Portfolio Optimization

PSTAT 234

UCSB

Ben Ku, Mi Yu, Xining Li, Zhipu Zhou

June 6, 2018

Overview

Introduction to Portfolio Theory

- Invest the allocation of assets that can maximize the expected return and minimize the risks.
- Our objective is to compare the estimations of the three methods, Sample Covariance Method, Graphical-Lasso Method, and Ledoit-Wolf Method.
- Since all three methods give the estimation of the covariance matrix, instead of mean-variance portfolio(MVP), we will choose the minimum-variance portfolio, whose solution is solely based on the estimation of covariance matrix.
- The MVP has the following form

$$\text{minimize}_{w \in \mathbb{R}^p} \quad w^T \Sigma w \quad \text{subject to} \quad \mathbf{1}^T w = 1$$

, where Σ is the true covariance matrix, $\mathbf{1}$ is a vector contains all 1.

Portfolio Rebalancing Scheme

- True covariance matrix Σ is unobservable and needs to be estimated (denoted as $\hat{\Sigma}$) from historical data.
- Because returns are not stationary over time, the estimated covariance of returns must be periodically updated. A general way is to periodically re-calculate $\hat{\Sigma}$ and rebalance the portfolio weight.

Minimum Variance Portfolio Selection at Each Period

- At each rebalancing period T_k for $k = 1, 2, \cdots$, the portfolio weights can be calculated from the L days(estimation horizon) before the beginning of the rebalancing period T_k.
- The portfolio weights of T_k can be estimated using the following form

  $$w_k = (\mathbf{1}^T \hat{\Sigma}_k^{-1} \mathbf{1})^{-1} \hat{\Sigma}_k^{-1} \mathbf{1}$$

  , where $\mathbf{1}$ is a vector contains only 1.
- The estimated covariance matrix is estimated using sample covariance matrix estimator, Graphical Lasso estimator, and Ledoit-Wolf shrinkage estimator.

Sample Covariance Matrix Estimator

- A standard way to estimate the true covariance matrix, the estimation have the following mathematical form

$$S = \frac{1}{L-1} \sum_{t=T_{k-1}+1-L}^{T_{k-1}} (r_t - \bar{r}_{k-1})(r_t - \bar{r}_{k-1})^T$$

, where $\bar{r}_{k-1} = \left(\frac{1}{L} \sum_{t=T_{k-1}+1-L}^{T_{k-1}} r_{t1}, \cdots, \frac{1}{L} \sum_{t=T_{k-1}+1-L}^{T_{k-1}} r_{tp} \right)$.

- This method can be ill-conditioned because when the observations L less than the dimension p, the sample covariance matrix will be singular, and $\hat{\Sigma}^{-1}$ is not defined. The weights then cannot be calculated.
- Sample covariance matrix sometimes contains large estimation error that could distort the portfolio optimization

Ledoit-Wolf Method

- Sample Variance is usual but know to perform poorly when the dimension of the data is high
- This method has the following form

  $$\Sigma_{lw} = \lambda S + (1 - \lambda)\mathbf{I}$$

  where S is just the sample covariance matrix and $\mathbf{I}$ is the identity matrix.
- The shrinkage parameter $\lambda \in [0, 1)$ needs to be determined though cross-validation. We do not consider $\lambda = 1$ because that case will be the same as sample covariance matrix estimator.

Graphical Lasso Method

- Graphical Lasso is an acronym for Graphical Least Absolute Shrinkage and Selection Operator, and it is a penalized regression analysis method that performs both variable selection and shrinkage in order to enhance the prediction accuracy.
- It focuses on estimating inverse covariance matrix Ω(the precision matrix), and it has the form

$$\text{argmin}_{\Omega \in \mathbb{R}^{p \times p}} - \log \det \Omega + \text{tr}(S\Omega) + \lambda \|\Omega\|_1 .$$

- It is better than Ledoit-Wolf Method when studying stocks because it used to find a sparse inverse covariance matrix, and it is very easy to see if there is a correlation between stocks.

K-folds Cross-Validation

- The tuning parameters λ for Ledoit-Wolf method and Graphical Lasso method need to be determined by cross-validation.
- Splitting the data into k folds first. Find the λ that makes the predictive risk the smallest. The predictive risk is defined as

$$PR(\lambda) = \frac{1}{k}\sum_{m=1}^{k}\left(\frac{1}{N_m}||R^m\hat{\Omega}(\lambda)\hat{\Omega}(\lambda)_D^{-1}||_F\right)$$

, where N_m represents the number of observations in m-th fold, R^m is the matrix that contains the observations in the m-th fold, $\hat{\Omega}$ is the inverse of the estimated covariance from the tuning parameter λ, and $\hat{\Omega}_D$ is the diagonal matrix with its diagonal equals to the diagonal of $\hat{\Omega}$.

Performance Metrics

The performance of the portfolios can be measured through different performance metrics.

- **Realized return**: the annualized average return of the portfolio over the entire investment horizon.
- **Realized risk**: the standard deviation of portfolio over the entire investment horizon.
- **Sharpe ratio**: the realized excess return of portfolio over the annualized risk-free interest rate r_f per unit realized risk.
- **Turnover**: the amount of new portfolio assets purchased or sold over k-th period.
- **Size of short side**: the proportion of negative weights to the sum of absolute weights of portfolio allocation at k-th period.
- **Normalized wealth**: accumulated wealth of portfolio with initial budget is 1 dollar. Transaction costs and borrowing costs are taken into consideration.

Data Description and Preprocessing

- Data source: Historical closing prices of Dow Jones Industrial Average 30 component stocks, downloaded from IEX.
- Time horizon: May/22/2013 - May/18/2018
- Rebalancing periods L: 250, 200, 150, 100

Tables

Table: Annualized Realized Return(%)

	sample	lw	gl
250	5.93	6.71	7.07
200	8.47	7.73	7.76
150	5.32	7.33	7.6
100	7.22	8.9	9.3

Table: Annualized realized risk(%)

	sample	lw	gl
250	11.98	11.2	11.35
200	11.73	10.97	11.08
150	12.03	11.14	11.4
100	12.76	11.04	11.19

Table: Realized Sharpe Ratio

	sample	lw	gl
250	0.328	0.420	0.447
200	0.552	0.522	0.520
150	0.276	0.479	0.492
100	0.409	0.625	0.652

Tables

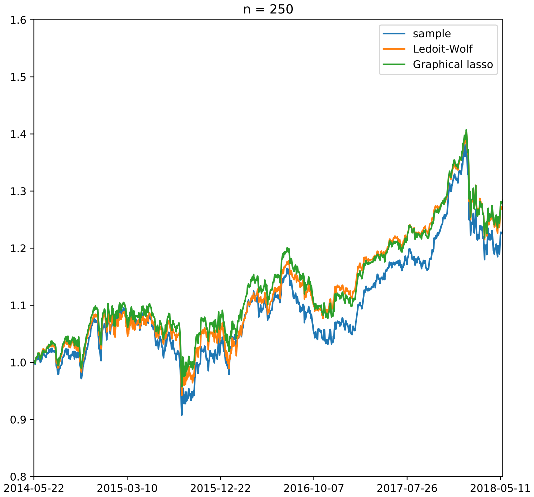
Table: Turnover

	sample	lw	gl
250	2.372	0.985	1.393
200	2.344	0.941	1.436
150	2.503	0.868	1.433
100	3.137	0.872	1.393

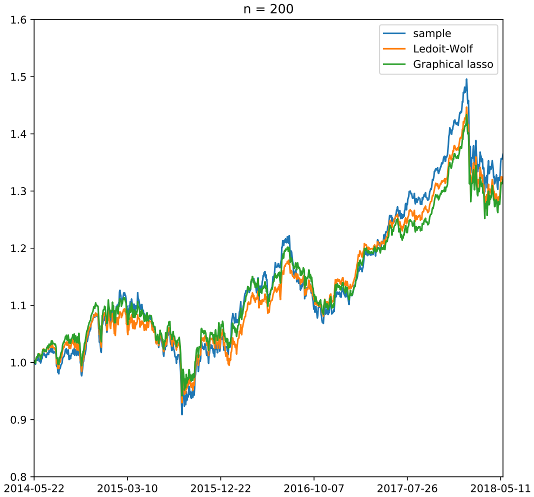
Table: Size of Short Size(%)

	sample	lw	gl
250	23.88	4.54	12.6
200	24.63	5.88	14.51
150	24.56	3.55	13.02
100	29.59	4.98	13.31

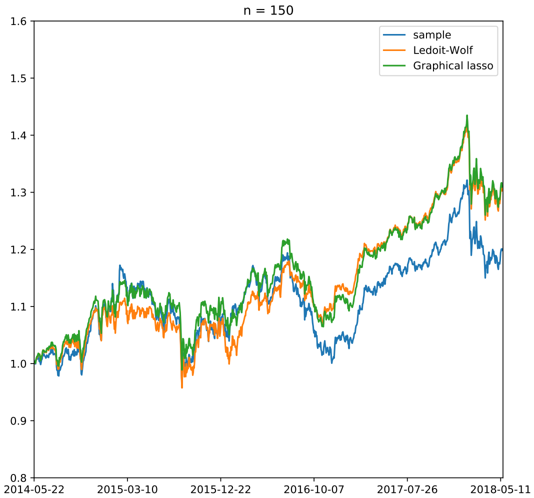
Portfolio Wealth Growth for 250 Days Period



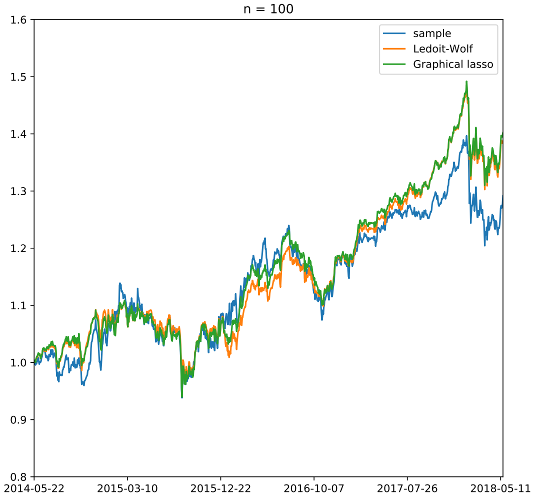
Portfolio Wealth Growth for 200 Days Period



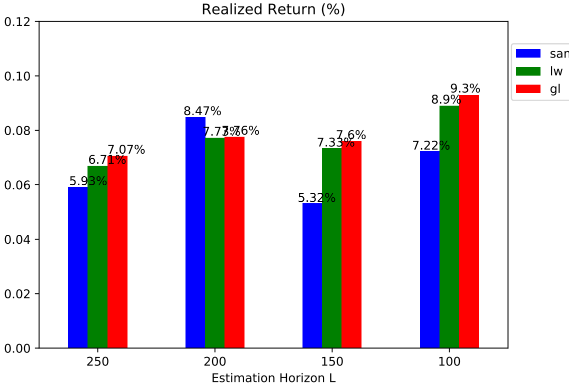
Portfolio Wealth Growth for 150 Days Period



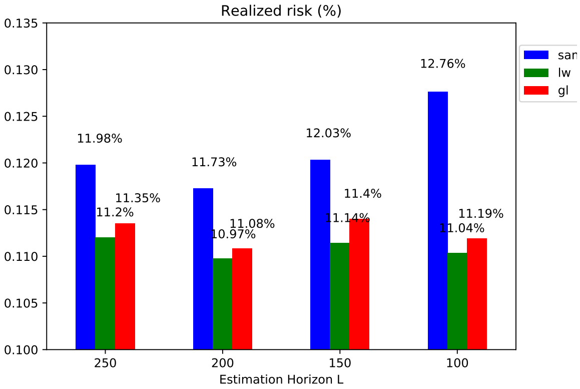
Portfolio Wealth Growth for 100 Days Period



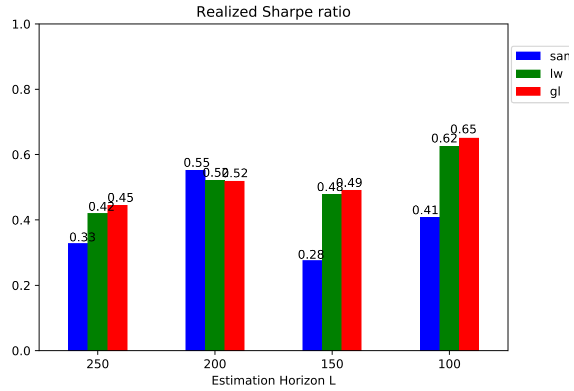
Realized Return



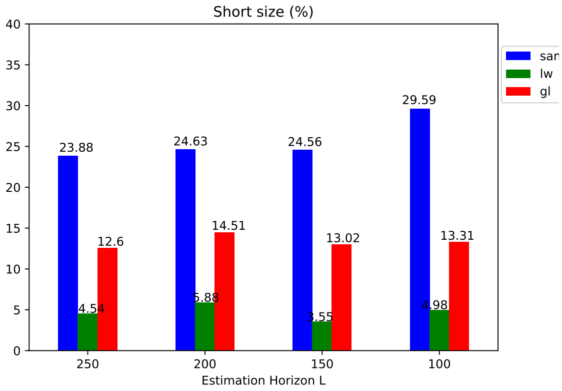
Realized Risk



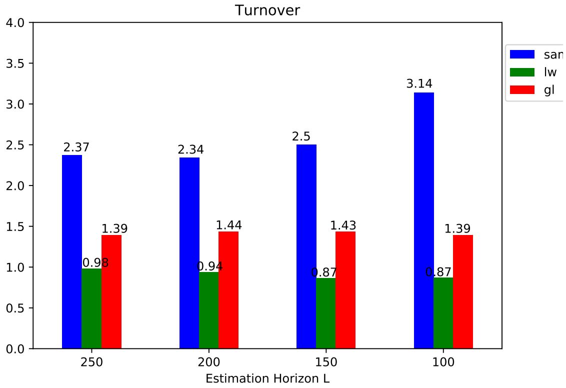
Realized Sharpe Ratio



Short Size


Short size (%)
40
35
30
25
20
15
10
5
0
san
lw
gl
23.88
4.54
12.6
24.63
5.88
14.51
24.56
3.55
13.02
29.59
4.98
13.31
250
200
150
100
Estimation Horizon L

Turnover


Turnover
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2.37
0.98
1.39
2.34
0.94
1.44
2.5
0.87
1.43
3.14
0.87
1.39
250
200
150
100
Estimation Horizon L
san
lw
gl

Conclusions

- The regularization methods produce better estimations for (inverse) covariance matrix than sample covariance estimation. Thus it effectively reduced realized risk of the portfolio.
- Ledoit-Wolf estimation performs better than Graphical Lasso in terms of realized risk.

Appendix: Optimization_At_Each_Period

This is the code for optimization for each investment period:

```
def Optimization_At_Each_Period(self):
    w=[]
    for current_period_num in range(int(np.floor(
        (len(self.stock_data)/self.days_of_period) ))):
        if current_period_num==0:
            stock_data_partition =
                self.stock_data[current_period_num*
            self.days_of_period:(current_period_num+1)*
            self.days_of_period].copy()
        else:
            stock_data_partition =
                self.stock_data[(current_period_num*
            self.days_of_period-1):
            (current_period_num+1)*
            self.days_of_period].copy()
            w.append(self.optimal_solution(stock_data_partition))
    return np.column_stack(w)
```

Appendix: Realized return

The average return of the portfolio over the entire investment horizon.

$$r_p = \frac{1}{T} \sum_{t=1}^{T} r_t' w_{k_t}$$

In the Portfolio Optimization class, we split data in chunks. We put the return over time of different investment periods which in this case is the annualized average return of the portfolio over the entire investment horizon $[T_0, T_E]$: into a list with .

$$r_p = \frac{252}{T_E - T_0} \sum_{t=T_0+1}^{T_E} r_t^T w_{k_t}$$

Appendix: Return Over Time

```
def return_over_time(self):
    re=[]
    for current_period_num in range(int(np.floor(
        (len(self.stock_data)/self.days_of_period) ))):
        after =
            self.stock_data[(self.days_of_period*current_period_num+
        before =
            self.stock_data[(self.days_of_period*current_period_num)
        tmp = np.log(np.array(after)/np.array(before))
        tmp = pd.DataFrame(tmp,columns = after.columns)
        re.append(tmp)
```

Appendix: Return Over Time

```
if len(self.stock_data)/self.days_of_period-
          int(len(self.stock_data)/self.days_of_period)==0:
    return re # Which Means The Last Part of Investment Time
        is a whole Period
else:
    # print ("Last Trading Period Not Finished")
    last_start=int(np.floor(
        (len(self.stock_data)/self.days_of_period)
        ))*self.days_of_period
    last_end=len(self.stock_data)-1
    before = self.stock_data[(last_start-1):(last_end-1)]
    after = self.stock_data[last_start:last_end]
    tmp = np.log(np.array(after)/np.array(before))
    tmp = pd.DataFrame(tmp,columns = after.columns)
    re.append(tmp)
    return re
```

Appendix: Turnover

the amount of new portfolio assets purchased or sold over k-th period:

$$TO(k) = \left\| w_k - w_{k-1} \cdot \bigodot_{t=T_{k-1}+1}^{T_k} (1 + r_t) \right\|_1$$

where

$$\bigodot_{t=T_{k-1}+1}^{T_k} (1 + r_t) = (1 + r_{T_{k-1}+1}) \cdot (1 + r_{T_{k-1}+2}) \cdot \cdots \cdot (1 + r_{T_k})$$

and $\cdot$ is dot product, $w_0 = (0, 0, \cdots, 0)$. The average turnover over the entire investment horizon is:

$$\overline{TO} = \frac{1}{\mathcal{T}} \sum_{k=1}^{\mathcal{T}} TO(k)$$

Appendix: Turnover Code

```
def turnover(self):
    interval = self.generate_interval() # Interval
    Mat_return_over_time = self.return_over_time() # dataframe
        by chunks results
    TO = np.empty(interval.shape[0])
    for i in range(interval.shape[0]):
        if(i == 0):
            wold = np.zeros(self.stock_data.shape[1])
        else:
            wold = np.array([j[0] for j in
                np.asarray(self.Mat_Optimization_At_Each_Period[:,i-1
        wnew = np.array([j[0] for j in
            np.asarray(self.Mat_Optimization_At_Each_Period[:,i])])
        TO[i] = np.sum(np.abs(wnew -
            np.multiply.accumulate(Mat_return_over_time[0] + 1,
            axis = 0).iloc[-1] * wold))
    self.TO = TO
    return TO
```

Appendix: Realized risk

The standard deviation of portfolio over the entire investment horizon:

$$\sigma_p = \sqrt{\frac{252}{T_E - T_0} \sum_{t=T_0+1}^{T_E} (r_t^T w_{k_t} - r_p)^2}$$

Appendix: Realized risk Code

```
def Realized_Return(self):
    interval = self.generate_interval()
    data =
        self.stock_data[interval.loc[0,'es']:interval.loc[interval.s
    Mat_return_over_time = self.return_over_time() # return
        dataframe by chunks
    Total_time = self.Mat_Optimization_At_Each_Period.shape[1] #
        Number of periods
    ss = self.shortsize() # Size of short side
    daily_r = pd.DataFrame()
    for t in range(Total_time):
        tmp = pd.DataFrame(np.matmul(Mat_return_over_time[t +
            1], self.Mat_Optimization_At_Each_Period[:,t]))
        daily_r = pd.concat([daily_r, tmp])
        self.daily_r = daily_r
    index_num = daily_r.shape[0]
    daily_r.index = list(self.stock_data.index)[-index_num:]
    return daily_r
```

Appendix: Portfolio Wealth

Accumulated wealth derived from the portfolio over the trading period when the initial budget is normalized to one. Note that both transaction costs and borrowing costs are taken into account. Let $W(t-1)$ denote the wealth of the portfolio after the $(t-1)$-th trading day. Then, the wealth of the portfolio after the t-th trading day is given by

$$W(t) = \begin{cases} W(t-1)(1 + r_t' w_{k_t} - TC(k_t) - BC(k_t)) & t = T_{k_t} + 1 \\ W(t-1)(1 + r_t' w_{k_t}) & t \neq T_{k_t} + 1 \end{cases}$$

Denote the transaction cost by r_c then $TC(k) = r_c TO(k)$.
$BC(k) = ((1 + r_b)^{L_{k-1}}) Upper$ where $Upper = \sum_{i=1}^{p} |\min(w_{i(k-1)}, 0)|$

Appendix: Portfolio Wealth Code

```
def Port_Wealth(self,trans_cost = 0, borr_cost = 0,init = 1):
    daily_r = self.Realized_Return()
    TO = self.turnover()
    ss, upper = self.shortsize()
    interval = self.generate_interval()
    ptf_r = daily_r.copy()
    invs = interval["is"] - interval['is'][0]
    n = interval['ee'][0] - interval['es'][0] + 1
    for k in range(interval.shape[0]):
        ptf_r.iloc[invs[k]] = ptf_r.iloc[invs[k]] - TO[k] *
            trans_cost - upper[k] * (math.pow(1 + borr_cost, n) -
            1)
    ptf_w = init * np.multiply.accumulate(ptf_r + 1)
    index_num = ptf_w.shape[0]
    ptf_w.index = list(self.stock_data.index)[-index_num:]
    return ptf_w
```

Appendix: Data Extraction

```
from datetime import datetime
from concurrent import futures
import pandas as pd
from pandas import DataFrame
import pandas_datareader.data as web
def download_stock(stock):
   """ try to query the iex for a stock, if failed note with
       print """
   try:
      print(stock)
      stock_df = web.DataReader(stock,'iex', start_time,
          now_time)
      stock_df['Name'] = stock
      output_name = stock + '_data.csv'
      stock_df.to_csv(output_name)
   except:
      bad_names.append(stock)
      print('bad: %s' % (stock))
```

```
if __name__ == '__main__':

  """ set the download window """
  now_time = datetime.now()
  start_time = datetime(now_time.year - 5, now_time.month ,
     now_time.day)

  """ list of s_anp_p companies """
  s_and_p = ticket

  bad_names =[] #to keep track of failed queries

  """here we use the concurrent.futures module's
     ThreadPoolExecutor
   to speed up the downloads buy doing them in parallel
   as opposed to sequentially """

  #set the maximum thread number
  max_workers = 50
```

```
workers = min(max_workers, len(s_and_p)) #in case a smaller
    number of stocks than threads was passed in
with futures.ThreadPoolExecutor(workers) as executor:
   res = executor.map(download_stock, s_and_p)
""" Save failed queries to a text file to retry """
if len(bad_names) > 0:
   with open('failed_queries.txt','w') as outfile:
      for name in bad_names:
         outfile.write(name+'\n')
finish_time = datetime.now()
duration = finish_time - now_time
minutes, seconds = divmod(duration.seconds, 60)
print('getSandP_threaded.py')
print(f'The threaded script took {minutes} minutes and
    {seconds} seconds to run.')
```

Appendix: Graphical Lasso Cross Validation

Here is the code for minimizing Predicitve Risk using Graphical Lasso and K-Fold Cross Validation

```
def graphlasso_cv(self, data,alpha_num=10,n_fold=5):
    datac = pd.DataFrame(scale(data, axis = 0, with_mean = True,
        with_std = True, copy = True))
    sdv = data.std(axis = 0)
    alpha = np.arange(0, 0.5, 0.05)

    default_fold ={'fold1':np.zeros(len(alpha))}
    for ticker in range(n_fold):
        default_fold.update({str('fold'+str(ticker+1)):
            np.zeros(len(alpha))})

    pr_all =pd.DataFrame(default_fold)
    pr_all.index = alpha
```

Appendix: Graphical Lasso Cross Validation

```
#split into n-fold
kf = KFold(n_splits = n_fold)
#initialize the predictive risk
ii = -1
for train, test in kf.split(datac):
    ii = ii + 1
    x_train = datac.iloc[train]
    x_test = datac.iloc[test]
    ##next get the Graph Lasso covariance matrix using
        the shriankgage
    model = GraphLasso(l)
    #print(x_train.shape)
    model.fit(x_train)
    omega = model.precision_
    ##calculate the predictive risk
    omega_diag = np.diag(np.diag(omega))
    omega_diag_inv = np.diag(1/np.diag(omega))
```

Appendix: Graphical Lasso Cross Validation

```
        #This calculates the quantity inside the norm
        z = np.matmul(omega,omega_diag_inv)
        m = np.matmul(x_test,z)
        #calculate the predictive risk
        pr = (np.linalg.norm(m,2))**2/x_test.shape[0]
        pr_all.loc[l][ii] = pr
#This gives the best alpha value after the cross-validation
best_alpha = np.argmin(pr_all.mean(axis = 1))
model = GraphLasso(best_alpha)
model.fit(datac)
omega = np.matmul(np.matmul(np.diag(sdv), model.precision_),
    np.diag(sdv))
return omega, best_alpha
```

Appendix: Graphical Lasso Cross Validation

We set optimal to be a tuple that contains both the precision matrix and the optimal value of alpha for minimizing the predictive risk according to Graphical Lasso.

```
optimal = graphlasso_cv(stock_return[0])
optimal[0].shape #shows that it is a 30x30 matrix
optimal[0]
optimal[1] #optimal alpha value
```

Appendix: Class Implementation

Here is the initialization of the Portfolio Optimization Class

```
def __init__(self, stock_dat_address,selected_companies_names
    ,days_of_period,start, covariance_method,
    **method_parameters):
   self.selected_companies_names = selected_companies_names
   self.stock_data =
       pd.read_pickle(stock_dat_address).dropna(axis =
       1)[selected_companies_names]
   self.days_of_period =days_of_period
   self.covariance_method = covariance_method
   self.start = start
   if covariance_method!='sample':
       for key, value in method_parameters.items():
           setattr(self, key, value)
   self.Mat_Optimization_At_Each_Period =
       self.Optimization_At_Each_Period()
```

Appendix: Class Implementation

The essence of class is actually an approximation of the logical relationship of the reality. The world is composed of different objects and the relationship between them continues to occur. Class describes the object and its relationship in a nature way.
The reason of using a class in this case is because the scope of investment (i.e. the companies) is unchanged, the starting data of the investment is unchanged, and some function should only be called when necessary.
The method_parameters are keyword arguments. The reason of using this is because different estimation of inverse covariance matrix may acquire different parameters. The keyword arguments can help us to pass in necessary arguments without including unnecessary arguments.
Class also reduce the complexity of code. We can generate table of performance measurements based on inverse covariance matrix estimation and number of days in a period in only several lines of code.