks in S&P index at 01/04/2010, the positions from 251-500 corresponds to nearly one years return data from 01/2010 to 12/2010. We plot the cumulative excess return of two strategies in Figure 3, one can easily observe that 'TV-MVP' strategy obtain a higher cumulative return than '1/N' strategy. And the output $info =$

```
clear; clc;
load('example_data.mat');% load provided data
[info, Tv_return, eq_return] = Rolling_window(R0, 250, 5, 251, 500)
% plot the cumulative excess return of TV-MVP portfolio and '1/N' strategy.
plot(cumsum(Tv_return))
hold on;
plot(cumsum(eq_return),'--')
legend('Tv\_return',' Equal\_return',' Location',' northwest',' NumColumns', 2)
```

2.4 *FWYZtest_step1_bt.m*

Description: Hypothesis test for checking the constancy of factor loading using our bootstrap procedure. Bandwidth used in hypothesis test is set to be rule of thumb $2.35/\sqrt(12)T^{-0.2}N^{-0.1}$, and the loop for bootstrap is 199. The p-value of step 1 test is given by the code $sum(J_pT < J_pT_set)/199$.

Usage:

$$[J\_pT\_set, J\_pT] = FWYZtest\_step1\_bt(R, K, Kernel\_set)$$

Input:

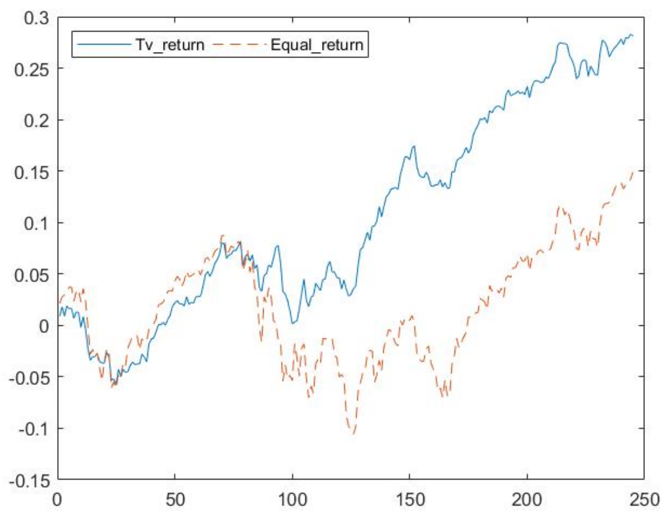

Figure 3: Cumulative excess return

(1). R is $N \times T$ excess return matrix, N is dimension, T is sample size.

(2). K is the number of factors

(3). $Kernel\_set$ is pre-calculating kernel weighting matrix by using default bandwidth.

Output:

(1). J_pT_set is the set contains all bootstrap statistics values J^*_{pT} defined in appendix C.2.

(2). J_pT is statistics values in equation (5.5).

4. Example: Continue with example 2, the step 1 test should reject the constant factor loading null hypothesis since the smooth time-varying factor loading is used. The p_val calculated by this example is 0, that is to say, we have sufficient evidence to reject the constant factor loading null hypothesis.

```
[J_pT_set, J_pT] = FWYZtest_step1_bt(Y, K, Kernel_set);
p_val = sum(J_pT < J_pT_set)/199;
```

2.5 $FWYZtest\_step2.m$

Description: Hypothesis test for checking the constancy of residual covariance matrix in step 2.

Usage:

$$[Ln, ifa] = FWYZtest\_step2(R, alpha)$$

Input:

(1). R is $N \times T$ residual return matrix, N is dimension, T is sample size.

(2). $alpha$ is significance level, e,g, $alpha = 0.05$.

Output:

(1). Ln gives the value of statistic in (5.12).

(2). ifa is a logistic variable, it takes 1 if the result rejects the null hypothesis, and takes 0 if not.

5. Example: Continue with example 4, the step 2 test can not reject the hypothesis that residual covariance matrix is time-invariant. The ifa computed by this example is 0, which gives us the evidence for constant residual covariance matrix.

```
% since we have already rejected the constant factor loading,
% we get use the estimated residual in example 2

% We set the significance level to be 0.05.
[Ln, ifa] = FWYZtest_step2(Residuals, 0.05);
```

2.6 *size_test.m*

Description: This code offers another example for our two-step hypothesis test procedure, it calculates the size for each step test under "Size2" model set-up in appendix C.2.1. Note that in this procedure, for conservative, we give the significance level for each step of the test as half of the total level.

Usage:

$$level = size\_test(d, T, loop, alpha1);$$

Input:

(1). d is the dimension of the data.

(2). T is the sample size of the data.

(3). $loop$ is the replications for computing size.

(4). $alpha1$ is the total significant level, e,g, $alpha1 = 0.05$.

Output:

(1). $level$ is 3-dimensional vector, the first and second element give the sizes of first step test and second step test under $alpha1/2$, respectively. And the third element is the size of the whole testing procedure under $alpha1$.

6. Example: We provide an example to illustrate the usage, we compute the size under the setting 'Size2' in appendix C.2.1. The dimension, sample size and replications are 50, 300 and 500. The output $level = [0.028, 0.043, 0.07]$, which is exactly the same as the results in Table C.5 in appendix.

```
clear; clc;
alpha1 = 0.05; d = 100; T = 300; % set value for significance level, dimension and sample size.
loop = 500;
level = size_test(d, T, loop, alpha1);
```

3 Others

3.1 *Kernel2.m*

Description: Calculate $k^*_{h,tx}$ in section 3.1 of the main paper. In detail, the method of Hong and Li(2005), Su and Wang(2017) to use the boundary kernel to solve it, that is

$$k^*_{h,tx} = h^{-1}K^*_x\left(\frac{t-x}{Th}\right) = \begin{cases} h^{-1}K\left(\frac{t-x}{Th}\right) / \int_{-(x/Th)}^{\infty} K(u)du, & \text{if } x \in [0, \lfloor Th \rfloor) \\ h^{-1}K\left(\frac{t-x}{Th}\right), & \text{if } x \in [\lfloor Th \rfloor, T - \lfloor Th \rfloor] \\ h^{-1}K\left(\frac{t-x}{Th}\right) / \int_{-\infty}^{(1-x/T)/h} K(u)du, & \text{if } x \in (T - \lfloor Th \rfloor, T] \end{cases}$$

and we take Epanechnikov kernel $K(u) = \frac{3}{4}(1-u^2)\mathbb{1}(|u| \le 1)$.

Usage:

$$result = Kernel2(T, t, x, h)$$

Input:

(1). T is sample size.

(2). t is the time point around target.

(3). x is the target point.

(4). h is bandwidth.

Output:

(1). *result* gives the value of $k^*_{h,tx}$.

3.2 *factor_number_selection.m*

Description: Estimate the number of factors by using BIC-type information criterion, that is

$$\widehat{m} = \arg\min_m IC(m)$$

where $IC(m) = \log V(m, \{\check{B}_x(m)\}) + \frac{p+Th}{pTh}\log(\frac{pTh}{p+Th})m$, see more in our appendix B.3. Under some regularity conditions, it is shown that $\widehat{m}$ is consistent to the true number of factors m_0.

Usage:

$$f\_number = factor\_number\_selection(R, max\_number)$$

Input:

(1). R is $p \times T$ excess return matrix.

(2). *max_number* is the pre-set maximum number of factors.

Output:

(1). *f_number* gives the estimated number of common factors.

3.3 $CV\_for\_spcov.m$

Description: Using cross-validation method to choose the value for sparse tuning parameter λ, that is

$$\widehat{\lambda}_{CV} = \arg\max_{\lambda} k^{-1} \sum_{j=1}^{k} \{-\log\det(\widehat{\Sigma}_e(A_j^c)) - \mathrm{tr}(S_{A_j}\widehat{\Sigma}_e^{-1}(A_j^c))\}$$

see more details in our appendix B.1.

Usage:

$$lambda = CV\_for\_spcov(R, k, tau)$$

Input:

(1). R is $p \times T$ residual matrix.

(2). k is the fold of cross-validation.

(3). tau is positive definite transformation tuning parameter.

Output:

(1). $lambda$ gives the selected value of sparse tuning parameter.

3.4 $CV\_for\_h.m$

Description: Using cross-validation method to choose the value for bandwidth h, that is

$$h_{CV} = \arg\max_{h} \sum_{j=2}^{k} \{SR(\widehat{W}(\cup_{i=1}^{j-1} A_i, h), r_t(A_j))\}$$

see more details in our appendix B.2.

Usage:

$$h\_cv = CV\_for\_h(R, lambda, f\_number, tau)$$

Input:

(1). R is $p \times T$ excess return matrix.

(2). $lambda$ is sparse tuning parameter.

(3). K is the number of factors.

(4). tau is positive definite transformation tuning parameter.

Output:

(1). h_cv gives the selected bandwidth.

3.5 $data\_generate$

Description: Generate virtual data by using smooth time-varying factor loadings $b_{it}^{(1)}, b_{it}^{(2)}$ and residuals from multivariate normal distribution $N(0, \Sigma_e)$ where $\Sigma_e = (a_{ij})_{i,j=1,\ldots p}$ and $a_{ij} = 0.5^{|i-j|}$.

$$b_{it}^{(1)} = \rho\left((3 + z_i^{(1)})t/T + z_i^{(2)} sin(4\pi t/T)\right) + z_i^{(3)}$$
$$b_{it}^{(2)} = \rho\left((3 + v_i^{(1)})t/T + v_i^{(2)} sin(4\pi t/T) + (v_i^{(4)} t/T)^2\right) + v_i^{(3)}$$

where $z^{(j)}, j = 1, 2, 3$ and $v^{(j)}, j = 1, 2, 3, 4$ are standard normal variables.

Usage:

$$[Y, r\_cov, s\_cov] = data\_generate(p, T, rho)$$

Input:

(1). p is dimension.

(2). T is sample size.

(3). rho controls the departure from constant factor loading.

Output:

(1). Y is generated virtual data.

(2). r_cov is population return covariance matrix.

(3). s_cov is Σ_e.

3.6 $Markowitz\_MVP$

Description: Obtain the minimum variance portfolio.

Usage:

$$W = Markowitz\_MVP(r\_cov);$$

Input:

(1). r_cov is covariance matrix of excess return.

Output:

(1). W is the optimal weight by solving the minimum variance problem.

4 Example

In this section, we provide a large example for providing a whole picture for the users. We firstly generate the virtual excess return data by using Function $data\_generate$ with dimension $d = 50$, sample size $T = 200$ and $\rho = 1$.

```
clear; clc;
d = 50; T = 200; rho = 1;
h = (2.35/sqrt(12)) * T^(-0.2) * d^(-0.1);

rng(2);
% generate virtual excess return data
[Y, r_cov, s_cov] = data_generate(d, T, rho);
```

With excess return data Y in hand, we will show how to apply our codes to make analysis and get the time-varying minimum variance portfolio step by step. Firstly, we estimate the number of common factors by using suggested BIC, and further test for time varyingness of factor loading and residual covariance matrix as follow.

```
alpha1 = 0.05; % set the significant level
% pre-calculating all kernel weights.
Kernel_set = zeros(T,T);
for r = 1 : T
for t = 1 : T
   k = (h^-1) * Kernel2(T,t,r,h);
   Kernel_set(t,r) = k^0.5;
end
end

% Using BIC-type information criterion to estimate the number of factors
K = factor_number_selection(Y,10);

% Test for the constantcy of factor loadings.
[J_pT_set, J_pT] = FWYZtest_step1_bt(Y,K,Kernel_set);
p_val = sum(J_pT < J_pT_set)/199;
```

After conducting the codes in above box, we can input K and p_val directly in command window, and the results are displayed in the following box. The estimated number of factors is exactly the same as the true number of common factors, and we can reject the hypothesis of the constancy of factor loadings at least under the significant level 0.05.

```
>> K
K = 2
>> p_val
p_val = 0
```

We next test for the constancy of residual covariance matrix. Since rejecting the constancy of factor loadings, we need to use estimated time varying factor loadings to estimate the residuals firstly.

```
% Obtain the estimated residuals by using estimated time varying factor loadings with rule ob
thumb
% bandwidth.
[~,~,Residuals] = Time_COV(Y,0,K,0)
[Ln,ifa] = FWYZtest_step2(Residuals,alpha1);
```

Input ifa into the command window, we can get the results displayed below, $ifa = 0$ means we can not reject the constancy of residual covariance matrix under the significant level 0.05.

```
>> ifa
ifa =
logical
0
```

By now, we get the first sight of the data, the data Y has time varying factor loadings and constant residual covariance matrix(the same as the population settings), therefore, it is suitable to estimate the covariance matrix of excess return by using our time-varying tools proposed in the main text. Next, we start constructing the minimum variance portfolio. In detail, we firstly select the values for sparse tuning parameter by suggested three-fold cross-validation, and then estimate the time-varying covariance excess return matrix, finally, we get the minimum variance portfolio.

```
tau = 0.001;
% using three-fold corss-validation method
lambda = CV_for_spcov(Residuals, 3, tau);
% estimate return covariance matrix at the end point of sample and residual covariance matrix
[Sigma, RES_cov, Residuals] = Time_COV(Y, lambda, K, tau);

% estimate sample covariance matrix of return.
sample_cov = cov(Y');

wopt = Markowitz_MVP(r_cov); % theoretical minimum variance portfolio
wopt_tv = Markowitz_MVP(Sigma); % time-varying minimum variance portfolio
% minimum variance portfolio based on sample covariance
wopt_sam = Markowitz_MVP(sample_cov);

% We plot theoretical optimal weight and estimated weight by 'TV-MVP' of all dimension.
plot(wopt,'xr-'); hold on; plot(wopt_tv,'ob--');
```

The plots of oracle weight and its estimation from 'TV-MVP' are shown in figure 4, it is clear that the weight allocation of 'TV-MVP' is quiet close to theoretical optimal level.

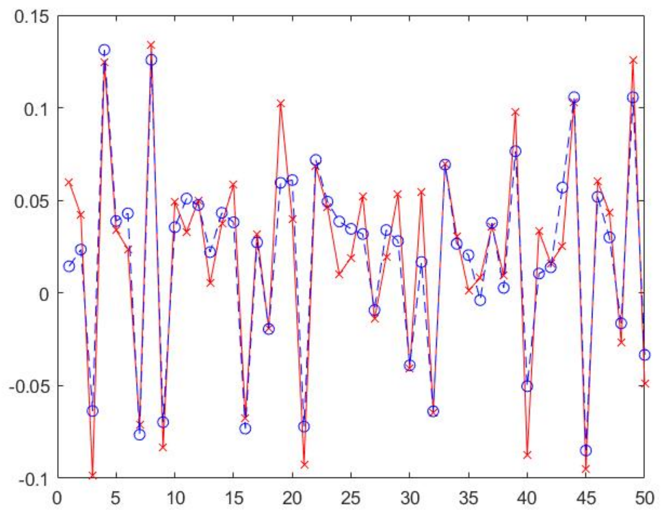

Figure 4: Theoretical optimal weight(red real line with 'x' at the nod) and estimated weight from 'TV-MVP'(blue dot line with 'o' at the nod).

To show the superiority of 'TV-MVP', we further plot the patterns of absolute deviation of weights between theoretical optimal weights and its estimation from 'TV-MVP' and sample covariance-based strategy.

```
% We plot the absolute deviation from theoretical weight.
devia_tv = abs(wopt − wopt_tv);
devia_sam = abs(wopt − wopt_sam);
plot(devia_tv,'r'); hold on; plot(devia_sam,'b − −');
```

The plots are shown in figure 5, it is easy to see that time-varying minimum variance portfolio is closer to theoretical version than sample covariance-based portfolio.

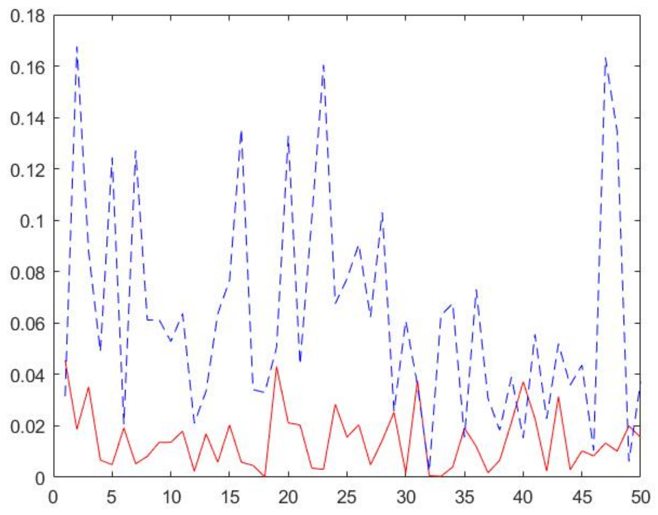

Figure 5: Absolute deviation of each dimension between theoretical minimum variance portfolio and time-varying minimum variance portfolio(red line), theoretical minimum variance portfolio and minimum variance portfolio based on sample covariance(dot line)

Next, we calculate the risk of theoretical portfolio , TV-MVP and minimum variance portfolio based on sample covariance matrix.

```
% Compute risk
risk = wopt' ∗ r_cov ∗ wopt;
risk_tv = wopt_tv' ∗ Sigma ∗ wopt_tv;
risk_sam = wopt_sam' ∗ sample_cov ∗ wopt_sam;
```

Similar with before, one can just input $'risk'$, $'risk\_tv'$ and $'risk_sam'$ into command window to see the final results as below.

```
>> risk
risk = 0.1373
>> risk_tv
risk_tv = 0.1000
>> risk_sam
risk_sam = 0.2117
```

It can be easy concluded that the estimated portfolio variance from TV-MVP is much closer to the theoretical level than that from sample covariance matrix.