Week 2 Project - Financial Optimization Computational Project

---

Submitted to:
Dr. Roy Kwon

Created by:
Fendi Tjoa – 1007535017
Yue Zhang – 1007580823

APS 1022
University of Toronto
Toronto, ON
6/11/2021

Part 1. Introduction Financial Optimization Models

Portfolio optimization models are different type of models that can determine the weights of assets of portfolios depending on user's preferences. There are three different financial portfolio models that will be discussed in this report which consist of Mean-Variance Optimization Model (MVO), Robust Mean-Variance Optimization Model, and Risk Parity Optimization Model. The last model that will also be discussed is the market capitalization model, which is calculated based on market capitalizations. Based on historical data in the form of estimated return and variance of each asset, these models will give different result on assigning weights of assets of portfolios.

Part 2. Setup

The 20 stocks that formed the stock universe and analyzed are 'F', 'CAT', 'DIS', 'MCD', 'KO', 'PEP', 'WMT', 'C', 'WFC', 'JPM', 'AAPL', 'IBM', 'PFE', 'JNJ', 'XOM', 'MRO', 'ED', 'T', 'VZ', 'NEM'. Start and end time of this period is from 30 Dec 2004 to 30 Sep 2008. Historical stock prices were gathered from yahoo finance using python library called **pandas_datareader**. Covariance matrix Q was calculated using monthly returns calculated from these stock prices.

Part 3a. Mean-Variance Optimization Model (MVO)

Mean-Variance Optimization Model determines weights of a portfolio based on estimated expected return μ, covariance matrix of assets Q, and risk aversion λ. The model tries to maximize portfolio return while minimizing risk according to the risk aversion coefficient. The optimization problem is below.

$$\boldsymbol{minimize}\ \lambda \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} - \mu^T \boldsymbol{x}$$

$$\boldsymbol{subject\ to}$$

$$e^T x = 1$$

Where $e \in R^n$ is a vector of n ones, and short selling is allowed. This objective function can also be written in maximizing form by multiplying its objective function by negative 1. As a result, MVO objective function becomes:

$$\boldsymbol{maximize}\ \ \mu^T \boldsymbol{x} - \ \lambda \boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x}$$

Since this objective function is a convex function, a python library called CVXPY can be used to find the best weights for MVO portfolio. Writing the optimization problems using CVXPY follows a certain format that optimization variables (20 asset means 20 variables total), cost function, constraints, and either maximize or minimize need to be declared (details can be observed on python notebook cell 8).

Part 3b. Robust Mean-Variance Optimization Model

Mean-Variance Optimization Model determines weights of a portfolio based solely based on estimated parameters such as expected returns and variances. It ignores estimation errors that can affect the stability of a portfolio and is very sensitive to the estimated parameters. The Robust MVO model was developed to incorporate estimating errors and uncertainties into the optimization model. On top of the MVO model, an ellipsoidal uncertainty set was introduced to the objective function. As a result, portfolio weights will be less sensitive to changes of estimated parameters. The modified MVO model becomes:

$$\boldsymbol{minimize}\ \lambda \boldsymbol{x}^T \boldsymbol{Q}\boldsymbol{x} - \mu^T \boldsymbol{x} + \varepsilon_2 \sqrt{\boldsymbol{x}^T \Theta \boldsymbol{x}}$$

$$\boldsymbol{subject\ to}$$

$$e^T \boldsymbol{x} = 1$$

where $\Theta = \frac{1}{\mathrm{T}} diag(diag(Q))$, ε_2 denotes confidence level, and assuming short selling is allowed.

This objective function can also be written in maximizing form by multiplying its objective function by negative 1. As a result, Robust MVO objective function becomes:

$$\boldsymbol{maximize}\text{:}\ \mu^T \boldsymbol{x} - \lambda \boldsymbol{x}^T \boldsymbol{Q}\boldsymbol{x} - \varepsilon_2 \sqrt{\boldsymbol{x}^T \Theta \boldsymbol{x}}$$

Formulating this problem in python follows similar python implementation as MVO model on part 3a with the difference in the objective function. Details of Robust MVO python code can be observed on python notebook cell 10).

Part 3c. Risk Parity Optimization Model / Equal Risk Contribution Model (ERC)

Risk Parity Optimization Model or Equal Risk Contribution Model (ERC) determines weights of a portfolio that each asset contributes equal risk. This portfolio compromises MVO technique and equal weighting approach. The advantages of this model are the portfolio will be fully diversified from a risk perspective and does not require estimated expected returns. The original objective function of risk parity is:

$$\boldsymbol{minimize}: \sum_{i=1}^{n}\sum_{j=1}^{n}\left(x_i(\boldsymbol{Qx})_i - x_j(\boldsymbol{Qx})_j\right)^2$$

$$\boldsymbol{subject\ to}$$

$$e^T\boldsymbol{x} = 1$$

$$\boldsymbol{x} \geq 0$$

Short selling is not allowed for this model to ensure unique solution. There is an issue regarding this objective function having sum of n^2 elements. The objective function can be reduced to n elements by replacing second risk contribution term with a dummy variable θ and becomes:

$$\boldsymbol{minimize}: \sum_{i=1}^{n}(x_i(\boldsymbol{Qx})_i - \theta)^2$$

Where $\theta \epsilon R$ is an auxiliary unconstrained variable and at optimality, $(x_i(\boldsymbol{Qx})_i = \theta)$ for all i. The least square function is more challenging to be formulated and therefore the team utilizes both MATLAB and Python to create ERC model. Covariance matrix from python file is imported to MATLAB to build ERC cost function. MATLAB function **optimproblem** is used to create an objective problem that includes objective function and constraints. Initial points of every single asset x and theta are declared as 0. Given the input of objective problem and initial points, **solve** function will automatically choose **fmincon** as a nonlinear programming solver to find the optimal weights for ERC. Details of ERC MATLAB code can be observed on MATLAB code appendix.

The other approach is to use python to optimize ERC model. Utilizing **scipy.optimize.least_squares** function in python. However, this function only accepts objective function and bounds on variables. Hence the results must be normalized so they sum up to 1. Details of ERC Model python code can be observed on python notebook cell 11-12).

Note: Both approaches yield the same weights and risk contributions.

Part 3d. Market Capitalization Weights Model

Weights of the fourth portfolio are based on market capitalization on October 2008 and November 2008. These data were gathered from Wharton Research Data Service https://wrds-www.wharton.upenn.edu/. Weight of each asset will be the market capitalization of that asset divided by the sum of market capitalization of all 20 assets. Detail of Market Capitalization Weights Model python code can be observed on python notebook cell 14-16).

Moreover, the 10-year U.S. treasury yield was used as a good approximation for the risk-free rate. This risk-free interest rate is chosen because according to forbes.com, 10-year U.S treasury yield serves as a vital economic benchmark. It can also influence other interest rates such and has an impact on economic growth and the economy[1]. According to U.S. department of treasury, the average 10-year treasury yield for October and November 2008 are 3.8072% and 3.521%, respectively. These values will be used as risk free rates for calculations regarding corresponding months later.

Part 4a. Result on Oct 2008 Data

Once obtained, the Oct 2008 market portfolio (the fourth portfolio), the risk aversion value λ can be calculated as:

$$\lambda = \frac{E[r_{mkt}] - r_f}{\sigma^2_{\mathrm{mkt}}}$$

Where $E[r_{mkt}]$ is expected return of the market portfolio, r_f is risk-free rate and $r_f = 3.8072\%$ for Oct 2008, σ^2_{mkt} is variance of the market portfolio. For Oct 2008, calculated risk aversion coefficient $\lambda_{oct} = 3.82$. Using this risk aversion coefficient as one of the input parameters, along with estimated returns and variances, MVO and robust MVO model could generate portfolio weights for Oct 2008. Portfolio weights generated for the four portfolios are shown on table 1.

Table 1: Weights of Different Portfolios (Oct 2008)

October 2008					
Company	**MVO Weights**	**Robust MVO Weights (95%)**	**Robust MVO Weights (90%)**	**ERC Weights**	**Market Cap Weights**
F	-0.170128708	-0.071273884	-0.171286186	0.016483434	0.002320393
CAT	-0.022832493	0.016913144	0.033956242	0.032428923	0.011146119
DIS	0.578532735	0.120899599	0.041265979	0.066086335	0.023516525
MCD	0.980631156	0.082690194	0.28005966	0.028920912	0.031514197
KO	-0.800065174	0.030673551	0.111469789	0.056319493	0.049305894
PEP	0.597783892	0.187780791	0.201227526	0.092790407	0.042828302
WMT	0.10595972	0.233571441	0.191539716	0.144747935	0.106198467
C	-1.167513614	-0.136510295	-0.275892571	0.02506937	0.035980221
WFC	1.295100013	0.182371754	0.218785717	0.051896133	0.054766521
JPM	0.394162499	0.040773816	0.153099929	0.029003629	0.07436117
AAPL	0.028927922	-0.011829487	0.087710894	0.01541729	0.046261151
IBM	-0.64160798	-0.026560837	-9.14499E-05	0.03697039	0.060414536
PFE	-1.002512132	-0.089186954	-0.171418717	0.040234404	0.05774535
JNJ	-0.068521798	0.078233312	0.05031258	0.082093175	0.082912839
XOM	0.771529019	0.131787607	0.163299819	0.035847634	0.182377672
MRO	0.299173553	0.071903348	0.149858574	0.032689262	0.009931067
ED	-0.427365519	0.062941293	0.043052699	0.061740874	0.005724392
T	0.546017355	0.051846585	0.038295071	0.033857129	0.076306113
VZ	-0.270793138	-0.0308717	-0.109394205	0.034393286	0.040764715
NEM	-0.026477307	0.073846719	-0.035851065	0.083009985	0.005624356

With the portfolio weights above, and using realized returns of Oct 2008, the portfolio return, variance, and Sharpe ratio calculated for the four strategies are shown on table 2.

According to the result of different strategies, surprisingly the MVO portfolio was the highest for both return and Sharpe ratio. While the ERC portfolio lost the most and had the lowest Sharpe ratio. Reason for this was that all the optimization models were developed for normal market times and all parameters used

[1] https://www.forbes.com/advisor/investing/10-year-treasury-yield/

were estimated based on non-crisis market data. But October 2008 was one of the crisis times that 19 out of the 20 stocks went down during this time. Since only the MVO and robust MVO allowed short selling, they were more likely to have better returns than ERC and the market portfolio in a declining market. Moreover, the 90% robust MVO had positive return during the crisis with much lower variance than MVO. As for the ERC model, although it had the lowest variance, it still lost the most during the crisis.

Table 2: Return, Variance, Standard Deviation, and Sharpe Ratio of Different Portfolios (Oct 2008)

Portfilio	return (monthly)	variance (monthly)	std (monthly)	Sharpe ratio
MVO	0.301900	0.009725	0.098614	3.029251
Robust MVO 95%	-0.003076	0.001435	0.037888	-0.164914
Robust MVO 90%	0.010489	0.001624	0.040295	0.181580
ERC	-0.145946	0.000474	0.021772	-6.848979
Market portfolio	-0.103726	0.000840	0.028974	-3.689422

Part 4b. Result on Nov 2008 Data

Table 3: Weights of Different Portfolios (Nov 2008)

November 2008					
Company	MVO Weights	Robust MVO Weights (95%)	Robust MVO Weights (90%)	ERC Weights	Market Cap Weights
F	-0.165529142	-0.147440909	-0.158103784	0.016483434	0.003111545
CAT	-0.022301827	0.032984467	0.032832075	0.032428923	0.012338818
DIS	0.564167905	0.048560034	0.049827473	0.066086335	0.020801323
MCD	0.947658509	0.237098228	0.259937623	0.028920912	0.032674977
KO	-0.773044243	0.107571948	0.101203485	0.056319493	0.05411088
PEP	0.586250276	0.18837919	0.196681687	0.092790407	0.043943856
WMT	0.111430902	0.185051246	0.197767229	0.144747935	0.1096978
C	-1.130758298	-0.232183164	-0.262723256	0.02506937	0.022543648
WFC	1.259936851	0.191654155	0.216662395	0.051896133	0.05375477
JPM	0.377527494	0.129858601	0.145773885	0.029003629	0.058966319
AAPL	0.027099634	0.070606735	0.072878243	0.01541729	0.041107303
IBM	-0.621991076	0.006035349	-0.004040986	0.03697039	0.054704628
PFE	-0.971464467	-0.138967013	-0.162582989	0.040234404	0.055283604
JNJ	-0.064772648	0.063154027	0.052738293	0.082093175	0.081106293
XOM	0.750198787	0.1444613	0.160172474	0.035847634	0.203444106
MRO	0.290370963	0.129906871	0.141822702	0.032689262	0.009217715
ED	-0.411931809	0.051804083	0.045054925	0.061740874	0.005515024
T	0.530466372	0.036104143	0.03953955	0.033857129	0.083985528
VZ	-0.261976914	-0.087411746	-0.101472919	0.034393286	0.046279147
NEM	-0.02133727	-0.017227546	-0.023968104	0.083009985	0.007412716

Similar to what was done for October 2008, the risk aversion coefficient for November 2008 was calculated as 4.493. And the portfolio weights generated for November 2008 are shown on table 3.

With the portfolio weights above, and using realized returns of Nov 2008, the portfolio return, variance and Sharpe ratio calculated for the four strategies are shown on table 4.

The results are very similar to October 2008. The MVO model was the highest and only positive for both return and Sharpe ratio, while its variance was the highest as well. The robust MVO got the second highest Sharpe ratio and the ERC got similar Sharpe ratio as the market portfolio. Since only the MVO and robust MVO allowed short selling, they are more likely to make profit in declining market.

Table 4: Return, Variance, Standard Deviation, and Sharpe Ratio of Different Portfolios (Nov 2008)

Portfilio	return (monthly)	variance (monthly)	std (monthly)	Sharpe ratio
MVO	0.215702	0.007081	0.084149	2.528462
Robust MVO 95%	-0.009918	0.001248	0.035323	-0.363844
Robust MVO 90%	-0.007618	0.001318	0.036306	-0.290648
ERC	-0.008341	0.000474	0.021772	-0.517868
Market portfolio	-0.016005	0.000819	0.028615	-0.661860

Part 4c. Efficient Frontier on Oct 2008 Data

100 risk aversion values spaced from 20 to 1000 were used to generate the efficient frontier plot. For each risk aversion value: an MVO portfolio, a 95% confident robust MVO portfolio and a 90% confident robust MVO portfolio was created. The estimated expected returns calculated from historical stock price was used to calculated estimated portfolio returns. Actual return calculated from October price data was used to calculated actual portfolio returns. As for the true portfolio return, average of estimated expected return and actual return was used. The MVO and robust MVO efficient frontiers can be depicted on figure 1.

In the plot, the estimated frontiers, true frontiers and actual frontiers lied apart from each other. This was due to the significant difference between estimated expected return and actual return. Since the estimated expected return was calculated based on historical data from 2004/12/30 to 2008/9/30, which was normal market time that most of the stocks were expected to have positive return. On the other hand, the actual return was calculated based on data from October 2008 crisis, when the market crashed that most of the stocks had negative

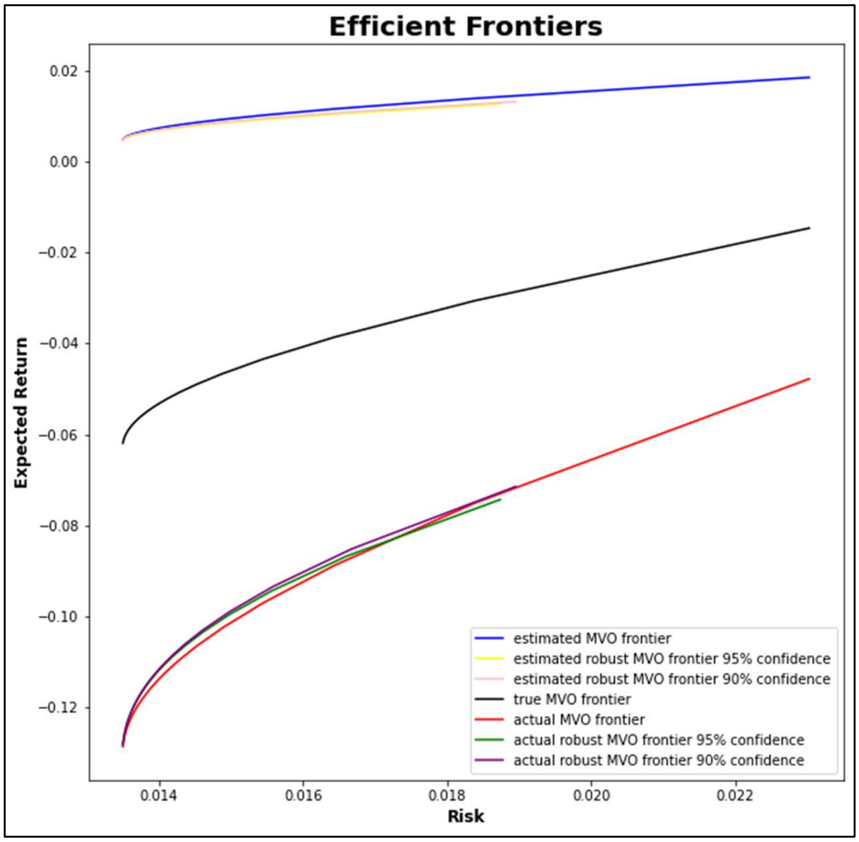


Figure 1: Efficient Frontiers Comparison

returns. Hence the estimated return and actual return were so different apart and caused the department of estimated frontiers and actual frontiers. Moreover, the 95% confident robust MVO frontiers acted similarly to the 90% confident robust MVO frontiers. The 95% and 90% confident actual robust MVO frontiers both outran the actual MVO frontier.

Part 5. Discussion

In this report, returns of four strategies during crisis time were compared: MVO, robust MVO, ERC and market portfolio. Theoretically, ERC model is great at handling risks. However, when the whole market is crashing down as in 2008 crisis, the ERC model could not prevent losing. Since MVO and robust MVO model allowed short selling instead of long only, they have extra option to prevent loss by short selling declining stocks. But this requires the estimation of expected return to be good enough to help investor foreseeing the market crash. The MVO model was considered rather bold among the four models, and it was criticized as "error maximization" sometimes. Surprisingly, it got 30% and 21% return in October and November of 2008 due to its bold short selling against some of the top-loser stocks and it was the only strategy that had all positive returns here. As for the robust MVO, it had the minimal loss among the three losing strategies, which implied that it is a rather reliable strategy regarding risk control. One thing that could be improved is to reduce the time step from monthly to daily. Even though this method will not improve the ability to estimate the return, it will greatly improve the estimate of variance.

Appendix 1: Python code

APS 1022 - Week 2 Project

Name: Fendi Halim Tjoa (1007535017) & Yue Zhang (1007580823)

Instructor: Dr. Roy Kwon

---

we suggest running this notebook on Google Colab

Import Libraries and upload csv file

In [27]:
```
#load 2008 stock price and shares.csv file as 2008 stock price and shares.csv
#this is the only file that needs to be loaded
from google.colab import files
uploaded = files.upload()

#This csv file contains the stock price and shares outstanding data.
#It was downloaded from WRDS(Wharton Research Data Services) database
#with these data we can calculate stocks' market cap in the part "Calculate market po
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving 2008 stock price and shares.csv to 2008 stock price and shares (5).csv
```

In [3]:
```
pip install cvxpy
import cvxpy as cp
import datetime # Package for making dates
import bs4 as bs
import pandas_datareader as web
import numpy as np
import numpy.linalg as linalg
import pandas as pd
```

Data Gathering

- Stock Price Data from Yahoo Finance
- Estimate Expected Returns
- Covariance Matrix

In [4]:
```
#Our investment universe consists of 20 stocks (n=20) all of which are con-
#stituents of the S&P 500. The stock symbols are
#F (Ford Motor Co.);CAT (Catepillar Inc.);DIS;MCD;KO; PEP;WMT;C;
#WFC; JPM ,AAPL; IBM; PFE; JNJ;XOM;MRO;ED; T; V Z; and NEM:

start = datetime.datetime(2004, 12, 30)
end = datetime.datetime(2008, 9, 30)
ticker_symbol = ['F','CAT','DIS','MCD','KO','PEP','WMT','C','WFC','JPM','AAPL','IBM',

n = len(ticker_symbol)
df_stock = web.DataReader(ticker_symbol, 'yahoo', start, end)

#month-end stock prices
monthly_prices = df_stock['Adj Close'].resample('M').ffill()

#calculate monthly returns based on stock prices
monthly_returns = df_stock['Adj Close'].resample('M').ffill().pct_change().iloc[1:,:]
monthly_log_returns = (np.log(df_stock['Adj Close'].resample('M').ffill()) - np.log(d
```

In [5]:
```
#calculate expected returns
exp_returns=((monthly_returns+1).cumprod().iloc[-1])**(1/len(monthly_returns))-1
exp_returns
```

Out[5]:
```
Symbols
F      -0.021158
CAT     0.006003
DIS     0.003179
MCD     0.016680
KO      0.007545
PEP     0.008628
WMT     0.004109
C      -0.015202
WFC     0.007098
JPM     0.006873
AAPL    0.028424
IBM     0.004888
PFE    -0.004837
JNJ     0.003931
XOM     0.010835
MRO     0.018531
ED      0.003832
T       0.005584
VZ     -0.000519
NEM    -0.002289
Name: 2008-09-30 00:00:00, dtype: float64
```

In [6]:
```
#calculate covariance matrix
monthly_returns.cov()
```

Out[6]:

Symbols	F	CAT	DIS	MCD	KO	PEP	WMT	C	WFC
Symbols									
F	0.014721	0.001548	0.001057	0.002166	7.086874e-04	-0.000391	1.724197e-03	0.004618	0.001966
CAT	0.001548	0.004283	0.000862	0.001250	4.969156e-04	-0.000104	-1.611960e-04	0.000337	-0.000827
DIS	0.001057	0.000862	0.001953	0.000913	5.041601e-04	0.000338	1.726771e-04	0.000499	-0.000515
MCD	0.002166	0.001250	0.000913	0.002626	1.013901e-03	0.000560	1.547019e-04	0.001143	0.000282
KO	0.000709	0.000497	0.000504	0.001014	1.234342e-03	0.000708	6.927632e-07	0.000575	0.000264
PEP	-0.000391	-0.000104	0.000338	0.000560	7.084121e-04	0.001304	-1.565357e-04	0.000575	0.000483
WMT	0.001724	-0.000161	0.000173	0.000155	6.927632e-07	-0.000157	1.959695e-03	0.000282	0.000560
C	0.004618	0.000337	0.000499	0.001143	5.748997e-04	0.000575	2.816591e-04	0.006352	0.003721
WFC	0.001966	-0.000827	-0.000515	0.000282	2.643581e-04	0.000483	5.599563e-04	0.003721	0.005304
JPM	0.003574	-0.000236	-0.000423	0.000909	7.300467e-04	0.000724	9.855057e-04	0.004288	0.004728
AAPL	0.001562	0.003675	0.001410	0.004075	1.176363e-03	0.000849	-5.819182e-04	0.000854	-0.002496
IBM	0.002104	0.000983	0.000938	0.001398	4.750712e-04	0.000035	4.633211e-04	0.000991	0.000556
PFE	0.001282	0.001181	0.000169	0.000633	3.679896e-04	0.000283	-7.104404e-04	0.001176	0.001344
JNJ	0.000321	-0.000261	0.000273	0.000538	5.186153e-04	0.000677	2.319626e-04	0.000687	0.000434
XOM	0.002238	0.001743	0.000244	0.001275	5.834055e-04	-0.000149	-8.377282e-04	0.000538	-0.001066
MRO	0.000780	0.002795	0.000657	0.001455	3.519736e-04	-0.000256	-1.978513e-03	-0.000697	-0.002649
ED	0.001087	0.000097	0.000264	0.000758	4.336833e-04	0.000508	-2.099479e-04	0.000907	0.000527
T	0.001222	0.001910	0.000797	0.001092	7.768483e-04	0.000373	2.837188e-05	0.001081	-0.000344
VZ	0.001768	0.001639	0.000725	0.001138	6.244782e-04	0.000397	7.827954e-05	0.001191	-0.000262
NEM	-0.000203	0.001442	-0.001018	0.000067	-6.541535e-04	-0.001172	-1.250594e-03	-0.001138	-0.001545

(1) Mean-Variance Optimization Model (MVO)

Note: short selling is allowed

$$\begin{aligned} \max_{w} \quad & \mu^T x - \lambda x^T Q x \\ \text{s.t.} \quad & \textstyle\sum_i x_i = 1 \end{aligned} \tag{1}$$

In [7]:
```
Q=np.array(monthly_returns.cov())
mu=np.array(exp_returns)
```

In [8]:
```
#define a function for MVO strategy, for later use
def MVO(Q, mu, lambd):
  x1 = cp.Variable(n)
  prob1 = cp.Problem(cp.Maximize(mu.T@x1-cp.quad_form(x1, Q)*lambd),
                     [sum(x1) == 1])
  prob1.solve()

  return(x1.value)
```

(2) Robust mean-variance optimization

using ellipsoidal uncertainty set (assuming allow short selling)

$$\begin{aligned} \max_{w} \quad & \mu^T x - \lambda x^T Q x - \varepsilon_2 \sqrt{x^T \Theta x} \\ \text{s.t.} \quad & \textstyle\sum_i x_i = 1 \end{aligned} \tag{2}$$

In [9]:
```
T=len(monthly_returns)
theta = np.sqrt(np.diag(np.diag(Q))/T)
```

In [10]:
```
#define function for robust MVO strategy, for later use
def robustMVO(Q, mu, lambd, epsilon, theta):
  x2 = cp.Variable(n)
  prob2 = cp.Problem(cp.Maximize(mu.T@x2 - cp.quad_form(x2, Q)*lambd - cp.quad_form(x2
                     [sum(x2) == 1])
  prob2.solve()

  return(x2.value)
```

(3) Risk Parity optimization without short selling

$$\begin{aligned} \min_{w} \quad & \textstyle\sum_{i=1}^{n} (x_i (Qx)_i - \theta)^2 \\ \text{s.t.} \quad & \textstyle\sum_i x_i = 1 \\ & x \geq 0 \end{aligned} \tag{3}$$

In [11]:
```
from scipy.optimize import least_squares

def objectfunc(x):
    return np.array(x[0:20]*(Q@x[0:20])-x[20])
x0 = np.array([0.5]*20+[0])
res_1 = least_squares(objectfunc,x0,bounds=([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-
res_1.x

#The least_squares method doesn't take linear constraints, so we could only normalize
```

Out[11]:
```
array([0.37166534, 0.7312012 , 1.49010214, 0.65210325, 1.26988124,
       2.09222052, 3.263749  , 0.5652594 , 1.17014416, 0.65396833,
       0.34762613, 0.83360135, 0.90719772, 1.85102131, 0.80828565,
       0.73707128, 1.39212151, 0.76340413, 0.77549329, 1.87169341,
       0.01205018])
```

In [12]:
```
#normalize the result, so they sum up to 1
w_erc=res_1.x[0:20]/sum(res_1.x[0:20])
w_erc
```

Out[12]:
```
array([0.01648343, 0.03242892, 0.06608633, 0.02892091, 0.05631949,
       0.09279041, 0.14474794, 0.02506937, 0.05189613, 0.02900363,
       0.01541729, 0.03697039, 0.0402344 , 0.08209317, 0.03584763,
       0.03268926, 0.06174087, 0.03385713, 0.03439329, 0.08300999])
```

In [13]:
```
#check risk contribution
var_ERC = np.dot(w_erc, np.dot(Q, w_erc))
std_ERC = np.sqrt(var_ERC)
RC_ERC = (w_erc * np.dot(Q, w_erc)) / std_ERC
RC_ERC
```

Out[13]:
```
array([0.00108862, 0.00108862, 0.00108862, 0.00108862, 0.00108862,
       0.00108862, 0.00108862, 0.00108862, 0.00108862, 0.00108862,
       0.00108862, 0.00108862, 0.00108862, 0.00108862, 0.00108862,
       0.00108862, 0.00108862, 0.00108862, 0.00108862, 0.00108862])
```

Calculate market portfolio weights for OCT 2008 and NOV 2008

In [14]:
```
import pandas as pd

df = pd.read_csv('2008 stock price and shares.csv')
#calculate market cap
df['Market Cap'] = df['PRC'] * df['SHROUT']
#calculate weights for OCT 2008
OCT_2008 = df[df['date']=='2008/10/31']
OCT_2008['weight'] = OCT_2008['Market Cap']/(OCT_2008['Market Cap'].sum())
#calculate weights for NOV 2008
NOV_2008 = df[df['date']=='2008/11/28']
NOV_2008['weight'] = NOV_2008['Market Cap']/(NOV_2008['Market Cap'].sum())
```

```
<ipython-input-14-62941a7863c3>:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy
  OCT_2008['weight'] = OCT_2008['Market Cap']/(OCT_2008['Market Cap'].sum())
<ipython-input-14-62941a7863c3>:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/use
r_guide/indexing.html#returning-a-view-versus-a-copy
  NOV_2008['weight'] = NOV_2008['Market Cap']/(NOV_2008['Market Cap'].sum())
```

In [15]:
```
OCT_2008['weight']
```

Out[15]:
```
1     0.049306
5     0.005724
9     0.182378
13    0.060415
17    0.042828
21    0.046261
25    0.009931
29    0.011146
33    0.005624
37    0.057745
41    0.082913
45    0.002320
49    0.023517
53    0.054767
57    0.031514
61    0.074361
65    0.106198
69    0.040765
73    0.076306
77    0.035980
Name: weight, dtype: float64
```

In [16]:
```
NOV_2008['weight']
```

Out[16]:
```
2     0.054111
6     0.005515
10    0.203444
14    0.054705
18    0.043944
22    0.041107
26    0.009218
30    0.012339
34    0.007413
38    0.055284
42    0.081106
46    0.003112
50    0.020801
54    0.053755
58    0.032675
62    0.058966
66    0.109698
70    0.046279
74    0.083986
78    0.022544
Name: weight, dtype: float64
```

In [17]:
```
#obtain price data for OCT and NOV 2008 to calculate actual return and variance

start_2008 = datetime.datetime(2008, 8, 28)
end_2008 = datetime.datetime(2008, 11, 30)
ticker_symbol = ['F','CAT','DIS','MCD','KO','PEP','WMT','C','WFC','JPM','AAPL','IBM',

df_stock_2008 = web.DataReader(ticker_symbol, 'yahoo', start_2008, end_2008)

#month returns of stock prices for OCT and NOV 2008
monthly_returns_2008 = df_stock_2008['Adj Close'].resample('M').ffill().pct_change().i

#variances
Q_2008_OCT = monthly_returns_2008.iloc[0:2].cov()
Q_2008_NOV = monthly_returns_2008.iloc[1:3].cov()
```

OCT 2008

In [18]:
```
#average US treasury yield for OCT.2008 was 3.8072%
rf_OCT=3.8072/100
```

In [19]:
```
x_market = np.array(OCT_2008.set_index(['TICKER']).reindex(ticker_symbol)['weight'])
var_market = x_market@Q@x_market

#calculate risk aversion coefficient lambd based of the market portfolio
lambd_OCT = (mu@x_market-rf_OCT/12)/var_market
lambd_OCT
```

Out[19]:
```
3.820287489857027
```

In [20]:
```
#set weights of the four portfolios as arries
x_mvo = MVO(Q, mu, lambd_OCT)
x_robust_95 = robustMVO(Q,mu, lambd_OCT, 1.96, theta)
x_robust_90 = robustMVO(Q,mu, lambd_OCT, 1.645, theta)
x_erc = w_erc.copy()
x_market = np.array(OCT_2008.set_index(['TICKER']).reindex(ticker_symbol)['weight'])

return_mvo = monthly_returns_2008.iloc[1] @ x_mvo
return_robust_95 = monthly_returns_2008.iloc[1] @ x_robust_95
return_robust_90 = monthly_returns_2008.iloc[1] @ x_robust_90
return_erc = monthly_returns_2008.iloc[1] @ x_erc
return_market = monthly_returns_2008.iloc[1] @ x_market

var_mvo = x_mvo@Q@x_mvo
var_robust_95 = x_robust_95@Q@x_robust_95
var_robust_90 = x_robust_90@Q@x_robust_90
var_erc = x_erc@Q@x_erc
var_market = x_market@Q@x_market

result_OCT_2008 = pd.DataFrame({'Portfilio':['MVO', 'Robust MVO 95%','Robust MVO 90%',
                    'return (monthly)':[return_mvo,return_robust_95,return_robust_
                    'variance (monthly)':[var_mvo,var_robust_95,var_robust_90,var_e
                    'std (monthly)':[np.sqrt(var_mvo),np.sqrt(var_robust_95),np.sq

result_OCT_2008['Sharpe ratio'] = (result_OCT_2008['return (monthly)']-rf_OCT/12)/resu

result_OCT_2008
```

Out[20]:

	Portfilio	return (monthly)	variance (monthly)	std (monthly)	Sharpe ratio
0	MVO	0.301900	0.009725	0.098614	3.029251
1	Robust MVO 95%	-0.003076	0.001435	0.037888	-0.164914
2	Robust MVO 90%	0.010489	0.001624	0.040295	0.181580
3	ERC	-0.145946	0.000474	0.021772	-6.848979
4	Market portfolio	-0.103726	0.000840	0.028974	-3.689422

NOV 2008

In [21]:
```
#average US treasury yield for NOV.2008 was 3.521%
rf_NOV=3.521/100
```

In [22]:
```
x_market = np.array(NOV_2008.set_index(['TICKER']).reindex(ticker_symbol)['weight'])
var_market = x_market@Q@x_market

#calculate risk aversion coefficient lambd based of the market portfolio
lambd_NOV = (mu@x_market-rf_NOV/12)/var_market
lambd_NOV
```

Out[22]:
```
4.493019828474737
```

In [23]:
```
#set weights of the four portfolios as arries
x_mvo = MVO(Q,mu,lambd_NOV)
x_robust_95 = robustMVO(Q, mu, lambd_NOV, 1.96,theta)
x_robust_90 = robustMVO(Q,mu, lambd_NOV, 1.645, theta)
x_erc = w_erc
x_market = np.array(NOV_2008.set_index(['TICKER']).reindex(ticker_symbol)['weight'])

return_mvo = monthly_returns_2008.iloc[2] @ x_mvo
return_robust_95 = monthly_returns_2008.iloc[2] @ x_robust_95
return_robust_90 = monthly_returns_2008.iloc[2] @ x_robust_90
return_erc = monthly_returns_2008.iloc[2] @ x_erc
return_market = monthly_returns_2008.iloc[2] @ x_market

var_mvo = x_mvo@Q@x_mvo
var_robust_95 = x_robust_95@Q@x_robust_95
var_robust_90 = x_robust_95@Q@x_robust_90
var_erc = x_erc@Q@x_erc
var_market = x_market@Q@x_market

result_NOV_2008 = pd.DataFrame({'Portfilio':['MVO', 'Robust MVO 95%','Robust MVO 90%',
                    'return (monthly)':[return_mvo,return_robust_95,return_robust_
                    'variance (monthly)':[var_mvo,var_robust_95,var_robust_90,var_e
                    'std (monthly)':[np.sqrt(var_mvo),np.sqrt(var_robust_95),np.sq

result_NOV_2008['Sharpe ratio'] = (result_NOV_2008['return (monthly)']-rf_NOV/12)/resu

result_NOV_2008
```

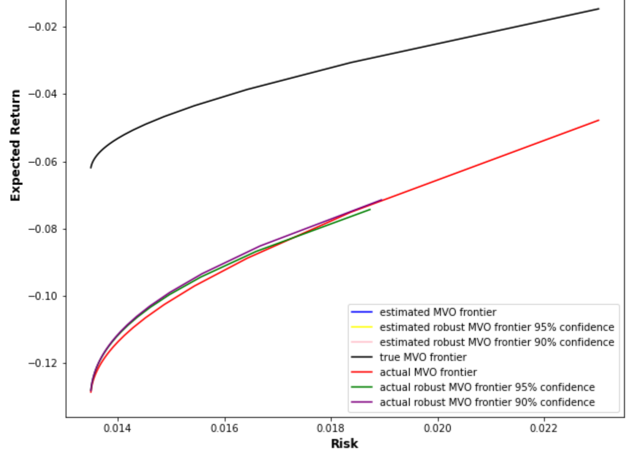
Out[23]:

	Portfilio	return (monthly)	variance (monthly)	std (monthly)	Sharpe ratio
0	MVO	0.215702	0.007081	0.084149	2.528462
1	Robust MVO 95%	-0.009918	0.001248	0.035323	-0.363843
2	Robust MVO 90%	-0.007618	0.001318	0.036306	-0.290648
3	ERC	-0.008341	0.000474	0.021772	-0.517870
4	Market portfolio	-0.016005	0.000819	0.028615	-0.661860

Plot efficient frontiers for MVO and robust MVO

In [24]:
```
ret_MVO=[]
ret_MVO_actual=[]
ret_MVO_true=[]
var_MVO=[]
ret_robust_95=[]
ret_robust_actual_95=[]
var_robust_95=[]
ret_robust_90=[]
ret_robust_actual_90=[]
var_robust_90=[]

for lambd_i in np.linspace(1000,20,100):
  #calculate estimated portfolio return, actual portfolio return , true return and po
  x_MVO=MVO(Q,mu,lambd_i)
  ret_MVO.append(mu@x_MVO)
  ret_MVO_actual.append(monthly_returns_2008.iloc[1]@x_MVO)
  var_MVO.append(x_MVO@Q@x_MVO)
  ret_MVO_true.append(((mu+monthly_returns_2008.iloc[1])/2)@x_MVO)

  #calculate estimated portfolio return, actual portfolio return and portfolio varian
  x_robust = robustMVO(Q,mu,lambd_i, 1.96, theta)
  ret_robust_95.append(mu@x_robust)
  ret_robust_actual_95.append(monthly_returns_2008.iloc[1]@x_robust)
  var_robust_95.append(x_robust@Q@x_robust)

  #calculate estimated portfolio return, actual portfolio return and portfolio varian
  x_robust = robustMVO(Q,mu,lambd_i, 1.645, theta)
  ret_robust_90.append(mu@x_robust)
  ret_robust_actual_90.append(monthly_returns_2008.iloc[1]@x_robust)
  var_robust_90.append(x_robust@Q@x_robust)

#Plot
from matplotlib import pyplot as plt
plt.figure(figsize=(10,10))
plt.title("Efficient Frontiers", fontweight = 'bold', fontsize=20)
plt.xlabel("Risk", fontweight = 'bold', fontsize=12)
plt.ylabel('Expected Return',fontweight = 'bold', fontsize=12)
ax = plt.gca()
# Plot efficient frontier
ax.plot(np.sqrt(var_MVO), ret_MVO, color='Blue', label='estimated MVO frontier')
ax.plot(np.sqrt(var_robust_95), ret_robust_95, color='yellow', label='estimated robust
ax.plot(np.sqrt(var_robust_90), ret_robust_90, color='pink', label='estimated robust M
ax.plot(np.sqrt(var_MVO), ret_MVO_true, color='black', label='true MVO frontier')
ax.plot(np.sqrt(var_MVO), ret_MVO_actual, color='Red', label='actual MVO frontier')
ax.plot(np.sqrt(var_robust_95), ret_robust_actual_95, color='green', label='actual rob
ax.plot(np.sqrt(var_robust_90), ret_robust_actual_90, color='purple', label='actual ro
ax.legend()
```

Out[24]:
```
<matplotlib.legend.Legend at 0x1a2f5578850>
```



Appendix 2: Matlab code for Equal Risk Parity model

```
clear
clc
% data of covariance matrix is calcualted from python file
cov_matrix_raw = readmatrix('risk_parity_cov.xlsx'); % import cov
 matrix
Q = cov_matrix_raw(2:end,:); % covariance matrix
x = optimvar('x',length(Q)); % optimization variable (x_1 to x_20)
theta = optimvar('theta'); % dummy variable theta
obj = sum((x.*(Q*x)-theta).^2); % cost function
prob = optimproblem("Objective",obj); % optimization problem

% constraints
cons1 = sum(x) == 1; % sum of all the weights have to be equal to 1
cons2 = x >= 0; % long only
cons3 = x.*(Q*x) == theta; % at optimality, x_i*(Q*x)_i = theta for
 all i
prob.Constraints.cons1 = cons1; % attach constraint 1 to prob
prob.Constraints.cons2 = cons2; % attach constraint 2 to prob
prob.Constraints.cons3 = cons3; % attach constraint 3 to prob

%initial point
x0.x = zeros(1,length(Q))';
x0.theta = 0;

fprintf('ERC Optimization\n')
show(prob) % display optimization problem along with constraints

% solving optimization problem using SOLVE function
% given prob and x0, SOLVE function will choose fmincon to solve
[sol,fval] = solve(prob,x0);

% result
fprintf('Weights of each asset from x1 to x20 respectively:\n')
disp((sol.x)') % Weight of each asset
fprintf('Risk of each asset from x1 to x20 respectively:\n')
disp((sol.x.*(Q*sol.x))') % Risk vector
```

```
ERC Optimization

  OptimizationProblem :

 Solve for:
       theta, x

 minimize :
       sum(((x .* (extraParams{1} * x)) - theta).^2)

         extraParams{1}:

       Columns 1 through 7

         0.0147    0.0015    0.0011    0.0022    0.0007   -0.0004
 0.0017
```

```
         0.0015    0.0043    0.0009    0.0013    0.0005   -0.0001
-0.0002
         0.0011    0.0009    0.0020    0.0009    0.0005    0.0003
0.0002
         0.0022    0.0013    0.0009    0.0026    0.0010    0.0006
0.0002
         0.0007    0.0005    0.0005    0.0010    0.0012    0.0007
0.0000
        -0.0004   -0.0001    0.0003    0.0006    0.0007    0.0013
-0.0002
         0.0017   -0.0002    0.0002    0.0002    0.0000   -0.0002
0.0020
         0.0046    0.0003    0.0005    0.0011    0.0006    0.0006
0.0003
         0.0020   -0.0008   -0.0005    0.0003    0.0003    0.0005
0.0006
         0.0036   -0.0002   -0.0004    0.0009    0.0007    0.0007
0.0010
         0.0016    0.0037    0.0014    0.0041    0.0012    0.0008
-0.0006
         0.0021    0.0010    0.0009    0.0014    0.0005    0.0000
0.0005
         0.0013    0.0012    0.0002    0.0006    0.0004    0.0003
-0.0007
         0.0003   -0.0003    0.0003    0.0005    0.0005    0.0007
0.0002
         0.0022    0.0017    0.0002    0.0013    0.0006   -0.0001
-0.0008
         0.0008    0.0028    0.0007    0.0015    0.0004   -0.0003
-0.0020
         0.0011    0.0001    0.0003    0.0008    0.0004    0.0005
-0.0002
         0.0012    0.0019    0.0008    0.0011    0.0008    0.0004
0.0000
         0.0018    0.0016    0.0007    0.0011    0.0006    0.0004
0.0001
        -0.0002    0.0014   -0.0010    0.0001   -0.0007   -0.0012
-0.0013

      Columns 8 through 14

         0.0046    0.0020    0.0036    0.0016    0.0021    0.0013
0.0003
         0.0003   -0.0008   -0.0002    0.0037    0.0010    0.0012
-0.0003
         0.0005   -0.0005   -0.0004    0.0014    0.0009    0.0002
0.0003
         0.0011    0.0003    0.0009    0.0041    0.0014    0.0006
0.0005
         0.0006    0.0003    0.0007    0.0012    0.0005    0.0004
0.0005
         0.0006    0.0005    0.0007    0.0008    0.0000    0.0003
0.0007
```

```
         0.0003    0.0006    0.0010   -0.0006    0.0005   -0.0007
0.0002
         0.0064    0.0037    0.0043    0.0009    0.0010    0.0012
0.0007
         0.0037    0.0053    0.0047   -0.0025    0.0006    0.0013
0.0004
         0.0043    0.0047    0.0058   -0.0006    0.0007    0.0013
0.0006
         0.0009   -0.0025   -0.0006    0.0166    0.0023    0.0008
0.0006
         0.0010    0.0006    0.0007    0.0023    0.0033   -0.0001
0.0004
         0.0012    0.0013    0.0013    0.0008   -0.0001    0.0033
0.0002
         0.0007    0.0004    0.0006    0.0006    0.0004    0.0002
0.0010
         0.0005   -0.0011   -0.0005    0.0040    0.0007    0.0010
0.0001
        -0.0007   -0.0026   -0.0024    0.0055    0.0009    0.0015
-0.0004
         0.0009    0.0005    0.0006    0.0011   -0.0000    0.0007
0.0003
         0.0011   -0.0003    0.0005    0.0035    0.0010    0.0009
0.0003
         0.0012   -0.0003    0.0005    0.0029    0.0009    0.0009
0.0003
        -0.0011   -0.0015   -0.0012    0.0026    0.0001    0.0009
-0.0009

      Columns 15 through 20

         0.0022    0.0008    0.0011    0.0012    0.0018   -0.0002
         0.0017    0.0028    0.0001    0.0019    0.0016    0.0014
         0.0002    0.0007    0.0003    0.0008    0.0007   -0.0010
         0.0013    0.0015    0.0008    0.0011    0.0011    0.0001
         0.0006    0.0004    0.0004    0.0008    0.0006   -0.0007
        -0.0001   -0.0003    0.0005    0.0004    0.0004   -0.0012
        -0.0008   -0.0020   -0.0002    0.0000    0.0001   -0.0013
         0.0005   -0.0007    0.0009    0.0011    0.0012   -0.0011
        -0.0011   -0.0026    0.0005   -0.0003   -0.0003   -0.0015
        -0.0005   -0.0024    0.0006    0.0005    0.0005   -0.0012
         0.0040    0.0055    0.0011    0.0035    0.0029    0.0026
         0.0007    0.0009   -0.0000    0.0010    0.0009    0.0001
         0.0010    0.0015    0.0007    0.0009    0.0009    0.0009
         0.0001   -0.0004    0.0003    0.0003    0.0003   -0.0009
         0.0037    0.0042    0.0005    0.0011    0.0011    0.0019
         0.0042    0.0100    0.0008    0.0015    0.0016    0.0034
         0.0005    0.0008    0.0015    0.0005    0.0003   -0.0002
         0.0011    0.0015    0.0005    0.0034    0.0022   -0.0004
         0.0011    0.0016    0.0003    0.0022    0.0028   -0.0001
         0.0019    0.0034   -0.0002   -0.0004   -0.0001    0.0074
```

```
subject to cons1:
      x(1) + x(2) + x(3) + x(4) + x(5) + x(6) + x(7) + x(8) + x(9) +
x(10)
    + x(11) + x(12) + x(13) + x(14) + x(15) + x(16) + x(17) + x(18) +
x(19)
    + x(20) == 1

subject to cons2:
      x(1) >= 0
      x(2) >= 0
      x(3) >= 0
      x(4) >= 0
      x(5) >= 0
      x(6) >= 0
      x(7) >= 0
      x(8) >= 0
      x(9) >= 0
      x(10) >= 0
      x(11) >= 0
      x(12) >= 0
      x(13) >= 0
      x(14) >= 0
      x(15) >= 0
      x(16) >= 0
      x(17) >= 0
      x(18) >= 0
      x(19) >= 0
      x(20) >= 0

subject to cons3:
      (x .* (extraParams{1} * x)) == arg_RHS

      where:

        arg1 = theta(ones(1,20));
        arg_RHS = arg1(:);

        extraParams{1}:

      Columns 1 through 7

        0.0147    0.0015    0.0011    0.0022    0.0007   -0.0004
0.0017
        0.0015    0.0043    0.0009    0.0013    0.0005   -0.0001
-0.0002
        0.0011    0.0009    0.0020    0.0009    0.0005    0.0003
0.0002
        0.0022    0.0013    0.0009    0.0026    0.0010    0.0006
0.0002
        0.0007    0.0005    0.0005    0.0010    0.0012    0.0007
0.0000
       -0.0004   -0.0001    0.0003    0.0006    0.0007    0.0013
-0.0002
```

```
          0.0017   -0.0002    0.0002    0.0002    0.0000   -0.0002
0.0020
          0.0046    0.0003    0.0005    0.0011    0.0006    0.0006
0.0003
          0.0020   -0.0008   -0.0005    0.0003    0.0003    0.0005
0.0006
          0.0036   -0.0002   -0.0004    0.0009    0.0007    0.0007
0.0010
          0.0016    0.0037    0.0014    0.0041    0.0012    0.0008
-0.0006
          0.0021    0.0010    0.0009    0.0014    0.0005    0.0000
0.0005
          0.0013    0.0012    0.0002    0.0006    0.0004    0.0003
-0.0007
          0.0003   -0.0003    0.0003    0.0005    0.0005    0.0007
0.0002
          0.0022    0.0017    0.0002    0.0013    0.0006   -0.0001
-0.0008
          0.0008    0.0028    0.0007    0.0015    0.0004   -0.0003
-0.0020
          0.0011    0.0001    0.0003    0.0008    0.0004    0.0005
-0.0002
          0.0012    0.0019    0.0008    0.0011    0.0008    0.0004
0.0000
          0.0018    0.0016    0.0007    0.0011    0.0006    0.0004
0.0001
         -0.0002    0.0014   -0.0010    0.0001   -0.0007   -0.0012
-0.0013

      Columns 8 through 14

          0.0046    0.0020    0.0036    0.0016    0.0021    0.0013
0.0003
          0.0003   -0.0008   -0.0002    0.0037    0.0010    0.0012
-0.0003
          0.0005   -0.0005   -0.0004    0.0014    0.0009    0.0002
0.0003
          0.0011    0.0003    0.0009    0.0041    0.0014    0.0006
0.0005
          0.0006    0.0003    0.0007    0.0012    0.0005    0.0004
0.0005
          0.0006    0.0005    0.0007    0.0008    0.0000    0.0003
0.0007
          0.0003    0.0006    0.0010   -0.0006    0.0005   -0.0007
0.0002
          0.0064    0.0037    0.0043    0.0009    0.0010    0.0012
0.0007
          0.0037    0.0053    0.0047   -0.0025    0.0006    0.0013
0.0004
          0.0043    0.0047    0.0058   -0.0006    0.0007    0.0013
0.0006
          0.0009   -0.0025   -0.0006    0.0166    0.0023    0.0008
0.0006
```

```
         0.0010    0.0006    0.0007    0.0023    0.0033   -0.0001
0.0004
         0.0012    0.0013    0.0013    0.0008   -0.0001    0.0033
0.0002
         0.0007    0.0004    0.0006    0.0006    0.0004    0.0002
0.0010
         0.0005   -0.0011   -0.0005    0.0040    0.0007    0.0010
0.0001
        -0.0007   -0.0026   -0.0024    0.0055    0.0009    0.0015
-0.0004
         0.0009    0.0005    0.0006    0.0011   -0.0000    0.0007
0.0003
         0.0011   -0.0003    0.0005    0.0035    0.0010    0.0009
0.0003
         0.0012   -0.0003    0.0005    0.0029    0.0009    0.0009
0.0003
        -0.0011   -0.0015   -0.0012    0.0026    0.0001    0.0009
-0.0009

      Columns 15 through 20

         0.0022    0.0008    0.0011    0.0012    0.0018   -0.0002
         0.0017    0.0028    0.0001    0.0019    0.0016    0.0014
         0.0002    0.0007    0.0003    0.0008    0.0007   -0.0010
         0.0013    0.0015    0.0008    0.0011    0.0011    0.0001
         0.0006    0.0004    0.0004    0.0008    0.0006   -0.0007
        -0.0001   -0.0003    0.0005    0.0004    0.0004   -0.0012
        -0.0008   -0.0020   -0.0002    0.0000    0.0001   -0.0013
         0.0005   -0.0007    0.0009    0.0011    0.0012   -0.0011
        -0.0011   -0.0026    0.0005   -0.0003   -0.0003   -0.0015
        -0.0005   -0.0024    0.0006    0.0005    0.0005   -0.0012
         0.0040    0.0055    0.0011    0.0035    0.0029    0.0026
         0.0007    0.0009   -0.0000    0.0010    0.0009    0.0001
         0.0010    0.0015    0.0007    0.0009    0.0009    0.0009
         0.0001   -0.0004    0.0003    0.0003    0.0003   -0.0009
         0.0037    0.0042    0.0005    0.0011    0.0011    0.0019
         0.0042    0.0100    0.0008    0.0015    0.0016    0.0034
         0.0005    0.0008    0.0015    0.0005    0.0003   -0.0002
         0.0011    0.0015    0.0005    0.0034    0.0022   -0.0004
         0.0011    0.0016    0.0003    0.0022    0.0028   -0.0001
         0.0019    0.0034   -0.0002   -0.0004   -0.0001    0.0074



Solving problem using fmincon.

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-
decreasing in
feasible directions, to within the value of the optimality tolerance,
```

```
and constraints are satisfied to within the value of the constraint
 tolerance.

Weights of each asset from x1 to x20 respectively:
  Columns 1 through 7

    0.0165    0.0324    0.0661    0.0289    0.0563    0.0928    0.1450

  Columns 8 through 14

    0.0251    0.0519    0.0290    0.0154    0.0370    0.0402    0.0821

  Columns 15 through 20

    0.0358    0.0327    0.0617    0.0338    0.0344    0.0830

Risk of each asset from x1 to x20 respectively:
   1.0e-04 *

  Columns 1 through 7

    0.2368    0.2369    0.2369    0.2369    0.2369    0.2368    0.2380

  Columns 8 through 14

    0.2369    0.2369    0.2369    0.2368    0.2369    0.2369    0.2369

  Columns 15 through 20

    0.2369    0.2369    0.2369    0.2369    0.2369    0.2368
```

Published with MATLAB® R2021a