# Risk-Budgeted Portfolio Construction using Metaheuristics

Graphical user interface, text, application

Description automatically generated

# 1. What is a Risk-Budgeted portfolio?

Risk Budgeting also referred to as Risk Decomposition or Risk Contribution is an investment strategy where portfolio risk is distributed across assets or asset classes in the portfolio, ensuring investment exposure as well as market protection at the same time. This is effected by imposing limits on the individual risks of the assets or asset classes comprising the portfolio and ensuring that the asset allocation is done in strict compliance to the risk limits imposed.

Risk-Budgeted portfolios can be long-short portfolios or long-only portfolios, with or without leveraged bounds, as dictated by the investor's preferences and risk appetites.

# 2. Risk Budgeted Portfolio Construction

Let us consider an investor who desires to construct a leveraged long-short Risk-Budgeted portfolio, optimal with regard to an objective function and subject to constraints that reflect restrictions on asset allocation and the investor's preferences.

(1) The objective is to maximize Sharpe Ratio of the portfolio.

(See <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson6_SharpeRatioOptimization/Lesson6_MainContent.ipynb> to know about Sharpe Ratio based portfolio optimization)

The following constraints are imposed on the portfolio by the investor:

(2) A budget constraint on the portfolio, where the capital is fully invested (i.e.)the sum of the portfolio weights equals 1.

(3) Leveraged bounds of [0.001, +3] on investor chosen long positions in which investment is mandatory.

(4) Leveraged bounds of [-3, 0] on investor chosen short positions in which investment is optional.

(5) Leveraged bounds of [0, +3] on all other long positions in the portfolio in which investment is optional.

Here the bounds [-3, +3] are indicative of the fact that the investor is open to leveraging investments in the positions concerned to any extent, not withstanding the fact that the portfolio needs to be a fully invested one (that is, sum of weights equals 1).

Risk Budgeting is enforced through what is called Marginal Contribution to Risk (MCR) and is defined as the partial derivative of the risk with respect to its weights as given below.

A picture containing text

Description automatically generated

where $\bar m = (m\_1, m\_2, ...m\_N)'$ is the weight set and V is the variance-covariance matrix of returns.

The Absolute Contribution to Total Risk is given by



The Percentage Contribution to Total Risk is given by



The sum of the Absolute Contribution to Total Risk for a portfolio equals its risk.

The Risk Budgeting constraint is defined as,

(6) The percentage contribution to risk(%) of the portfolio cannot exceed r%, where r is a risk limit imposed by the investor.

Thus, the objective and constraints of the Risk-Budgeted portfolio optimization model are described by points (1) -(6).

# 3. Mathematical Formulation of the Investor's Risk-Budgeted Portfolio Model

Let us suppose that the investor desires to construct a portfolio P comprising assets $A\_1, A\_2, ...A\_N$, with $\bar \mu\_{(1 X N)} = [\mu\_1, \mu\_2, ...\mu\_N]$ as the asset returns and $\bar W\_{(N X1)}=[W\_1, W\_2, ...W\_N]'$ as the weights.

Let $R\_f$ be the risk free rate of return.

The portfolio return $r$ determined by a weighted summation of its individual asset returns is given by $\bar \mu . \bar W = \sum\left({\mu\_i.W\_i}\right)$ and the risk is given by $\sigma\_P=\sqrt{{\bar W}'.V.\bar W}$, where V is the variance-covariance matrix of returns.

(See <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson1_FundaRiskReturnPortfolio/Lesson1_MainContent.ipynb>, to know about risk and return of a portfolio)

The mathematical model for Risk-Budgeted Portfolio Optimization whose objective function and constraints were defined in the previous section, is as shown below:

1.Objective function for maximizing Sharpe Ratio of the portfolio



subject to the constraints that describe constraints (2)-(6) listed in Sec. 2. viz.,

Text

Description automatically generated

In the above system of equations, $W\_i^{long}$, $W\_j^{Short}$ and $W\_k^{Others}$ indicate the weights of the mandatory long postions, optional short positions and optional long positions to be invested in the optimal portfolio. $\sigma\_P$ denotes portfolio risk. $m\_i$ indicates the marginal contribution to risk defined by equation [1]. Equations [5] to [8] illustrate the budget and leveraged bound constraints imposed on the long and short positions. Equation [9] denotes the risk budgeting constraint.

# 4. Metaheuristic solution strategy

We proceed to undertake the optimization of the model described by equations [4] to [9] employing a robust metaheuristic strategy viz., Differential Evolution with Hall of Fame (DE HOF). DE HOF dynamically arrives at the optimal composition of long and short positions of the portfolio that yield the maximal Sharpe Ratio, subject to all the constraints enforced on the assets and the portfolio.

(See Chapter 2: A Brief Primer on Metaheuristics [PAI 2018], to know more about metaheuristics and Differential Evolution)

Metaheuristic strategies build feasible solution sets that satisfy the constraints imposed, during the course of their execution process. However constraint handling has been a major problem in the application of metaheuristic strategies to complex constrained optimization problems, to tackle which several methods such as Penalty Function Strategy and Repair Strategy have been proposed.

For the Risk Budgeted portfolio optimization model, DE HOF makes use of Joines and Houck's Dynamic Penalty Function Strategy [JOI 1994] to handle the Risk Budgeting constraint (equation [9]) and Repair Strategies to handle the budget constraint on the portfolio and the leveraged bounds constraints on the long /short positions (equations [5] to [8]).

Repair strategies are custom made to suit the requirements of the problem and evolving such a strategy that will help satisfy one or more constraints at one go, can turn out to be difficult. Nevertheless, once the strategy is evolved, it can help churn out populations of feasible solution sets leading to faster convergence of the metaheuristic strategy.

(See Sec. 4.5 of Chapter 4: Metaheuristic Risk-Budgeted Portfolio Optimization [PAI 2018], to know about the Repair Strategies evolved for the Risk-Budgeted portfolio optimization model)

# 5. Transformation of the Mathematical Model

Joines and Houck's dynamic penalty function strategy is used to tackle the Risk Budgeting constraint represented by equation [9]. The constraint is accomodated in the "penalized" objective function by defining appropriate penalty functions. The transformed mathematical model is shown below.

Text

Description automatically generated

In the system of equations [11], $(C, \alpha, \beta)$ are all constants and the penalty term $(C.t)^\alpha$ increases constantly with each generation count t of the metaheuristic strategy DE HOF.

DE HOF now strives to solve the optimization model whose penalized objective function is described by equations [10]-[11] subject to the budget and leveraged bounds constraints imposed on the long and short positions of the portfolio, described by equations [5]-[8] only.

To reiterate, DE HOF employs repair strategies to tackle these constraints resulting in its exploring only feasible solution space which eventually leads to faster convergence.

# 6. Differential Evolution with Hall of fame - a brief roundup

DE HOF is a population based metaheuristic strategy that can efficiently solve complex constrained optimization problems. We restrict the discussion to the DE HOF designed to solve the Risk-Budgeted portfolio optimization problem model. The input, process and output of the DE HOF strategy are as follows:

### Inputs

It is essential that the portfolio parameters and the DE HOF strategy parameters are clearly set before the optimization process begins.

The portfolio parameters are (1) assets in the portfolio (2) mean asset returns (4) covariance of returns (the variance-covariance matrix of returns) (4) risk limit r% and (5) risk free rate of return.

The DE HOF parameters are (1) population size (2) number of generations (3) dynamic penalty function parameters $(C, \alpha, \beta)$ (4) Scale factor $\beta\_S$ and (5) probability of recombination $p\_r$.

### Process

DE HOF begins its execution by generating an initial random population of individuals that represent the candidate solution sets to the optimization problem concerned. Each individual in the population represents a collection of N genes that represent random weights generated in the range [-3, +3] for the N assets in the portfolio.

The candidate solution sets are transformed into feasible solution sets by calling the weight repair strategies which repair each individual in the population, so that it satisfies constraints represented by equations [5] to [8]. At the end of the process the initial population is transformed into a population that is a feasible solution set and is termed as the parent population

Typical of metaheuristic strategies, DE HOF now computes the fitness function values of each individual in the parent population making use of the penalized objective function described by equations [10]-[11].

Making use of mutation and crossover operators (Binomial Crossover operator) and the scale factor $\beta\_A$, DE HOF generates what are called trial vectors which eventually leads to the production of the offspring population of individuals.

The offspring population of individuals is standardized to satisfy the respective constraints, by calling the weight repair strategies once again and their fitness function values are computed as was done for the parent population.

Based on the fitness function values of the individuals in the parent population and offspring population, the best fit individuals are selected using the Deterministic Selection operator and pushed to the next generation. The best among the better individuals selected for the next generation with maximal fitness function value and zero penalty function value (no constraints violated) is inducted into the Hall of Fame.

At the end of the first generation, the next generation pool of individuals get set as the parent population and the second generation begins. The second generation offspring population is now generated by repeating the reproduction process and the best among the better individuals selected for the third generation competes with the individual in the Hall of Fame. DE HOF ensures that only the best individual by way of maximal fitness function value and zero penalty function value generated this far, is inducted into the Hall of Fame.

The generation cycles progress until the termination criterion is met with, at which stage the individual in the Hall of Fame is declared to be the optimal solution to the problem concerned.

### Output

The genes of the HOF individual represent the optimal weights or proportion of capital to be invested in the assets of the portfolio. Positive weights indicate assets that are to be longed and negative weights indicate assets that are to be shorted. Given the mean returns and the covariance of returns, the risk and return of the optimal Risk-Budgeted portfolio can be easily computed. It can be verified that the portfolio is fully invested and all the long and short positions in the optimal portfolio satisfy their respective bounds constraints and the percentage contribution to risk of the portfolio lies within the stipulated risk limits.

(See Sec. 4.6 of Chapter 4 Metaheuristic Risk-Budgeted Portfolio Optimization [PAI 2018] to know more about the design, process flow chart and execution of Differential Evolution with Hall of Fame based Risk-Budgeted Portfolio construction)

# 7. Case Study

We proceed to demonstrate the construction of Risk-Budgeted long-short portfolios over S&P BSE200 (April 2009- July 2020) data set of Bombay Stock Exchange, India, which includes the Covid 19 crisis period as well. To keep the story short, we assume that the investor has already made a technically diverse choice of assets in the portfolio (a k-portfolio, in fact) and is ready with the mean returns $\mu$ and the variance-covariance matrix of returns V.

Note that a k-portfolio is an outcome of a heuristic portfolio selection strategy, where the universe of stocks is grouped into clusters that display intra-class similarity and inter-class dissimilarity with regard to the mean-returns and covariance of returns. Since assets belonging to a cluster are similar in behavior, the investor now makes a choice of one asset each from each cluster to ensure diversification of assets in the portfolio. A clustering technique such as k-means algorithm can be used to group the stock universe into k clusters with the investor exercising the choice of k.

(See <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson3_HeuristicPortfolioSelection/Lesson3_MainContent.ipynb> and Chapter 3 Heuristic Portfolio Selection[PAI 2018], to know more about the construction of k-portfolios and their merits)

Let the k-portfolio selected by the investor comprise the following 30 assets, after making a heuristic portfolio selection for k = 30:

Adani Enterprises Ltd. ["'ADANIENT'"], Astral Poly Technik Ltd. ["'ASTRAL'"], Bajaj Finance Ltd.["'BAJFINANCE'"], Bharat Forge Ltd. ["'BHARATFORG'"], Bharat Petroleum Corp Ltd. ["'BPCL'"], Cholamandalam Investment and Fin Co Ltd. ["'CHOLAFIN'"], Dabur India Ltd["'DABUR'"], GAIL (India) Ltd. ["'GAIL'"], HDFC Bank Ltd.["'HDFCBANK'"], Hindustan Petroleum Corporation Ltd. [ "'HINDPETRO'"], Vodafone Idea Ltd. ["'IDEA'"], Indusind Bank Ltd. ["'INDUSINDBK'"], Indian Oil Corporation Ltd.[ "'IOC'"], ITC Ltd. ["'ITC'"], JSW Steel Ltd.[ "'JSWSTEEL'"], Jubilant Life Sciences Ltd.,["'JUBILANT'"], Larsen & Toubro Ltd.["'LT'"], MindTree Ltd. ["'MINDTREE'"], Mahindra &Mahindra Ltd. ["'M&M'"], Nestle India Ltd. ["'NESTLEIND'"], NMDC Ltd. ["'NMDC'"], Petronet LNG Ltd. ["'PETRONET'"], REC Ltd. ["'RECLTD'"], Relaxo Footwears Ltd. ["'RELAXO'"], State Bank of India Ltd. ["'SBIN'"], Sun Pharmaceutical Industries Ltd. ["'SUNPHARMA'"], Tata Motors Ltd. ["'TATAMOTORS'"], Tata Consultancy Services Ltd. ["'TCS'"],Whirlpool of India Ltd. ["'WHIRLPOOL'"], Wipro Ltd. ["'WIPRO'"]

The objective is to construct a Risk-Budgeted portfolio that will yield maximal Sharpe Ratio subject to the constraints listed in equations [5]-[9] for a risk budget r =12.5%.

A fragment of the CSV file S&PBSE200\_kPortfolioParams.csv, which describes the asset labels, mean returns and variance-covariance matrix of returns, to be used by DE HOF for the construction of optimal Risk-Budgeted portfolio is shown below:

Table

Description automatically generated

#### Fig. 1 Structure of the input file to DE HOF that captures details about assets in the portfolio, their mean and covariance of returns (daily %)

# 8. Python coding of DE HOF for Risk-Budgeted Portfolio Construction

The DE HOF program is a conglomeration of functions typical of any metaheuristic strategy. The functions are listed first followed by the main program, with a brief description of the task accomplished by the function code.

### 8.1 Function SatisfyLowBounds

In the risk-budgeted model which demands leveraging, specific asset weights in the portfolio have stipulated lower bounds defined by the investor. However, the input to the function is a population of individuals whose weights may or may not satisfy their respective lower bounds. Therefore this function prepares to standardize each asset weight in the population to satisfy their respective lower bounds, in preparation for the next stage where the entire population of individuals will be standardized to satisfy the budget and bounds constraints described by equations [5]-[8].

The population of individuals each of which represents the weights of the N assets in the portfolio is in reality an array (AssetWeightMat) of size (Population Size X N). LowBounds is an array of size (2 X N) where LowBounds[0, :] denotes the respective lower bounds of the asset weights in the portfolio of size N.

The function returns the lower bound standardized weights as OutputWeightMat.

In [1]:

"""

Function ensures that the asset weights satisfy their respective lower bounds

in preparation for the next stage where all the weights will be standardized to

satisfy the budget and bounds constraints imposed on the portfolio

Reference: Section 4.5, Chapter 4 Metaheuristic Risk-Budgeted Portfolio Optimization of

[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

---------------------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** SatisfyLowBounds(AssetWeightMat, LowBounds, AssetsIndx):

*#dependencies*

**import** numpy **as** np

WeightMat **=** AssetWeightMat**.**copy()

[Row, Col]**=** np**.**shape(WeightMat)

Length **=** len(AssetsIndx)

OutputWeightMat **=** np**.**zeros([Row, Col], dtype **=** float)

**for** j **in** range(Length):

jIndx **=** AssetsIndx[j]

**for** i **in** range(Row):

**if** (WeightMat[i,jIndx] **<** LowBounds[0,jIndx]):

WeightMat[i,jIndx] **=** LowBounds[0,jIndx]

OutputWeightMat **=** WeightMat

**return** OutputWeightMat

### 8.2 Function WeightStdzn

This function receives the population of individuals standardized by the function SatisfyLowBounds. This population of individuals is now repaired to satisfy the leveraged bounds and budget constraint described by equations [5]-[8]. The output array StdWeightMat of this function represents a feasible solution set to the transformed mathematical model.

In [4]:

"""

Weight Repair Strategy

-----------------------------------------------------------------------------

Pre-requisite: The input population of individuals to this function

must have undergone Lower bounds pre-processing initiated by the function SatisfyLowBounds.

Standardization of weights of each individual in the population is undertaken so that

they satisfy the leveraged bounds constraints imposed on long positions and the respective

bounds constraints imposed on short positions, subject to the fully invested

constraint of sum of weights equals 1.

Reference: Sec. 4.5 Chapter 4 Metaheuristic Risk-Budgeted Portfolio Optimization of

[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

---------------------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** WeightStdzn(AssetsWeightMat, LowBounds, AssetsIndx):

*#dependencies*

**import** numpy **as** np

WeightMat **=** AssetsWeightMat**.**copy()

[Row, Col]**=**np**.**shape(WeightMat)

StdWeightMat **=** np**.**zeros(shape **=**(Row, Col))

**for** i **in** range(Row):

SumWgts **=** np**.**sum(WeightMat[i,:])

**if** (SumWgts **>**1.0):

WeightMat[i,:]**=** StdznExcessWgts(WeightMat[i,:], LowBounds, AssetsIndx)

**elif** (SumWgts **<**1.0):

IncrAmt **=** (1.0**-**SumWgts)**/**Col

WeightMat[i,:]**=** WeightMat[i,:] **+** IncrAmt

**else**:

**continue**

StdWeightMat **=** WeightMat

**return** StdWeightMat

### 8.2.1 Function StdznExcessWgts

This sub function called by WeightStdzn function performs the subtask of standardizing the weights when the sum of weights of the population individuals exceeds the portfolio budget constraint described by equation [5].

In [3]:

"""

This function is a subfunction of WeightStdzn and undertakes standardization of weights

for those individuals that violate the portfolio budget constraint

described by equation [5].

---------------------------------------------------------------------------------------

@author: Dr G Vijayalakshmi Pai

"""

**def** StdznExcessWgts(AssetWghtVector, LowUpBounds, AssetsIndx):

*#dependencies*

**import** numpy **as** np

WeightVector **=** AssetWghtVector**.**copy()

Col **=** len(WeightVector)

StdzWeightVector **=** np**.**zeros(shape **=**(Col))

*# find excess weights and shear off equal portions of the excess from*

*# each of the weights*

SumWgts **=** np**.**sum(WeightVector)

DecrAmt **=** (SumWgts**-**1.0)**/**Col

WeightVector**=** WeightVector**-**DecrAmt

R **=** []

Length **=** len(AssetsIndx)

**for** j **in** range(Length):

jIndx**=** AssetsIndx[j]

**if** (WeightVector[jIndx]**<** LowUpBounds[0,jIndx]):

WeightVector[jIndx]**=** LowUpBounds[0,jIndx]

R**.**append(j)

*#Q and R work to adjust excess weights so that all weights*

*#satisfy their stipulated lower bounds and the sum of weights*

*#equals 1*

Flag **=** **True**

**while** (Flag **==** **True**):

Flag **=** **False**

Q **=** list(set(range(Col))**-**set(R))

L **=** np**.**sum(WeightVector)

F **=** L **-** 1.0

LengthQ**=** len(Q)

DecrAmt**=**F**/**LengthQ

**for** j **in** range(LengthQ):

WeightVector[Q[j]] **=** WeightVector[Q[j]]**-**DecrAmt

**if** (Q[j] **in** AssetsIndx):

**if** (WeightVector[Q[j]] **<** LowUpBounds[0, Q[j]]):

WeightVector[Q[j]] **=** LowUpBounds[0,Q[j]]

Flag **=** **True**

R**.**append(Q[j])

*# Any shortfall in the sum of weights equalling 1 is set right*

*#by leveraging the weights, which is permissible in the model*

SumWgts **=** np**.**sum(WeightVector)

**if**(SumWgts **<** 1.0):

IncrAmt **=** (1.0**-**SumWgts)**/**Col

WeightVector **=** WeightVector**+**IncrAmt

StdzWeightVector **=** WeightVector

**return** StdzWeightVector

### 8.3 Function ConstrViolnFunctionRiskBudgeting

This function computes the constraint violation function defined by the system of equations in [11] and returns $\psi\left ( \bar{W}, \bar{m}, t\right )$ and $G\_l$ as output (optional).

In [2]:

"""

Constraint violation function for the Risk-Budgeted portfolio optimization model described by

system of equations in [11].

Reference: Chapter 4 Metaheuristic Risk-Budgeted Portfolio Optimization of [PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

---------------------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** ConstrViolnFunctionRiskBudgeting( StandardWeightMat, CovarMat,RiskBudget, C\_param, beta\_param, alpha\_param, GenerationCount):

*#dependencies*

**import** numpy **as** np

WeightMat **=** StandardWeightMat**.**copy()

[RowMat, ColMat]**=** np**.**shape(WeightMat)

GTerm **=** np**.**zeros(shape **=**(RowMat, ColMat))

psi **=** np**.**zeros(shape **=** (RowMat))

**for** i **in** range(RowMat):

*#select each individual from the population*

xIndividual **=** np**.**zeros(shape **=**(ColMat))

xIndividual **=** WeightMat[i,:]

*#compute portfolio risk*

PortfolioRisk **=** np**.**sqrt(np**.**matmul( np**.**matmul(xIndividual, CovarMat), xIndividual**.**T))

*#compute Marginal Contribution to Risk*

mcr **=** np**.**zeros(shape **=**(ColMat))

mcr **=** np**.**matmul(CovarMat, xIndividual**.**T)**/**PortfolioRisk

*#compute function phi in equation [11]*

phi **=** np**.**zeros(shape **=**(ColMat))

phi **=** np**.**multiply(xIndividual, mcr**.**T)**-** ((RiskBudget**/**100.0)**\*** PortfolioRisk)

*#compute penalties GTerm*

**for** j **in** range(ColMat):

**if** (phi[j] **<=** 0 ):

GTerm[i, j]**=**0

**else**:

GTerm[i, j]**=**1

*#compute penalty term*

PenaltyTerm **=** np**.**power(C\_param **\*** GenerationCount, alpha\_param)

*#compute constraint violation function psi*

psi[i] **=** PenaltyTerm **\*** np**.**sum( np**.**multiply(GTerm[i,:], np**.**power(phi, beta\_param)))

**return** GTerm, psi

### 8.4 Function ComputeFitnessRiskBudgeting

This function computes the fitness function values of the population using the penalized objective function described by equation [10].

In [11]:

"""

Compute fitness function values for the population of individuals using the

penalized objective function defined by equation [10]

Reference: Chapter 4 Metaheuristic Risk-Budgeted Portfolio Optimization of [PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

---------------------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** ComputeFitnessFuncRiskBudgeting(PoplnMat, ReturnData, CovarianceData, riskfree, PsiFunc):

*#dependencies*

**import** numpy **as** np

[PoplnSize, Col] **=** np**.**shape(PoplnMat)

weight **=** np**.**zeros(shape **=** (Col), dtype **=** float)

PoplnFitness **=** np**.**zeros(PoplnSize)

**for** i **in** range(PoplnSize):

weight **=** PoplnMat[i,:]

PoplnFitness[i] **=** **-**((np**.**matmul(ReturnData, weight**.**T)**-**riskfree)**/** (np**.**sqrt(np**.**matmul( np**.**matmul(weight, CovarianceData), weight**.**T)) ) )**+** PsiFunc[i]

**return** PoplnFitness

### 8.5 Function DE\_Mutation

This function implements the standard Mutation operator of Differential Evolution strategy, which helps generate trial vectors.

In [5]:

"""

Differential Evolution's Mutation operator to generate trial vector population

--------------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** DE\_Mutation(IndividualPopln, beta\_val, PoplnSize):

*#dependencies*

**import** numpy **as** np

**import** array **as** arr

DifferentialVecIndex **=** [0]**\***2

[Row, Col]**=** np**.**shape(IndividualPopln)

TrialVecPopln **=** np**.**zeros(shape **=**(Row, Col))

**for** i **in** range(PoplnSize):

*#select target vector and two difference vectors randomly*

RandomIndex **=** arr**.**array('i', np**.**random**.**permutation(PoplnSize))

RandomIndex**.**remove(i)

TrialVecIndex **=** int(RandomIndex[0])

DifferentialVecIndex[0] **=** int(RandomIndex[1])

DifferentialVecIndex[1] **=** int(RandomIndex[2])

*# obtain trial vectors for each of the parent vectors*

TrialVecPopln[i,:] **=** IndividualPopln[TrialVecIndex,:] **+** beta\_val**\***(IndividualPopln[DifferentialVecIndex[0],:]**-**IndividualPopln[DifferentialVecIndex[1],:])

**return** TrialVecPopln

### 8.6 Function DEOperatorBinCrossOver

This function implements the standard Binomial Crossover operator of Differential Evolution.

In [12]:

"""

Differential Evolution Binomial Cross over operator

----------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** DEOperatorBinCrossover (ParentPopln, TargetVecPopln, ProbabilityRecombn, Components):

*#dependencies*

**import** numpy **as** np

[row, col] **=** np**.**shape(ParentPopln)

tau **=** DEComputeTau (Components, ProbabilityRecombn)

OffspringPopln **=** np**.**empty(shape **=**(row, col), dtype **=** float)

**for** i **in** range(col):

**if** i **in** tau:

OffspringPopln[:, i]**=** TargetVecPopln[:,i]

**else**:

OffspringPopln[:, i] **=** ParentPopln[:,i]

**return** OffspringPopln

### 8.6.1 Function DEComputeTau

This sub function computes a parameter (Tau) of the standard Binomial Crossover operator of Differential Evolution algorithm.

In [13]:

"""

This function computes Tau, a parameter required by Differential Evolution's Binomial Crossover operator

--------------------------------------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** DEComputeTau(GeneSize, ProbabilityRecombn):

*#dependencies*

**import** numpy **as** np

**import** random

h **=** list(np**.**random**.**permutation(range(GeneSize)))

*#set jStar to a random index*

jStar **=** h[0]

tau**=**[jStar]

h**.**remove(jStar)

**for** i **in** range(GeneSize**-**1):

**if** (random**.**random()**<** ProbabilityRecombn):

tau**.**append(h[i])

**return** tau

### 8.7 Function DEOperatorPenaltySelection

This function implements the standard Deterministic Selection operator of Differential Evolution, which selects the best fit amongst the parent and offspring population and prepares the population of individuals for the next generation.

In [14]:

"""

Deterministic selection operator of Differential Evolution that selects the best fit

individuals but modified to carry forward their respective penalty function values

when moving to the next generation

-------------------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** DEOperatorPenaltySelection(feas\_parent, Gterm, psip, feas\_parent\_fitness, offsprng, GOterm,psio, offsprng\_fitness, popln\_size):

*#dependencies*

**import** numpy **as** np

[row, col] **=** np**.**shape(feas\_parent)

*#initialization*

NextGenPool **=** np**.**zeros(shape **=** (popln\_size, col), dtype **=** float)

NextGenPoolFitness **=** np**.**zeros(shape **=** (popln\_size), dtype **=** float)

NextGenPoolPsi **=** np**.**zeros(shape **=** (popln\_size), dtype **=** float)

penalty **=** np**.**zeros(shape **=** (popln\_size, col), dtype **=** float)

**for** i **in** range(popln\_size):

**if** (feas\_parent\_fitness[i] **<=** offsprng\_fitness[i]):

NextGenPool[i,:]**=** feas\_parent[i,:]

penalty[i,:]**=** Gterm[i,:]

NextGenPoolPsi[i]**=** psip[i]

NextGenPoolFitness[i] **=** feas\_parent\_fitness[i]

**else**:

NextGenPool[i,:]**=** offsprng[i,:]

penalty[i,:]**=** GOterm[i,:]

NextGenPoolPsi[i] **=** psio[i]

NextGenPoolFitness[i] **=** offsprng\_fitness[i]

**return**[NextGenPool, NextGenPoolFitness, penalty, NextGenPoolPsi]

### 8.8 Main program for DE HOF based Risk Budgeting

The main Python program for DE HOF is shown below. A concise and clear Process Flow Chart of DE HOF constructing the optimal Risk-Budgeted portfolio can be found in Chapter 4 Metaheuristic Risk-Budgeted Portfolio Optimization in [PAI, 2018].

In [15]:

"""

Main Program

Risk-Budgeted Portfolio Construction using Differential Evoution with

Hall of Fame Metaheuristic strategy

---------------------------------------------------------------------------------------

Dataset: S&P BSE200 (April 03, 2009 - July 03, 2020), Bombay Stock Exchange, India

Input Data File: S&PBSE200\_kPortfolioParams.csv

The asset labels of the k-portfolio followed by the mean returns of the assets and

variance-covariance matrix of returns of the assetsare available in the input csv file.

---------------------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**import** numpy **as** np

**import** pandas **as** pd

**import** array **as** arr

**import** csv

**import** random

portfolioSize **=** 30

rows **=** 32

stockParamsFileName **=** 'S&PBSE200\_kPortfolioParams.csv'

df **=** pd**.**read\_csv(stockParamsFileName, nrows**=** rows)

*# extract asset labels*

assetLabels **=** df**.**columns**.**tolist()[0:portfolioSize]

print(assetLabels)

*# extract mean returns and variance-covariance matrix of returns*

stockParamsData **=** np**.**array(df**.**iloc[0:, 0:])

meanReturns**=** np**.**array(stockParamsData[0,:])

covarianceMatrix **=** np**.**array(stockParamsData[1:portfolioSize**+**1, :])

*#risk limit in percentage*

risk\_budget **=** 12.5

*#risk free rate of return in India (6.5%)*

annriskfree **=** 6.5**/**100

riskfree **=** (np**.**power((1**+**annriskfree),(1.0**/**360)) **-**1.0)**\***100

*# general bounds*

defaultLowBound**=** **-**3.0

defaultUpBound **=** 3.0

bounds **=** np**.**vstack( (np**.**repeat(defaultLowBound, portfolioSize), np**.**repeat(defaultUpBound, portfolioSize)))

*# indices of categories of assets in the portfolio*

assetIndices **=** range(0,30)

levrgLongPositionsMandatoryInvest **=** [0,1,2,3,5, 7,8,9, 11, 12, 14,15, 17,18, 21, 23, 27,28,29]

levrgLongPositionsOptionalInvest**=** [4,6,13,16,19, 22, 24,25,26]

levrgShortPositionsOptionalInvest **=** [10,20]

*# lower bounds for the categories of assets in the portfolio*

lowBoundLongPositionsMandatoryInvest**=** 0.001

lowBoundLongPositionsOptionalInvest **=** 0

lowBoundShortPositionsOptionalInvest **=** **-**3

*# Joines and Houck's (1994) dynamic penalty function strategy for constraint handling*

*# Set parameters C, alpha, beta*

C\_dp **=** 0.5

beta\_dp**=**2

alpha\_dp**=**2

*#set Differential Evolution strategy control parameters*

poplnSize **=** 400

totalGenerations **=** 500

beta **=** 0.5

probabilityRecombination **=** 0.87

individualLength **=** portfolioSize

*#set lower and upper bounds for the individual assets in the portfolio*

bounds **=** np**.**zeros(shape **=** (2, portfolioSize), dtype **=** float)

bounds[0,levrgLongPositionsMandatoryInvest]**=** np**.**repeat(lowBoundLongPositionsMandatoryInvest,len(levrgLongPositionsMandatoryInvest) )

bounds[0,levrgLongPositionsOptionalInvest]**=** np**.**repeat(lowBoundLongPositionsOptionalInvest, len(levrgLongPositionsOptionalInvest))

bounds[0,levrgShortPositionsOptionalInvest]**=** np**.**repeat(lowBoundShortPositionsOptionalInvest, len(levrgShortPositionsOptionalInvest))

bounds[1,levrgLongPositionsMandatoryInvest] **=** np**.**repeat(defaultUpBound,len(levrgLongPositionsMandatoryInvest) )

bounds[1,levrgLongPositionsOptionalInvest] **=** np**.**repeat(defaultUpBound, len(levrgLongPositionsOptionalInvest))

bounds[1,levrgShortPositionsOptionalInvest]**=** np**.**repeat(0, len(levrgShortPositionsOptionalInvest))

*#initialize Hall of Fame which will ultimately hold the optimal weights*

HOFFitness **=** 9999

HOFIndividual **=** np**.**zeros(shape**=**(1, portfolioSize))

*#initialize arrays that track the generation in which individuals*

*#are inducted into HOF and their respective fitness values*

HOFGenerationArray **=** np**.**zeros(totalGenerations)

HOFFitnessArray **=** np**.**zeros(totalGenerations)

*#initialize generation cycle and HOF tracking counters*

generationCount **=** 1

i1 **=**0

*#generate initial random population of individuals and obtain the parent population*

initialPoplnRandom **=** np**.**zeros(shape **=**(poplnSize, individualLength), dtype **=** float)

initialPoplnRandom **=** np**.**random**.**uniform(low **=** defaultLowBound, high **=** defaultUpBound, size **=**(poplnSize, individualLength))

initialPoplnUpgradedLowBounds **=** np**.**zeros(shape **=**(poplnSize, individualLength), dtype **=** float)

initialPoplnUpgradedLowBounds **=** SatisfyLowBounds(initialPoplnRandom, bounds,assetIndices)

initialPoplnFeasible **=** np**.**zeros(shape **=**(poplnSize, individualLength), dtype **=** float)

initialPoplnFeasible **=** WeightStdzn(initialPoplnUpgradedLowBounds, bounds, assetIndices)

GValuesInitialPopln **=** np**.**zeros(shape **=**(poplnSize, individualLength), dtype **=** float)

psiInitialPopln **=** np**.**zeros(shape **=**(poplnSize), dtype **=** float)

GValuesInitialPopln, psiInitialPopln **=** ConstrViolnFunctionRiskBudgeting(initialPoplnFeasible, covarianceMatrix, risk\_budget, C\_dp, beta\_dp, alpha\_dp, generationCount )

initialPoplnFeasibleFitness **=** np**.**zeros(shape **=**(poplnSize), dtype **=** float)

initialPoplnFeasibleFitness **=** ComputeFitnessFuncRiskBudgeting(initialPoplnFeasible, meanReturns, covarianceMatrix, riskfree, psiInitialPopln)

*#begin generation cycles*

**while** (generationCount **<=** totalGenerations):

print('Generation', generationCount)

*# set parent population, fitness and constraint violation function values of the population*

parentPoplnFeasible **=** np**.**zeros(shape **=**(poplnSize, individualLength), dtype **=** float)

parentPoplnFeasible **=** initialPoplnFeasible

parentPoplnFeasibleFitness **=** np**.**zeros(shape **=**(poplnSize), dtype **=** float)

parentPoplnFeasibleFitness **=** initialPoplnFeasibleFitness

GValuesParent **=** np**.**zeros(shape **=**(poplnSize, individualLength), dtype **=** float)

GValuesParent **=** GValuesInitialPopln

psiParent **=** np**.**zeros(shape **=**(poplnSize), dtype **=** float)

psiParent **=** psiInitialPopln

*#perform mutation to generate trial vectors*

trialVectorPopln **=** np**.**zeros(shape **=**(poplnSize, individualLength), dtype **=** float)

trialVectorPopln **=** DE\_Mutation(parentPoplnFeasible, beta, poplnSize)

*#generate offspring population using the parent and trial vector populations*

offsprngPoplnRandom **=** np**.**zeros(shape **=**(poplnSize, individualLength), dtype **=** float)

offsprngPoplnRandom **=** DEOperatorBinCrossover(parentPoplnFeasible, trialVectorPopln, probabilityRecombination, individualLength)

*# undertake standardization of offspring population weights to generate feasible solution set*

offsprngPoplnUpgradedLowBounds **=** np**.**zeros(shape **=**(poplnSize, individualLength), dtype **=** float)

offsprngPoplnUpgradedLowBounds **=** SatisfyLowBounds(offsprngPoplnRandom, bounds, assetIndices)

offsprngPoplnFeasible **=** np**.**zeros(shape **=**(poplnSize, individualLength), dtype **=** float)

offsprngPoplnFeasible **=** WeightStdzn(offsprngPoplnUpgradedLowBounds, bounds, assetIndices)

*# compute constraint violation function values for the offspring population*

GOffspring, psiOffsprng **=** ConstrViolnFunctionRiskBudgeting( offsprngPoplnFeasible, covarianceMatrix, risk\_budget, C\_dp, beta\_dp, alpha\_dp, generationCount)

*# compute fitness of the offspring population*

offsprngPoplnFeasibleFitness **=** ComputeFitnessFuncRiskBudgeting(offsprngPoplnFeasible,meanReturns, covarianceMatrix, riskfree, psiOffsprng)

*# Construct the population for the next generation*

nextGeneration **=** np**.**zeros(shape **=**(poplnSize, individualLength), dtype **=** float)

nextGenerationFitness **=** np**.**zeros(shape **=**(poplnSize), dtype **=** float)

[nextGeneration, nextGenerationFitness, Penalty, PsiFunctionValues] **=** DEOperatorPenaltySelection(parentPoplnFeasible, GValuesParent, psiParent, parentPoplnFeasibleFitness, offsprngPoplnFeasible, GOffspring, psiOffsprng, offsprngPoplnFeasibleFitness, poplnSize)

*# induct best individual with zero penalty values, into Hall of Fame*

**for** i **in** range(poplnSize):

**if** (np**.**sum(Penalty[i,:]) **==** 0):

**if** (nextGenerationFitness[i]**<** HOFFitness):

HOFFitness **=** nextGenerationFitness[i]

HOFIndividual **=** nextGeneration[i,:]

HOFGenerationArray[i1] **=** generationCount

HOFFitnessArray[i1] **=** HOFFitness

print('Penalty', Penalty[i,:])

print('HOF induction! Generation:', generationCount)

print('HOF fitness', HOFFitness)

i1**=**i1**+**1

**else**:

**continue**

generationCount **=** generationCount **+** 1

*# move to the next generation and get the parent population ready*

initialPoplnFeasible **=** np**.**zeros(shape **=**(poplnSize, individualLength), dtype **=** float)

initialPoplnFeasible **=** nextGeneration

initialPoplnFeasibleFitness **=** np**.**zeros(shape **=**(poplnSize), dtype **=** float)

initialPoplnFeasibleFitness**=** nextGenerationFitness

GValuesInitialPopln **=** np**.**zeros(shape **=**(poplnSize, individualLength), dtype **=** float)

GValuesInitialPopln **=** Penalty

psiInitialPopln **=** np**.**zeros(shape **=**(poplnSize), dtype **=** float)

psiInitialPopln**=** PsiFunctionValues

*# end of a generation cycle*

*#extract optimal weights from the individual in the Hall of Fame*

optimalWeights **=** HOFIndividual

*# compute risk budgeted portfolio characteristics: Sharpe Ratio, return and risk*

portfolioReturn **=** np**.**sum(np**.**multiply(meanReturns, optimalWeights**.**T))

portfolioRisk **=** np**.**sqrt(np**.**matmul( np**.**matmul(optimalWeights, covarianceMatrix), optimalWeights**.**T))

annPortfolioReturn **=** 261 **\*** portfolioReturn

annPortfolioRisk **=** np**.**sqrt(261) **\*** portfolioRisk

SharpeRatio **=** (annPortfolioReturn**-**annriskfree**\***100)**/** annPortfolioRisk

print('Sharpe Ratio', SharpeRatio)

print('Risk, Return', annPortfolioRisk, annPortfolioReturn)

print('Optimal Weights', HOFIndividual)

*#Check points for verification of constraints*

*#--------------------------------------------------*

print('Check Points!')

*#check budget constraint of portfolio*

print('Sum of optimal weights', np**.**sum(HOFIndividual))

*#compute marginal contribution to risk*

mcr **=** np**.**zeros(shape **=** (portfolioSize, 1))

mcr **=** np**.**matmul(covarianceMatrix, optimalWeights**.**T)**/**portfolioRisk

*#compute absolute contribution to risk*

abscr **=** np**.**zeros(shape **=**(portfolioSize, 1))

abscr **=** np**.**multiply(mcr, optimalWeights**.**T)

*#compute percentage contribution to risk*

pcr **=** np**.**zeros(shape **=**(portfolioSize,1))

pcr **=** (abscr**/**portfolioRisk)**\***100.0

print('percentage contribution to risk', pcr)

*#check if the optimal risk budgeted portfolio satisfies its risk budget constraint*

mcr\_status **=** np**.**zeros(shape **=** (portfolioSize, 1))

mcr\_status **=** np**.**multiply(optimalWeights, mcr**.**T)**-**((risk\_budget**/**100) **\*** portfolioRisk)

print('Risk Budgeting constraint check: To be less than 0', mcr\_status)

print('Successful execution!')

['ADANIENT', 'ASTRAL', 'BAJFINANCE', 'BHARATFORG', 'BPCL', 'CHOLAFIN', 'DABUR', 'GAIL', 'HDFCBANK', 'HINDPETRO', 'IDEA', 'INDUSINDBK', 'IOC', 'ITC', 'JSWSTEEL', 'JUBILANT', 'LT', 'M&M', 'MINDTREE', 'NESTLEIND', 'NMDC', 'PETRONET', 'RECLTD', 'RELAXO', 'SBIN', 'SUNPHARMA', 'TATAMOTORS', 'TCS', 'WHIRLPOOL', 'WIPRO']

('Generation', 1)

('Generation', 2)

('Generation', 3)

('Generation', 4)

('Generation', 5)

('Generation', 6)

('Generation', 7)

('Generation', 8)

('Generation', 9)

('Generation', 10)

('Generation', 11)

('Generation', 12)

('Generation', 13)

('Generation', 14)

('Generation', 15)

('Generation', 16)

('Generation', 17)

('Generation', 18)

('Generation', 19)

('Generation', 20)

('Generation', 21)

('Generation', 22)

('Generation', 23)

('Generation', 24)

('Generation', 25)

('Generation', 26)

('Generation', 27)

('Generation', 28)

('Generation', 29)

('Generation', 30)

('Generation', 31)

('Generation', 32)

('Generation', 33)

('Generation', 34)

('Generation', 35)

('Generation', 36)

('Generation', 37)

('Generation', 38)

('Generation', 39)

('Generation', 40)

('Generation', 41)

('Generation', 42)

('Generation', 43)

('Generation', 44)

('Generation', 45)

('Generation', 46)

('Generation', 47)

('Generation', 48)

('Generation', 49)

('Generation', 50)

('Generation', 51)

('Generation', 52)

('Generation', 53)

('Generation', 54)

('Generation', 55)

('Generation', 56)

('Generation', 57)

('Generation', 58)

('Generation', 59)

('Generation', 60)

('Generation', 61)

('Generation', 62)

('Generation', 63)

('Generation', 64)

('Generation', 65)

('Generation', 66)

('Generation', 67)

('Generation', 68)

('Generation', 69)

('Generation', 70)

('Generation', 71)

('Generation', 72)

('Generation', 73)

('Generation', 74)

('Generation', 75)

('Generation', 76)

('Generation', 77)

('Generation', 78)

('Generation', 79)

('Generation', 80)

('Generation', 81)

('Generation', 82)

('Generation', 83)

('Generation', 84)

('Generation', 85)

('Generation', 86)

('Generation', 87)

('Generation', 88)

('Generation', 89)

('Generation', 90)

('Generation', 91)

('Generation', 92)

('Generation', 93)

('Generation', 94)

('Generation', 95)

('Generation', 96)

('Generation', 97)

('Generation', 98)

('Generation', 99)

('Generation', 100)

('Generation', 101)

('Generation', 102)

('Generation', 103)

('Generation', 104)

('Generation', 105)

('Generation', 106)

('Generation', 107)

('Generation', 108)

('Generation', 109)

('Generation', 110)

('Generation', 111)

('Generation', 112)

('Generation', 113)

('Generation', 114)

('Generation', 115)

('Generation', 116)

('Generation', 117)

('Generation', 118)

('Generation', 119)

('Generation', 120)

('Generation', 121)

('Generation', 122)

('Generation', 123)

('Generation', 124)

('Generation', 125)

('Generation', 126)

('Generation', 127)

('Generation', 128)

('Generation', 129)

('Generation', 130)

('Generation', 131)

('Generation', 132)

('Generation', 133)

('Generation', 134)

('Generation', 135)

('Generation', 136)

('Generation', 137)

('Generation', 138)

('Generation', 139)

('Generation', 140)

('Generation', 141)

('Generation', 142)

('Generation', 143)

('Generation', 144)

('Generation', 145)

('Generation', 146)

('Generation', 147)

('Generation', 148)

('Generation', 149)

('Generation', 150)

('Generation', 151)

('Generation', 152)

('Generation', 153)

('Generation', 154)

('Generation', 155)

('Generation', 156)

('Generation', 157)

('Generation', 158)

('Generation', 159)

('Generation', 160)

('Generation', 161)

('Generation', 162)

('Generation', 163)

('Generation', 164)

('Generation', 165)

('Generation', 166)

('Generation', 167)

('Generation', 168)

('Generation', 169)

('Generation', 170)

('Generation', 171)

('Generation', 172)

('Generation', 173)

('Generation', 174)

('Generation', 175)

('Generation', 176)

('Generation', 177)

('Generation', 178)

('Generation', 179)

('Generation', 180)

('Generation', 181)

('Generation', 182)

('Generation', 183)

('Generation', 184)

('Generation', 185)

('Generation', 186)

('Generation', 187)

('Generation', 188)

('Generation', 189)

('Generation', 190)

('Generation', 191)

('Generation', 192)

('Generation', 193)

('Generation', 194)

('Generation', 195)

('Generation', 196)

('Generation', 197)

('Generation', 198)

('Generation', 199)

('Generation', 200)

('Generation', 201)

('Generation', 202)

('Generation', 203)

('Generation', 204)

('Generation', 205)

('Generation', 206)

('Generation', 207)

('Generation', 208)

('Generation', 209)

('Generation', 210)

('Generation', 211)

('Generation', 212)

('Generation', 213)

('Generation', 214)

('Generation', 215)

('Generation', 216)

('Generation', 217)

('Generation', 218)

('Generation', 219)

('Generation', 220)

('Generation', 221)

('Generation', 222)

('Generation', 223)

('Generation', 224)

('Generation', 225)

('Generation', 226)

('Generation', 227)

('Generation', 228)

('Generation', 229)

('Generation', 230)

('Generation', 231)

('Generation', 232)

('Generation', 233)

('Generation', 234)

('Generation', 235)

('Generation', 236)

('Generation', 237)

('Generation', 238)

('Generation', 239)

('Generation', 240)

('Generation', 241)

('Generation', 242)

('Generation', 243)

('Generation', 244)

('Generation', 245)

('Generation', 246)

('Generation', 247)

('Generation', 248)

('Generation', 249)

('Generation', 250)

('Generation', 251)

('Generation', 252)

('Generation', 253)

('Generation', 254)

('Generation', 255)

('Generation', 256)

('Generation', 257)

('Generation', 258)

('Generation', 259)

('Generation', 260)

('Generation', 261)

('Generation', 262)

('Generation', 263)

('Generation', 264)

('Generation', 265)

('Generation', 266)

('Generation', 267)

('Generation', 268)

('Generation', 269)

('Generation', 270)

('Generation', 271)

('Generation', 272)

('Generation', 273)

('Generation', 274)

('Generation', 275)

('Generation', 276)

('Generation', 277)

('Generation', 278)

('Generation', 279)

('Generation', 280)

('Generation', 281)

('Generation', 282)

('Generation', 283)

('Generation', 284)

('Generation', 285)

('Generation', 286)

('Generation', 287)

('Generation', 288)

('Generation', 289)

('Generation', 290)

('Generation', 291)

('Generation', 292)

('Generation', 293)

('Generation', 294)

('Generation', 295)

('Generation', 296)

('Generation', 297)

('Generation', 298)

('Generation', 299)

('Generation', 300)

('Generation', 301)

('Generation', 302)

('Generation', 303)

('Generation', 304)

('Generation', 305)

('Generation', 306)

('Generation', 307)

('Generation', 308)

('Generation', 309)

('Generation', 310)

('Generation', 311)

('Generation', 312)

('Generation', 313)

('Generation', 314)

('Generation', 315)

('Generation', 316)

('Generation', 317)

('Generation', 318)

('Generation', 319)

('Generation', 320)

('Generation', 321)

('Generation', 322)

('Generation', 323)

('Generation', 324)

('Generation', 325)

('Generation', 326)

('Generation', 327)

('Generation', 328)

('Generation', 329)

('Generation', 330)

('Generation', 331)

('Generation', 332)

('Generation', 333)

('Generation', 334)

('Generation', 335)

('Generation', 336)

('Generation', 337)

('Generation', 338)

('Generation', 339)

('Generation', 340)

('Generation', 341)

('Generation', 342)

('Generation', 343)

('Generation', 344)

('Generation', 345)

('Generation', 346)

('Generation', 347)

('Generation', 348)

('Generation', 349)

('Generation', 350)

('Generation', 351)

('Generation', 352)

('Generation', 353)

('Generation', 354)

('Generation', 355)

('Generation', 356)

('Generation', 357)

('Generation', 358)

('Generation', 359)

('Generation', 360)

('Generation', 361)

('Generation', 362)

('Generation', 363)

('Generation', 364)

('Generation', 365)

('Generation', 366)

('Penalty', array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]))

('HOF induction! Generation:', 366)

('HOF fitness', -0.07948219359412217)

('Generation', 367)

('Generation', 368)

('Generation', 369)

('Generation', 370)

('Generation', 371)

('Generation', 372)

('Generation', 373)

('Generation', 374)

('Generation', 375)

('Generation', 376)

('Generation', 377)

('Generation', 378)

('Generation', 379)

('Generation', 380)

('Generation', 381)

('Generation', 382)

('Generation', 383)

('Generation', 384)

('Generation', 385)

('Generation', 386)

('Generation', 387)

('Generation', 388)

('Generation', 389)

('Generation', 390)

('Generation', 391)

('Generation', 392)

('Generation', 393)

('Generation', 394)

('Generation', 395)

('Generation', 396)

('Generation', 397)

('Generation', 398)

('Generation', 399)

('Generation', 400)

('Generation', 401)

('Generation', 402)

('Generation', 403)

('Generation', 404)

('Generation', 405)

('Generation', 406)

('Generation', 407)

('Generation', 408)

('Generation', 409)

('Generation', 410)

('Generation', 411)

('Generation', 412)

('Generation', 413)

('Generation', 414)

('Generation', 415)

('Generation', 416)

('Generation', 417)

('Generation', 418)

('Generation', 419)

('Generation', 420)

('Generation', 421)

('Generation', 422)

('Generation', 423)

('Generation', 424)

('Generation', 425)

('Generation', 426)

('Generation', 427)

('Generation', 428)

('Generation', 429)

('Generation', 430)

('Generation', 431)

('Generation', 432)

('Generation', 433)

('Generation', 434)

('Generation', 435)

('Generation', 436)

('Generation', 437)

('Generation', 438)

('Generation', 439)

('Generation', 440)

('Generation', 441)

('Generation', 442)

('Generation', 443)

('Generation', 444)

('Generation', 445)

('Generation', 446)

('Generation', 447)

('Generation', 448)

('Generation', 449)

('Generation', 450)

('Generation', 451)

('Generation', 452)

('Generation', 453)

('Generation', 454)

('Generation', 455)

('Generation', 456)

('Generation', 457)

('Generation', 458)

('Generation', 459)

('Generation', 460)

('Generation', 461)

('Generation', 462)

('Generation', 463)

('Generation', 464)

('Generation', 465)

('Generation', 466)

('Generation', 467)

('Generation', 468)

('Generation', 469)

('Generation', 470)

('Generation', 471)

('Generation', 472)

('Generation', 473)

('Generation', 474)

('Generation', 475)

('Generation', 476)

('Generation', 477)

('Generation', 478)

('Generation', 479)

('Generation', 480)

('Generation', 481)

('Generation', 482)

('Generation', 483)

('Generation', 484)

('Generation', 485)

('Generation', 486)

('Generation', 487)

('Generation', 488)

('Generation', 489)

('Generation', 490)

('Generation', 491)

('Generation', 492)

('Generation', 493)

('Generation', 494)

('Generation', 495)

('Generation', 496)

('Generation', 497)

('Generation', 498)

('Generation', 499)

('Generation', 500)

('Sharpe Ratio', 1.2330205546695807)

('Risk, Return', 37.88035394260407, 53.20725502938971)

('Optimal Weights', array([ 0.001 , 0.001 , 0.001 , 0.001 , 0. ,

0.001 , 0.30454293, 0.001 , 0.001 , 0.05294095,

-0.26949028, 0.15586655, 0.05933541, 0.06097948, 0.001 ,

0.001 , 0. , 0.001 , 0.2473201 , 0. ,

-0.4685154 , 0.001 , 0.04730137, 0.001 , 0.10930836,

0. , 0.24799973, 0.16507081, 0.27433999, 0.001 ]))

Check Points!

('Sum of optimal weights', 1.0)

('percentage contribution to risk', array([ 0.0268034 , 0.01533056, 0.02254861, 0.08062262, 0. ,

0.02705263, 7.14466935, 0.15468202, 0.07251771, 10.71827281,

10.33461404, 5.61194609, 11.45764974, 2.61589726, 0.48215049,

0.01625797, 0. , 0.09478088, 8.3550133 , 0. ,

8.86172831, 0.06862245, 1.26988847, 0.0147864 , 2.90559575,

0. , 11.23859062, 6.86374192, 11.44149411, 0.10474247]))

('Risk Budgeting constraint check: To be less than 0', array([-0.29246341, -0.29273242, -0.29256318, -0.29120149, -0.29309188,

-0.29245757, -0.12556831, -0.289465 , -0.29139153, -0.04177678,

-0.05077256, -0.16150661, -0.02444035, -0.23175602, -0.28178673,

-0.29271067, -0.29309188, -0.29086952, -0.09718896, -0.29309188,

-0.08530783, -0.29148287, -0.26331636, -0.29274518, -0.22496336,

-0.29309188, -0.02957671, -0.13215532, -0.02481916, -0.29063595]))

Successful execution!

## 9. Risk-Budgeted Portfolio - Analysis of Results

Typical of metaheuristic strategies, DE HOF yields multiple optimal solutions during its various runs.

It needs to be emphasized that population based metaheuristic algorithms display stochastic behavior, since a random population of individuals explore large search spaces and evolve over generations into optimal solutions. Thus metaheuristics, to phrase it differently, are endowed with inherent capabilities to deliver multiple solutions that are either optimal or acceptable or near-optimal, to problems that traditional methods have found difficult to solve even. This characteristic of metaheuristic algorithms needs to be viewed meritoriously and exploited to one's own advantage.

Thus in the case of multiple optimal portfolio allocations that are attainable, a discerning investor can now make a judicious choice of that optimal allocation that best suits his or her investment sentiments, risk appetites etc.

Let us have a peek at two optimal portfolios, termed Portfolio A and Portfolio B for ease of reference, delivered by DE HOF during two of its runs. They are seemingly "near-optimal" but given the fact that the asset allocation, risk, expected return and Sharpe Ratios are known prior, it is left to the investor to make an appropriate choice.

### 9.1 Portfolio A

Maximal Sharpe Ratio: 2.28  
Annualized Risk(%): 21.5  
Expected Portfolio Annualized Return(%): 55.6 Leveraged allocation on Long Positions: 129% Leveraged allocation on Short Positions: 29%

Fig. 2(a) and Fig. 2(b) illustrate the asset allocation of the optimal risk budgeted long-short portfolio. Fig. 2(c) illustrates the percentage contribution to risk of the portfolio.

It can be observed that Portfolio A reports a high Sharpe Ratio with low risk.

#### (a) Portfolio A : Long Positions

Chart, bar chart

Description automatically generated

#### (b) Portfolio A: Short Positions

Chart

Description automatically generated

#### (b) Portfolio A: Percentage Contribution to Risk (risk limit 12.5%)

#### Fig. 2 Optimal Risk-Budgeted and Leveraged Long Short Portfolio invested in S&P BSE200 index (Portfolio A) obtained by DE HOF

### 9.2 Portfolio B

Maximal Sharpe Ratio: 1.02  
Annualized Risk(%): 68.24  
Expected Portfolio Annualized Return(%): 76.22 Leveraged allocation on Long Positions: 209% Leveraged allocation on Short Positions: 109%

Fig. 3(a) and Fig. 3(b) illustrate the asset allocation of the optimal risk budgeted long-short portfolio. Fig. 3(c) illustrates the percentage contribution to risk of the portfolio.

It can be observed that Portfolio B reports a decently good Sharpe Ratio, but with high risk and high returns.

Chart

Description automatically generated

#### (a) Portfolio B : Long Positions

Chart, bar chart

Description automatically generated

#### (b) Portfolio B: Short Positions

Chart

Description automatically generated

#### (c) Portfolio B: Percentage Contribution to Risk (risk limit 12.5%)

#### Fig. 3 Optimal Risk-Budgeted and Leveraged Long Short Portfolio invested in S&P BSE200 index (Portfolio B) obtained by DE HOF

## 10. Conclusion

High returns are always accompaned by high risk. Nevertheless, Risk-Budgeting is an investment approach which, while playing with the fire of risk, ensures that it is adequately protected to reap the desired returns. Thus risk budgeting strategies ensure investment exposure and market protection at the same time.

The risk budgeting approach was demonstrated over an all equity portfolio that is leveraged and long-short. Risk-budgeting can also be employed over leveraged long-short portfolios with multiple asset classes such as equities, bonds, commodities and currencies. A demonstration of risk-budgeting over one such global portfolio with multiple asset classes can be found in Chapter 4 Metaheuristic Risk-Budgeted Portfolio Optimization in [PAI 2018] and [PAI 2011a]. Risk budgeting approach has also been applied over equity market neutral portfolios, future portfolios and other long-short portfolios [PAI 2012][PAI 2015] and [PAI 2011b].

The metaheuristic strategy is inherently endowed with the potential to deliver multiple near-optimal risk-budgeted portfolios, providing investors the luxury of making choices that follow their investment sentiments.

## Companion Reading

[1] Chapter 4 Metaheuristic Risk-Budgeted Portfolio Optimization [PAI 2018]

[2] Chapter 2 A Brief Primer on Metaheuristics [PAI 2018]

[3] MATLAB Demonstration of Risk-Budgeted Portfolio Optimization using DE HOF, in Mathworks Central File Exchange <https://in.mathworks.com/matlabcentral/fileexchange/64507-metaheuristic-portfolio-optimization-models>

[4] Sharpe Ratio based Portfolio Optimization <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson6_SharpeRatioOptimization/Lesson6_MainContent.ipynb>

[5] Heuristic Portfolio Selection <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson3_HeuristicPortfolioSelection/Lesson3_MainContent.ipynb>

[6] Fundamentals of Risk and Return of a Portfolio <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson1_FundaRiskReturnPortfolio/Lesson1_MainContent.ipynb>

## References

[JOI 1994] Joines J A and C R Houck, "On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs", Proceedings of the First IEEE Conference on Evolutionary Computation, pp.579-584, 1994.

[PAI 2018] Vijayalakshmi Pai G. A., "Metaheuristics for Portfolio Optimization- An Introduction using MATLAB", Wiley-ISTE, 2018. <https://www.mathworks.com/academia/books/metaheuristics-for-portfolio-optimization-pai.html>

[PAI 2015] Vijayalakshmi G A and Thierry Michel, "Metaheuristic Construction of Long-Only Risk Budgeted Futures Portfolio with Maximal Diversification Index", Available at <http://ssrn.com/abstract=2622980> (Working paper Series), June 25, 2015.

[PAI 2012] Vijayalakshmi Pai G A and Thierry Michel, "Differential Evolution based Optimization of Risk Budgeted Equity Market Neutral Portfolios", Proc. IEEE World Congress on Computational Intelligence (IEEE WCCI 2012), 2012 IEEE Congress on Evolutionary Computation, pp. 1888-1895, Brisbane, Australia, June 2012.

[PAI 2011a] Vijayalakshmi Pai G A and Thierry Michel, "Metaheuristic Optimization of Risk Budgeted Global Asset Allocation Portfolios", Proc. World Congress on Information and Communication Technologies (WICT 2011), pp. 154-159, Mumbai, India, Dec. 2011.

[PAI 2011b] Vijayalakshmi Pai G A and Thierry Michel, "Evolutionary optimization of Risk Budgeted Long-Short Portfolios", Proc. IEEE Symposium Series in Computational Intelligence (IEEE SSCI 2011): 2011 IEEE symposium on Computational Intelligence for Financial Engineering and Economics (CIFEr 2011),pp. 59- 66, Paris, France, April 2011.

# Sharpe Ratio based Portfolio Optimization

Text

Description automatically generated with medium confidence

## 6.1 Introduction

Sharpe Ratio, developed by Nobel Laureate William F Sharpe [SHA 66], is a measure of calculating risk adjusted return. It serves to help investors know about the returns on their investments relative to the risks they hold. The Sharpe Ratio is defined as



##### ..........(6.1)

where $r\_x$ is the average rate of return on the investment $x$, $R\_f$, the best available risk free rate of return and $\sigma$ the standard deviation of $r\_x$, which denotes the risk on the investment.

Higher the Sharpe Ratio, more is the excess returns over that of holding a risk free investment, relative to the increased volatility that the investment is exposed to. A Sharpe Ratio of 0, needless to say, only denotes the investment to be risk-free or one that does not yield any excess return. In practice, while a Sharpe ratio of 1 marks the investment to be acceptable or good for investors, a value less than 1 grades the investment as sub-optimal, and values greater than 1 and moving towards 2 or 3, grades the investment as highly superior.

## 6.2 Maximizing Sharpe Ratio

Having understood the significance of the Sharpe Ratio, let us suppose an investor wishes to make an investment in assets in such a way that the Sharpe Ratio of the portfolio would be the best possible or the maximum, that can be ensured for the investment.

Let P be a portfolio comprising assets $A\_1, A\_2, ...A\_N$, with $\mu\_1, \mu\_2, ...\mu\_N$ as the asset returns and $W\_1, W\_2, ...W\_N$ as the weights.

The portfolio return $r$ determined by a weighted summation of its individual asset returns is given by, $\sum\left({W\_i.\mu\_i}\right)$ and the risk is given by $\sqrt{\sum\sum {W\_i.W\_j.\sigma\_{ij}} } $. (See Lesson 1 Fundamentals of Risk and Return of a Portfolio to know about risk and return of a portfolio).

To keep the discussion simple for now, let us suppose that the investor decides to enforce only basic constraints on the portfolio. (See Sec. 5.2 of Lesson 5 Mean Variance Optimization of Portfolios to know about basic constraints).

The mathematical model for the Sharpe Ratio based Portfolio Optimization is given by,

A picture containing shape

Description automatically generated

##### ..........(6.2)

The numerator of the objective function denotes the excess returns of the investment over that of a risk free asset $R\_f$ and the denominator the risk of the investment. The objective is to maximize the Sharpe Ratio. The basic constraints indicate that the investor wishes to have a fully invested portfolio.

## 6.3 Solving the Sharpe Ratio Optimization Model

To solve the Sharpe Ratio maximization model represented by (6.2), we make use of the minimize library function from scipy.optimize package of Python. However, since the original objective function insists on maximization as opposed to minimization demanded by the minimize solver, the principal of duality borrowed from Optimization Theory is employed to undertake the transformation. According to the principle,



##### ..........(6.3)

The Python code for the function MaximizeSharpeRatioOptimization which defines the objective function and the basic constraints represented by (6.2), is shown below:

In [21]:

*#function to undertake Sharpe Ratio maximization subject to*

*#basic constraints of the portfolio*

*#dependencies*

**import** numpy **as** np

**from** scipy **import** optimize

**def** MaximizeSharpeRatioOptmzn(MeanReturns, CovarReturns, RiskFreeRate, PortfolioSize):

*# define maximization of Sharpe Ratio using principle of duality*

**def** f(x, MeanReturns, CovarReturns, RiskFreeRate, PortfolioSize):

funcDenomr **=** np**.**sqrt(np**.**matmul(np**.**matmul(x, CovarReturns), x**.**T) )

funcNumer **=** np**.**matmul(np**.**array(MeanReturns),x**.**T)**-**RiskFreeRate

func **=** **-**(funcNumer **/** funcDenomr)

**return** func

*#define equality constraint representing fully invested portfolio*

**def** constraintEq(x):

A**=**np**.**ones(x**.**shape)

b**=**1

constraintVal **=** np**.**matmul(A,x**.**T)**-**b

**return** constraintVal

*#define bounds and other parameters*

xinit**=**np**.**repeat(0.33, PortfolioSize)

cons **=** ({'type': 'eq', 'fun':constraintEq})

lb **=** 0

ub **=** 1

bnds **=** tuple([(lb,ub) **for** x **in** xinit])

*#invoke minimize solver*

opt **=** optimize**.**minimize (f, x0 **=** xinit, args **=** (MeanReturns, CovarReturns,\

RiskFreeRate, PortfolioSize), method **=** 'SLSQP', \

bounds **=** bnds, constraints **=** cons, tol **=** 10**\*\*-**3)

**return** opt

The Sharpe Ratio optimization requires the computation of risk and return of the portfolio. The asset returns computing function StockReturnsComputing, is reproduced here for the reader's convenience.

In [22]:

*# function computes asset returns*

**def** StockReturnsComputing(StockPrice, Rows, Columns):

**import** numpy **as** np

StockReturn **=** np**.**zeros([Rows**-**1, Columns])

**for** j **in** range(Columns): *# j: Assets*

**for** i **in** range(Rows**-**1): *# i: Daily Prices*

StockReturn[i,j]**=**((StockPrice[i**+**1, j]**-**StockPrice[i,j])**/**StockPrice[i,j])**\*** 100

**return** StockReturn

## 6.4 Case Study

Let us suppose that an investor decides to invest in a $k$-portfolio ($k$-portfolio 1) comprising the following Dow stocks. ($k$ portfolio 1, is detailed and listed in Lesson 3 Heuristic Portfolio Selection)

𝑘-portfolio 1:

{Coca-Cola (KO), United Health (UNH), Walt Disney (DIS), IBM (IBM), Cisco (CSCO), JPMorgan Chase (JPM), Goldman Sachs (GS), Walgreens Boots Alliance (WBA), Apple (AAPL), Home Depot (HD), American Express (AXP), McDonald's (MCD), Merck (MRK), Boeing (BA), Caterpillar (CAT)}

The investor desires to explore the optimal portfolio set that would yield the maximal Sharpe Ratio. The objective is to find out the optimal weights that will ensure maximal Sharpe Ratio for the portfolio.

The following Python code reads the dataset concerned, computes the stock returns using the Python function StockReturnsComputing and obtains the mean returns and the variance-covariance matrix of returns.

In [23]:

*#obtain mean and variance-covariance matrix of returns for k-portfolio 1*

*#Dependencies*

**import** numpy **as** np

**import** pandas **as** pd

*#input k portfolio 1 dataset comprising 15 stocks*

StockFileName **=** 'DJIA\_Apr112014\_Apr112019\_kpf1.csv'

Rows **=** 1259 *#excluding header*

Columns **=** 15 *#excluding date*

*#read stock prices*

df **=** pd**.**read\_csv(StockFileName, nrows**=** Rows)

*#extract asset labels*

assetLabels **=** df**.**columns[1:Columns**+**1]**.**tolist()

print('Asset labels of k-portfolio 1: \n', assetLabels)

*#read asset prices data*

StockData **=** df**.**iloc[0:, 1:]

*#compute asset returns*

arStockPrices **=** np**.**asarray(StockData)

[Rows, Cols]**=**arStockPrices**.**shape

arReturns **=** StockReturnsComputing(arStockPrices, Rows, Cols)

*#set precision for printing results*

np**.**set\_printoptions(precision**=**3, suppress **=** **True**)

*#compute mean returns and variance covariance matrix of returns*

meanReturns **=** np**.**mean(arReturns, axis **=** 0)

covReturns **=** np**.**cov(arReturns, rowvar**=False**)

print('\nMean Returns:\n', meanReturns)

print('\nVariance-Covariance Matrix of Returns:\n', covReturns)

Asset labels of k-portfolio 1:

['AAPL', 'AXP', 'BA', 'CAT', 'CSCO', 'DIS', 'GS', 'HD', 'IBM', 'JPM', 'KO', 'MCD', 'MRK', 'UNH', 'WBA']

Mean Returns:

[ 0.09 0.029 0.1 0.039 0.081 0.04 0.033 0.085 -0.016 0.06

0.019 0.057 0.036 0.095 -0.002]

Variance-Covariance Matrix of Returns:

[[2.375 0.672 0.962 1.042 0.999 0.68 0.954 0.726 0.709 0.825 0.306 0.458

0.534 0.774 0.697]

[0.672 1.648 0.8 0.95 0.7 0.569 1.065 0.658 0.663 1.001 0.307 0.35

0.556 0.718 0.667]

[0.962 0.8 2.288 1.31 0.89 0.716 1.066 0.747 0.777 0.977 0.381 0.472

0.578 0.745 0.679]

[1.042 0.95 1.31 2.733 1.041 0.688 1.321 0.796 0.885 1.169 0.358 0.455

0.616 0.72 0.681]

[0.999 0.7 0.89 1.041 1.789 0.713 0.927 0.724 0.817 0.909 0.362 0.477

0.647 0.656 0.707]

[0.68 0.569 0.716 0.688 0.713 1.35 0.773 0.586 0.574 0.717 0.302 0.368

0.466 0.557 0.631]

[0.954 1.065 1.066 1.321 0.927 0.773 2.114 0.795 0.803 1.554 0.303 0.467

0.705 0.82 0.819]

[0.726 0.658 0.747 0.796 0.724 0.586 0.795 1.39 0.619 0.753 0.343 0.472

0.487 0.659 0.689]

[0.709 0.663 0.777 0.885 0.817 0.574 0.803 0.619 1.632 0.767 0.372 0.391

0.576 0.564 0.534]

[0.825 1.001 0.977 1.169 0.909 0.717 1.554 0.753 0.767 1.702 0.324 0.483

0.675 0.761 0.717]

[0.306 0.307 0.381 0.358 0.362 0.302 0.303 0.343 0.372 0.324 0.806 0.36

0.384 0.31 0.355]

[0.458 0.35 0.472 0.455 0.477 0.368 0.467 0.472 0.391 0.483 0.36 1.086

0.402 0.43 0.433]

[0.534 0.556 0.578 0.616 0.647 0.466 0.705 0.487 0.576 0.675 0.384 0.402

1.504 0.615 0.64 ]

[0.774 0.718 0.745 0.72 0.656 0.557 0.82 0.659 0.564 0.761 0.31 0.43

0.615 1.722 0.78 ]

[0.697 0.667 0.679 0.681 0.707 0.631 0.819 0.689 0.534 0.717 0.355 0.433

0.64 0.78 2.554]]

The annual average risk free rate of return in USA during April 2019 was 3%. The daily risk free rate is computed as



##### ..........(6.4)

The following Python code computes the maximal Sharpe Ratio for $k$-portfolio 1.

In [24]:

*#obtain maximal Sharpe Ratio for k-portfolio 1 of Dow stocks*

*#set portfolio size*

portfolioSize **=** Columns

*#set risk free asset rate of return*

Rf**=**3 *# April 2019 average risk free rate of return in USA approx 3%*

annRiskFreeRate **=** Rf**/**100

*#compute daily risk free rate in percentage*

r0 **=** (np**.**power((1 **+** annRiskFreeRate), (1.0 **/** 360.0)) **-** 1.0) **\*** 100

print('\nRisk free rate (daily %): ', end**=**"")

print ("{0:.3f}"**.**format(r0))

*#initialization*

xOptimal **=**[]

minRiskPoint **=** []

expPortfolioReturnPoint **=**[]

maxSharpeRatio **=** 0

*#compute maximal Sharpe Ratio and optimal weights*

result **=** MaximizeSharpeRatioOptmzn(meanReturns, covReturns, r0, portfolioSize)

xOptimal**.**append(result**.**x)

*#compute risk returns and max Sharpe Ratio of the optimal portfolio*

xOptimalArray **=** np**.**array(xOptimal)

Risk **=** np**.**matmul((np**.**matmul(xOptimalArray,covReturns)), np**.**transpose(xOptimalArray))

expReturn **=** np**.**matmul(np**.**array(meanReturns),xOptimalArray**.**T)

annRisk **=** np**.**sqrt(Risk**\***251)

annRet **=** 251**\***np**.**array(expReturn)

maxSharpeRatio **=** (annRet**-**Rf)**/**annRisk

*#set precision for printing results*

np**.**set\_printoptions(precision**=**3, suppress **=** **True**)

*#display results*

print('Maximal Sharpe Ratio: ', maxSharpeRatio, '\nAnnualized Risk (%): ', \

annRisk, '\nAnnualized Expected Portfolio Return(%): ', annRet)

print('\nOptimal weights (%):\n', xOptimalArray**.**T**\***100 )

Risk free rate (daily %): 0.008

Maximal Sharpe Ratio: [[1.26]]

Annualized Risk (%): [[14.749]]

Annualized Expected Portfolio Return(%): [21.584]

Optimal weights (%):

[[13.694]

[ 0. ]

[17.744]

[ 0. ]

[12.151]

[ 0. ]

[ 0. ]

[19.058]

[ 0. ]

[ 1.151]

[ 0. ]

[13.654]

[ 0. ]

[22.547]

[ 0. ]]

The output shows that the maximal Sharpe Ratio attainable for $k$-portfolio 1 is 1.26 which is good, going by practical standards. The annual expected portfolio return is 21.584% against an annualized risk of 14.749%. To achieve this, the optimal capital allocations on the assets of $k$-portfolio 1 are as follows:

['AAPL': 13. 694%] , ['BA': 17.744%], ['CSCO': 12.151%], ['HD': 19.058%], ['JPM': 1.151%], ['MCD': 13.654%], ['UNH': 22.547%].

No investments need be made in the rest of the assets of $k$-portfolio 1 since the optimal weights arrived at for these assets are 0.  
However, if the investor desires to invest in all the assets in the $k$-portfolio 1, with the weights distributed across all the assets in the portfolio, all that the investor needs to do is to redefine the bounds constraint of (6.2) as, $0\lt W\_i\lt 1$ and run the scipy solver over the optimization model.

Despite the wide use of Sharpe Ratios to compute risk adjusted returns, it is not without its disadvantages. Thus, if the expected returns do not follow a normal distribution or if the portfolios possess non-linear risks, to list a few of the instances, Sharpe Ratios may not deliver results. Therefore, alternative methodologies such as Sortino Ratio and Treynor Ratio have emerged as contenders to Sharpe Ratios.

### Companion Reading

This blog is an abridged adaptation of concepts discussed in Chapter 1 and Chapter 3 of [PAI 18] to Dow Jones dataset (DJIA index: April, 2014- April, 2019) and implemented in Python. Readers (read "worker bees"), seeking more information may refer to the corresponding chapters in the book.

### References

[SHA 66] Sharpe, William F. Mutual Fund Performance, Journal of Business, January 1966, pp. 119-138.

[PAI 18] Vijayalakshmi Pai G. A., Metaheuristics for Portfolio Optimization- An Introduction using MATLAB, Wiley-ISTE, 2018. <https://www.mathworks.com/academia/books/metaheuristics-for-portfolio-optimization-pai.html>

# Portfolio Rebalancing using Metaheuristics

Graphical user interface, text

Description automatically generated

## 1. What is Portfolio Rebalancing?

Asset allocation which concerns itself with the optimal allocation of capital to be invested in assets of a portfolio subject to the investor's risk-return appetite and investment goals, is a key factor in determining the long time investment performance. However, it is not a one-time deal since market forces tend to fluctuate and a portfolio kept untended for long can drift to a state that incurs exposure to more risk, which is not in sync with the investor's risk appetite and investment preferences. Hence the need to rebalance a portfolio.

Rebalancing a portfolio deals with buying and selling components of the portfolio to set the portfolio weights to satisfy their original goals or to devise a new asset allocation by readjusting the weights of each asset to realign the risk-return characteristics of the drifted portfolio. (See Chapter 7 of [PAI 2018] to know more about Rebalancing of portfolios)

Despite the need, portfolio rebalancing has its costs too, since buying and selling of assets incur transaction costs besides capital gains taxes, administrative costs and/or management fees, which can eat into the returns of the rebalanced portfolio. We term these generally as Transaction Costs. If the transaction costs can be accommodated within the gains made by the rebalanced portfolio leaving a respectable return, then such a portfolio is termed as a self-financing portfolio.

The objective of this work therefore, is to demonstrate an optimal portfolio rebalancing model that will (i) arrive at the optimal buy/sell decisions with regard to the assets in the portfolio (ii) ensure that the optimal rebalanced portfolio is a self financing portfolio (iii) ensures that the risk of the rebalanced portfolio does not exceed that of the original portfolio's risk (iv) The buy and sell limits are governed by the investor's preferences and (v) the rebalanced portfolio is a fully invested portfolio.

The optimal portfolio rebalancing model employs metaheuristics to arrive at the optimal buy/sell weights that will yield a rebalanced portfolio which satisfies the constraints specifed above. The objective of the model is to maximize the Sharpe Ratio of the rebalanced portfolio. Assets that need to be bought (increase in asset weights) will be referred to as BUY assets and those that need to be sold (decrease in asset weights) will be referred to as SELL assets.

## 2. Construction of Optimal Rebalanced Portfolio Model

Let us consider an investor who desires to construct a rebalanced self-financing portfolio with maximum Sharpe Ratio and subject to the constraints that reflect restrictions on asset allocation and the investor's preferences.

The objective function of the model is,

(1) To maximize the Sharpe Ratio of the rebalanced portfolio

(See <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson6_SharpeRatioOptimization/Lesson6_MainContent.ipynb> to know about Sharpe Ratio based portfolio optimization)

The constraints are,

(2) Bounds on the weights of BUY assets, where the investor specifies limits on how much can be bought maximum.

(3) Bounds on the weights of SELL assets, where the investor specifies limits on how much can be sold maximum.

(4) A budget constraint on the portfolio, where the capital is fully invested, (i.e.) the sum of the rebalanced portfolio weights equals 1.

(5) Ensure that the rebalanced portfolio is a self financing portfolio.

(6) Ensure that the rebalanced portfolio's risk does not exceed the original portfolio's risk.

## 3. Mathematical Formulation of the Portfolio Rebalancing Model

Let us suppose that an investor has invested in a portfolio $P\_{Original}$ comprising assets $A\_1, A\_2, ...A\_N$ with the weight allocation given by $\bar W\_{(N X1)}=[w\_1, w\_2, ...w\_N]'$ and let $Risk\_{Original}$ be the risk of $P\_{Original}$.

Let us also suppose that the portfolio was kept untended for long and the investor decides to rebalance the portfolio one fine day! Let $P\_{rebal}$ be the rebalanced portfolio to obtain which, the optimal buy/sell weights need to be determined. Let $\bar X\_{(N X1)}=[x\_1, x\_2, ...x\_N]'$ be the desired optimal allocation of weights of the rebalanced portfolio. Let $x\_{i}^{+}, i= 1, 2, ...N$ and $x\_{i}^{-}, i= 1, 2, ...N $ be the proportion of weights that need to be bought or sold of the BUY assets and SELL assets and hence need to be added or subtracted from $\bar W$ of $P\_{original}$, respectively.

For computational convenience we assume that the buy/sell decision with regard to asset i is characterized by a mutually exclusive pair $[x\_i^+, x\_i^-]$ where $x\_i^- =0$ if it is a buy decision and $x\_i^+ = 0$ if it is a sell decision.

Let p be the transaction cost rate and $u\_i$ be the upper limit of the buy weights. $u\_i$ acts as a ceiling to curtail the transaction costs, which can disrupt realizing a self financing portfolio that the investor expects.

Let $\bar \mu\_{(1 X N)} = [\mu\_1, \mu\_2, ...\mu\_N]$ be the asset returns for the historical data set considered for portfolio rebalancing and V be the variance-covariance matrix of asset returns for the said period. It is known that the portfolio return is given by ${\bar{\mu }}.{\bar{X}}$ and the portfolio risk is given by ${\sqrt{\bar{X}'.V.\bar{X}}} $ (See <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson1_FundaRiskReturnPortfolio/Lesson1_MainContent.ipynb>, to know about risk and return of a portfolio)

Let $R\_f$ be the risk free rate of return. The Sharpe Ratio is given by $\frac{ {\bar{\mu }}.{\bar{X}}-R\_{f} }{\sqrt{\bar{X}'.V.\bar{X}}} $.

The portfolio rebalancing model is defined as,

Text

Description automatically generated

The objective of the portfolio rebalancing model defined by equations (1)-(7) is to find the optimal $\bar{X}$ which is dependent on the optimal buy/sell weights $x\_i^+$ and $x\_i^-$.

## 4. Metaheuristic Optimization

We proceed to undertake the optimization of the model described by equations (1) to (7) employing a metaheuristic strategy viz., Evolution Strategy with Hall of Fame (ES HOF). ES HOF arrives at the optimal weights of the rebalanced portfolio that yield maximal Sharpe Ratio, subject to all the constraints enforced on the assets and the portfolio.

(See Chapter 2: A Brief Primer on Metaheuristics [PAI 2018], to know more about metaheuristics)

Metaheuristic strategies build feasible solution sets that satisfy the constraints imposed, during the course of their execution process. However constraint handling has been a major problem in the application of metaheuristic strategies to complex constrained optimization problems, to tackle which several methods such as Penalty Function Strategy and Repair Strategy have been proposed.

For the Portfolio Rebalancing model discussed in this work, ES HOF makes use of Joines and Houck's Dynamic Penalty Function Strategy [JOI 1994] to handle the constraint that imposes a ceiling on the rebalanced portfolio risk (equation (7)) and Repair Strategies to handle the bounds constraints on the buy/sell weights, the budget constraint on the rebalanced portfolio as well the self financing portfolio constraints (equations (2) to (6)).

Repair strategies are custom made to suit the requirements of the problem and evolving such a strategy that will help satisfy one or more constraints at one go, can turn out to be difficult. Nevertheless, once the strategy is evolved, it can help churn out populations of feasible solution sets during each of the generation cycles, leading to faster convergence of the metaheuristic strategy.

(See Sec. 7.3.3 of Chapter 7: Metaheuristic Portfolio Rebalancing with Transaction Costs [PAI 2018], to know more about the Repair Strategies evolved for the Portfolio Rebalancing model)

## 5. Transformation of the Mathematical Model

Joines and Houck's dynamic penalty function strategy is used to tackle the constraint represented by equation (7). The constraint is accomodated in the "penalized" objective function by defining appropriate penalty functions. The transformed mathematical model is shown below.

Text, letter

Description automatically generated

In the system of equations (8), $(C, \alpha, \beta)$ are all constants and the penalty term $(C.t)^\alpha$ increases constantly with each generation count t of the metaheuristic strategy ES HOF. $\epsilon$ is a tolerance limit for testing the inequality constraint represented by $\phi$.

The transformed portfolio rebalancing model with equation (8) as its objective function and equations (2-6) as its constraints is solved using the metaheuristic strategy ES HOF.

## 6. Evolution Strategy with Hall of fame, a run-through!

ES HOF is a population based metaheuristic strategy that evokes elitism using Hall of Fame (See Section 2.4 of Chapter2 of [PAI 2018] to know more about ES HOF).

The inputs, process and output of ES HOF are described below:

### Inputs

It is essential that the portfolio rebalancing parameters and the ES HOF strategy parameters are clearly set and input, before the optimization process begins.

The rebalancing portfolio parameters are (1) assets in the untended portfolio (2) mean asset returns and covariance of returns (the variance-covariance matrix of returns) for the historical period up till the day on which rebalancing is executed (3) bounds for the buy/sell weights (4)transaction cost rate and (5) risk free rate of return. The weights of the original portfolio $\bar{W}$ and its risk $Risk^{Original}$ are also input.

The ES HOF parameters are (1) population size (2) number of generations (3) dynamic penalty function parameters $(C, \alpha, \beta)$ (4) crossover rate and (5) mutation rate.

### Process

ES HOF begins its execution by generating an initial random population of chromosomes/individuals that represent random buy/sell weights $x\_i^+ / x\_i^-$ for the assets in the portfolio.

In the first stage, the population is normalized to enable each of the randomly generated buy/sell weights to satisfy the respective bounds described by equations (2)-(3).

In the second stage, a weight repair strategy is evolved so that the normalized population of chromosomes satisfies all the constraints described by equations (2)-(6), enabling it to transform itself into a feasible solution set. Let us call this population P.

Using the rebalanced weights $\bar{X}$ represented by population P, fitness function values are computed employing the penalized objective function described by equation (8).

Set population P to be the parent population.

Obtain its offspring population by applying the genetic inheritance operators of arithmetic variable point cross over and real number uniform mutation. Normalize the offspring population to enable them satisfy their respective bounds for the buy/sell weights and transform them into a feasible solution set by using the weight repair strategy. Call the population O.

Obtain the rebalanced weights for the population O of chromosomes and compute its fitness values using equation (8).

Select the best fit individuals of the population P and O in the ratio of $\mu:\lambda$ . Call the population NEXTGEN.

Select the best fit among NEXTGEN and induct it into the Hall of Fame after allowing it to compete with the individual already in it.

Set NEXTGEN as the parent population P for the next generation and repeat the generation cycle.

Once the termination criterion (the number of generations for instance) is met with, extract the optimal buy/sell weights from the individual in the Hall of Fame. Compute the rebalanced weights using equation (4).

### Output

It can be verified that the optimal buy/sell weights and the rebalanced weights satisfy all the constraints imposed on them, besides delivering a solution with the maximal Sharpe Ratio. Given the mean returns and the covariance of returns, the risk and return of the optimal Rebalanced portfolio can be easily computed. It can be verified that the Rebalanced portfolio is fully invested, its risk does not exceed that of the original portfolio risk and is a self-financing portfolio.

(See Sec. 7.3 of Chapter 7, Metaheuristic Portfolio Rebalancing with Transaction Costs [PAI 2018] to know more about the design, process flow chart and execution of Evolution Strategy with Hall of Fame for the portfolio rebalancing model)

## 7. Case Study

We proceed to demonstrate the portfolio rebalancing model over an equity portfolio invested in S&P BSE200 (Bombay Stock Exchange, India) markets. The historical data set for the period (April 2009- Jan 2021) encompassing all phases of the Covid 19 crisis period in India, has been considered for the demonstration of the model.

### 7.1 Investing in the portfolio

To keep the narrative short, we assume that the investor has already made a technically diverse choice of assets in the portfolio (a **k**-portfolio, in fact) and had invested in it on April 02, 2019. The historical data set for the period April 01, 2009 to April 01, 2019 was considered to compute the mean returns $\mu$ and the variance-covariance matrix of returns V.

Note that a k-portfolio is an outcome of a heuristic portfolio selection strategy, where the universe of stocks is grouped into clusters that display intra-class similarity and inter-class dissimilarity with regard to the mean-returns and covariance of returns. Since assets belonging to a cluster are similar in behavior, the investor now makes a choice of one asset each from each cluster to ensure diversification of assets in the portfolio. A clustering technique such as k-means algorithm can be used to group the stock universe into k clusters with the investor exercising the choice of k.

(See <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson3_HeuristicPortfolioSelection/Lesson3_MainContent.ipynb> and Chapter 3 Heuristic Portfolio Selection [PAI 2018], to know more about the construction of k-portfolios and their merits)

The k-portfolio selected by the investor comprised the following 30 assets, after making a heuristic portfolio selection for k = 30:

3MIndia Ltd. ["'3MINDIA'"], Ashok Leyland Ltd.["'ASHOKLEY'"], Bajaj Finance Ltd. ["'BAJFINANCE'"], Bharat Forge Ltd. ["'BHARATFORG'"], GAIL (India) Ltd. ["'GAIL'"], GMR Infra Ltd. ["'GMRINFRA'"], HDFC Bank Ltd.["'HDFCBANK'"], Hindustan Petroleum Corporation Ltd. [ "'HINDPETRO'"], Indian Oil Corporation Ltd.[ "'IOC'"], ITC Ltd. ["'ITC'"], Jindal Steel Ltd. ["'JINDALSTEEL'"], JSW Steel Ltd.[ "'JSWSTEEL'"], Jubilant Life Sciences Ltd.["'JUBILANT'"], Larsen and Toubro Ltd. ["'LT'"], Mahindra and Mahindra Ltd., ["'M&M'"], MindTree Ltd.["'MINDTREE'"], NMDC LTD. ["'NMDC'"], Power Finance Corporation Ltd. ["'PFC'"], Rajesh Exports Ltd. ["'RAJESHEXPO'"], Relaxo FootWears Ltd. ["'RELAXO'"], Reliance Industries Ltd. ["'RELIANCE'"], State Bank of India Ltd. ["'SBIN'"], Shree Cement Ltd. ["'SHREECEM'"], Tata Steel Ltd. ["'TATASTEEL'"], Tata Consultancy Services Ltd. ["'TCS'"], UltraTech Cement Ltd. ["'ULTRACEMCO'"], Voltas Ltd. ["'VOLTAS'"], Wipro Ltd. ["'WIPRO'"]

The characteristics of the original portfolio invested in are as shown in Fig. 1. The investor had imposed a bounds constraint of investing atleast 1% of the capital in all the assets.

Chart

Description automatically generated with low confidence

#### Fig. 1 Characteristics of the original S&P BSE200 equity portfolio invested on April 02, 2019

### 7.2 Untended portfolio

The portfolio was kept untended from April 03, 2019 to May 31, 2020, which included periods of long spells of lockdown and partial lockdown in India ordered by the Government to tackle the pandemic crisis.

### 7.3 Portfolio Rebalancing

The portfolio was rebalanced on June 02, 2020.

To undertake this and implement the portfolio rebalancing model described by equations (8) and (2-6), the historical data set from April 01, 2009 to June 01, 2020 for the stocks invested in the portfolio were considered. The mean returns and the variance-covariance matrix of returns were computed. The weights of the original portfolio $\bar{W}$ and its risk $Risk^{Original}$ were also input, along with the other parameters of the portfolio as explained in Section 6 to the ES HOF model.

A fragment of the CSV file S&PBSE200\_kPortfolioRebal.csv, which describes the asset labels, mean returns and variance-covariance matrix of returns for the historical period concerned, to be used by ES HOF for the construction of optimal rebalanced portfolio is shown below:

Table

Description automatically generated

#### Fig. 2 Structure of the input CSV file, which describes the asset labels, mean returns and variance-covariance matrix of returns for the historical period concerned, to be used by ES HOF for the construction of optimal rebalanced portfolio

## 8. Python Coding of ES HOF for Portfolio Rebalancing

The ESHOF program is a conglomeration of functions typical of any metaheuristic strategy. The functions are listed first followed by the main program, with a brief description of the task accomplished by the function code.

### 8.1 Function GenerateBoundedPopln

This function generates a random initial population of buy/sell weights that lie within their respective bounds as described by equations (2-3). Buy weights are stored positive in sign and sell weights are stored negative in sign, for computational convenience.

In [4]:

"""

Generate random population of individuals representing buy/sell weights

-----------------------------------------------------------------------------

To generate an initial random population of xi+ / xi-, the buy / sell weights for

portfolio rebalancing.

The buy /sell weights satisfy their respective lower bounds of 0 (sell\_low =0, buy\_low =0)

and their respective upper bounds of (sell\_high = the respective portfolio weights) and

(buy\_high = the fixed bounds of 0.025)

Reference: Chapter 7 Metaheuristic Portfolio Rebalancing with Transaction Costs[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** GenerateBoundedPopln( popln\_rows, popln\_cols, sell\_low, sell\_high, buy\_low, buy\_high ):

**import** random

**import** numpy **as** np

popln\_raw **=** np**.**zeros((popln\_rows, popln\_cols))

*# Generate a random population of sell weights*

**for** i **in** range(popln\_cols):

a **=** sell\_low[0,i]

b **=** sell\_high[0,i]

**for** j **in** range(popln\_rows):

relement **=** random**.**uniform(a,b)

popln\_raw[j,i] **=** **-**relement

*# Randomly wipe out sell weights and replace it with random buy weights*

s **=** np**.**zeros(popln\_rows)

**for** i **in** range(popln\_cols):

c **=** buy\_low[0,i]

d **=** buy\_high[0,i]

t **=** np**.**random**.**permutation(popln\_rows)

**for** j **in** range(popln\_rows):

s[j] **=** random**.**uniform(c,d)

**for** j **in** range(popln\_rows) :

**if** random**.**random() **>** 0.5:

popln\_raw[j, i] **=** s[t[j]]

**else**:

**continue**

**return** popln\_raw

### 8.2 Function DetermineBounds

This function determines the bounds of each of the asset weights in the random population of buy/sell weights generated by Function GenerateBoundedPopln and stores it in a Python dictionary for further computational use.

In [5]:

"""

Determine the bounds of the buy /sell weights represented by the

random population of individuals generated by Function GenerateBoundedPopln

----------------------------------------------------------------------------------------

Given the population of randomly generated individuals representing the

buy and sell proportions of weights of individual assets, the function determines

the upper and lower bounds of each of the asset weights, for each of the individuals

in the population and stores it in a Python dictionary (low\_up\_limits).

Note that the buy weights are stored positive and sell weights are stored negative in the

population of individuals (popln).

Reference: Chapter 7 Metaheuristic Portfolio Rebalancing with Transaction Costs[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** DetermineBounds( popln, pos\_low\_limits, pos\_up\_limits, neg\_low\_limits, neg\_up\_limits):

**import** numpy **as** np

[row, col]**=** np**.**shape(popln)

*#store the bounds in a dictionary for use of future computations*

low\_up\_limits **=** {}

**for** i **in** range(row):

low\_up\_limits[i] **=** np**.**zeros((2, col))

**for** j **in** range(col):

**if** (popln[i,j] **>=** 0):

low\_up\_limits[i][0,j] **=** pos\_low\_limits[0,j]

low\_up\_limits[i][1,j] **=** pos\_up\_limits[0,j]

**else**:

low\_up\_limits[i][0,j] **=** neg\_low\_limits[0,j]

low\_up\_limits[i][1,j] **=** neg\_up\_limits[0,j]

**return** low\_up\_limits

### 8.3 Function PortfolioRebalancingWeightRepair

This function implements a weight repair strategy typically evolved to repair the buy/sell weight population so that each individual in the population satisfies the constraints described by equations (4-6) and thereby transforms itself into a feasible solution vector.

The buy/sell weights are redistributed following a mathematical computation derived from the constraints. If even after redistribution, the buy/sell weights violate their respective lower bounds, then the function AdjustPosNegWgtsLowbounds is called, otherwise function AdjustPosNegWgtsUpbounds is called, at the end of which the entire weight vector satisfies the constraints described by equations (4-6) and hence becomes a feasible solution set for the transformed portfolio rebalancing model discussed in Section 5.

Details about the weight repair strategy can be found in Section 7.3.3 of Chapter 7 Metaheuristic Portfolio Rebalancing with Transaction Costs [PAI, 2018].

In [6]:

"""

To obtain a feasible solution set satisfying constraints described by

equations (4-6), by redistributing buy/sell weights, xi+ and xi- respectively.

In this function, Pos (itive) and Neg (ative) weights representing buy and sell respectively, are

tackled separately during standardization and later conjoined as a single weight set.

---------------------------------------------------------------------------------------

Reference: Sec. 7.3.3, Chapter 7 Metaheuristic Portfolio Rebalancing with Transaction Costs[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** PortfolioRebalancingWeightRepair(weight\_mat, weight\_limits, trn\_cost):

**import** numpy **as** np

[row\_mat, col\_mat]**=** np**.**shape( weight\_mat)

stdz\_weight\_mat **=** np**.**empty((row\_mat, col\_mat), dtype **=**float)

*#a is a constant given by (1-p)/(1+p) where p is the proportional transaction cost*

a **=** (1**-**trn\_cost)**/**(1**+**trn\_cost)

low\_up\_bounds **=** np**.**empty((2,col\_mat), dtype **=**float)

**for** i **in** range(row\_mat):

positive\_weights **=** np**.**empty((1,col\_mat), dtype **=**float)

negative\_weights **=** np**.**empty((1,col\_mat), dtype **=**float)

low\_up\_bounds **=** weight\_limits[i]

positive\_weights**=**np**.**reshape( np**.**multiply( weight\_mat[i,:], np**.**where( weight\_mat[i,:]**>=**0, 1,0)) , (1,col\_mat)) *#buy weights*

negative\_weights**=**np**.**reshape( **-**np**.**multiply( weight\_mat[i,:], np**.**where( weight\_mat[i,:]**<**0, 1,0)), (1,col\_mat)) *#sell weights*

x\_plus **=** np**.**sum(positive\_weights)

x\_minus **=** np**.**sum(negative\_weights)

nz\_set\_size **=** np**.**count\_nonzero(positive\_weights) **+** np**.**count\_nonzero(negative\_weights)

x\_minus\_whole **=** a**\***x\_minus

**if** (x\_plus **>** x\_minus\_whole):

diff **=** x\_plus **-** x\_minus\_whole

SIGNAL **=** 1 *#sum of buy weights greater than sum of sell weights*

**else**:

**if** (x\_plus **<** x\_minus\_whole):

diff **=** x\_minus\_whole**-** x\_plus

SIGNAL **=** 0 *#sum of sell weights greater than sum of buy weights*

**else**:

**continue**

LOWLIMITFLAG**=** 1

UPLIMITFLAG **=** 1

**if** (diff **!=**0):

term **=** diff**/**nz\_set\_size *# redistribute excess weights*

**if** (SIGNAL **==** 1):

**for** j **in** range(col\_mat):

**if** (positive\_weights[0,j] **>**0):

positive\_weights[0,j] **=** positive\_weights[0,j] **-**term

**if** (negative\_weights[0,j] **>**0):

negative\_weights[0,j] **=** negative\_weights[0,j] **+** term**/**a

**else**:

**if** (SIGNAL **==** 0): *#redistribute excess weights*

**for** j **in** range(col\_mat):

**if** (negative\_weights[0,j] **>**0):

negative\_weights[0,j] **=** negative\_weights[0,j] **-**term**/**a

**if** (positive\_weights[0, j] **>**0):

positive\_weights[0,j] **=** positive\_weights[0,j] **+** term

**for** j **in** range(col\_mat):

P **=** positive\_weights[0,j]

N **=** negative\_weights[0,j]

l **=** low\_up\_bounds[0,j]

u **=** low\_up\_bounds[1,j]

**if** (P **!=** 0):

**if** ((P **>=** l) **and** (P **<=** u)):

**continue**

**else**:

LOWLIMITFLAG**=** 0

**if** (N **!=** 0):

**if** ((N **>=** l) **and** (N **<=**u)) :

**continue**

**else**:

LOWLIMITFLAG**=**0

*# Adjust weights further, when they violate their lower bounds, even after redistribution of excess weights*

**if** (LOWLIMITFLAG**==**0):

[positive\_weights,negative\_weights, UPLIMITFLAG] **=** AdjustPosNegWgtsLowbounds(positive\_weights,negative\_weights, low\_up\_bounds, SIGNAL, a, col\_mat)

*# Adjust weights further, when they violate their upper bounds, even after redistribution of excess weights*

**if** (UPLIMITFLAG **==**0):

[positive\_weights, negative\_weights]**=** AdjustPosNegWgtsUpbounds(positive\_weights, negative\_weights, low\_up\_bounds, SIGNAL, a, col\_mat)

stdz\_weight\_mat[i,:] **=** positive\_weights**-** negative\_weights

**return** stdz\_weight\_mat

#### 8.3.1 Function AdjustPosNegWgtsLowbounds

This function is called by Function PortfolioRebalancingWeightRepair only when the buy/sell weights in a chromosome violate their lower bound constraints, after an involved redistribution of excess weights undertaken by it.

In [7]:

"""

This function is called by Function PortfolioRebalancingWeightRepair to adjust the

violations in lower bounds of the respective buy/sell weights, after redistribution of weights

undertaken by the function itself.

---------------------------------------------------------------------------------------

Reference: Sec. 7.3.3, Chapter 7 Metaheuristic Portfolio Rebalancing with Transaction Costs[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** AdjustPosNegWgtsLowbounds(pos\_wgts\_input, neg\_wgts\_input, low\_up\_bounds, SIGNAL, a, portfolio\_size ):

**import** numpy **as** np

positive\_weights\_vec **=** np**.**reshape(pos\_wgts\_input, (1, portfolio\_size))**.**copy()

negative\_weights\_vec **=** np**.**reshape(neg\_wgts\_input, (1, portfolio\_size))**.**copy()

col\_vec **=** portfolio\_size

first\_weights **=** np**.**zeros((1,col\_vec), dtype **=**float)

second\_weights **=** np**.**zeros((1,col\_vec), dtype **=**float)

*# Whichever category (buy or sell) exceeds its sum of weights is*

*# first\_weights, the other is second\_weights*

**if** (SIGNAL **==**1):

first\_weights[0,:] **=** positive\_weights\_vec[0,:] *# Test and adjust positive weights first*

second\_weights[0,:]**=** negative\_weights\_vec[0,:]

**else**:

first\_weights[0,:] **=** negative\_weights\_vec[0,:] *# Test and adjust negative weights first*

second\_weights[0,:] **=** positive\_weights\_vec[0,:]

*# R: those weights that fell below their lower bound and are now adjusted*

R **=** []

DEPOSIT **=** 0

EPSILON **=** 0.0001

**for** i **in** range(col\_vec):

**if** ((first\_weights[0,i] **!=** 0) **and** (first\_weights[0,i] **<** low\_up\_bounds[0,i])):

DEFICIT **=** (low\_up\_bounds[0,i]**-**first\_weights[0,i])

DEPOSIT **=** DEPOSIT **-** DEFICIT

first\_weights[0,i] **=** low\_up\_bounds[0,i]

R**.**append(i)

*# Q: those weights which satisfy their upper bounds*

Q **=** list(set(range(col\_vec))**-**set(R))

sharable\_weights **=** np**.**count\_nonzero(first\_weights[0,Q] )

**if** (sharable\_weights **!=** 0):

redistributed\_share **=** DEPOSIT**/**sharable\_weights

t **=** len(Q)

**for** i **in** range(t):

**if** ((first\_weights[0,Q[i]]**>**0) **and** ((first\_weights[0,Q[i]] **-**abs(redistributed\_share))**>=** (low\_up\_bounds[1, Q[i]]))):

first\_weights[0,Q[i]] **=** first\_weights[0,Q[i]] **-** abs(redistributed\_share)

DEPOSIT **=** DEPOSIT**-**redistributed\_share

**if** (abs(DEPOSIT) **<=** EPSILON) :

**break**

**else**:

**continue**

**if** (abs(DEPOSIT) **>** EPSILON):

redistributed\_share **=** 0.0

nonzero\_weights **=** np**.**count\_nonzero(second\_weights[0,:])

**if** (nonzero\_weights **!=** 0):

**if** (SIGNAL **==**1):

redistributed\_share **=** DEPOSIT**/**(a**\***nonzero\_weights )

**else**:

redistributed\_share **=** (DEPOSIT**/**nonzero\_weights)

**for** i **in** range(col\_vec):

**if** (second\_weights[0,i] **!=** 0) :

second\_weights[0,i] **=** second\_weights[0,i] **+** abs(redistributed\_share)

UPBOUNDSFLAG **=** 1

**for** j **in** range (col\_vec):

**if** ((second\_weights[0,j]**!=**0) **and** (second\_weights[0,j] **>** low\_up\_bounds[1,j])) :

UPBOUNDSFLAG **=** 0

positive\_weights\_adjusted **=** np**.**zeros((1, col\_vec), dtype **=**float)

negative\_weights\_adjusted **=** np**.**zeros((1,col\_vec), dtype **=**float)

**if** (SIGNAL **==** 1) :

positive\_weights\_adjusted **=** first\_weights[0,:]

negative\_weights\_adjusted **=** second\_weights[0,:]

**else**:

positive\_weights\_adjusted **=** second\_weights[0,:]

negative\_weights\_adjusted **=** first\_weights[0,:]

**return**(positive\_weights\_adjusted, negative\_weights\_adjusted, UPBOUNDSFLAG)

#### 8.3.2 Function AdjustPosNegWgtsUpbounds

This function is called by Function PortfolioRebalancingWeightRepair only when the buy/sell weights in a chromosome violate their upper bound constraints, after an involved redistribution of weights undertaken by it.

In [8]:

"""

This function is called by Function PortfolioRebalancingWeightRepair to adjust the

violations in upper bounds of the respective buy/sell weights after redistribution of weights

undertaken by the function itself.

---------------------------------------------------------------------------------------

Reference: Sec. 7.3.3, Chapter 7 Metaheuristic Portfolio Rebalancing with Transaction Costs[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** AdjustPosNegWgtsUpbounds(pos\_wgts\_input, neg\_wgts\_input, low\_up\_bounds, SIGNAL, a, portfolio\_size):

**import** numpy **as** np

positive\_weights\_vec **=** np**.**reshape(pos\_wgts\_input, (1, portfolio\_size))**.**copy()

negative\_weights\_vec **=** np**.**reshape(neg\_wgts\_input, (1, portfolio\_size))**.**copy()

col\_vec **=** portfolio\_size

first\_weights **=** np**.**zeros((1,col\_vec), dtype **=**float)

second\_weights **=** np**.**zeros((1,col\_vec), dtype **=**float)

*# Whichever category (buy or sell) exceeds its sum of weights is*

*# first\_weights, the other is second\_weights*

**if** (SIGNAL **==**1):

first\_weights **=** positive\_weights\_vec *# Test and adjust positive weights first*

second\_weights **=** negative\_weights\_vec

**else**:

first\_weights **=** negative\_weights\_vec *# Test and adjust negative weights first*

second\_weights **=** positive\_weights\_vec

*# R: those weights that exceeded their upper bounds and are now adjusted*

R **=** []

DEPOSIT **=** 0

EPSILON **=** 0.0001

**for** i **in** range(col\_vec):

**if** ((second\_weights[0,i] **!=** 0) **and** (second\_weights[0,i] **>** low\_up\_bounds[1,i])) :

EXCESS **=** (**-**low\_up\_bounds[1,i]**+**second\_weights[0,i])

DEPOSIT **=** DEPOSIT **+** EXCESS

second\_weights[0,i] **=** low\_up\_bounds[1,i]

R**.**append(i)

*# Q: those weights which satisfy their upper bounds*

Q **=** list(set(range(col\_vec))**-**set(R))

sharable\_weights **=** np**.**count\_nonzero(second\_weights[0,Q])

**if** (sharable\_weights **!=** 0):

**if** (SIGNAL **==** 1):

redistributed\_share **=** DEPOSIT**/**(a **\*** sharable\_weights)

**else**:

redistributed\_share **=** DEPOSIT**/** sharable\_weights

t **=** len(Q)

**for** i **in** range(t):

**if** (second\_weights[0, Q[i]]**>**0) **and** ((second\_weights[0, Q[i]]**+**redistributed\_share)**<=** low\_up\_bounds[1, Q[i]]):

second\_weights[0,Q[i]] **=** second\_weights[0,Q[i]] **+** redistributed\_share

DEPOSIT **=** DEPOSIT**-**redistributed\_share

**if** (DEPOSIT**<=** EPSILON) :

**break**

**else**:

**continue**

**if** (DEPOSIT **>** EPSILON) :

actual\_total **=** sum(first\_weights[0,:])

NZfirstweights\_index **=** np**.**where(first\_weights[0,:])

total **=** actual\_total **-**(DEPOSIT**/**a)**-**np**.**sum(low\_up\_bounds[0,NZfirstweights\_index])

**for** i **in** range(col\_vec):

**if** (first\_weights[0,i] **!=** 0):

proportion **=** (first\_weights[0,i]**/**actual\_total)**\***total

first\_weights[0,i] **=** low\_up\_bounds[0,i] **+** proportion

**if** (SIGNAL **==**1):

positive\_weights\_adjusted **=** first\_weights[0,:]

negative\_weights\_adjusted **=** second\_weights [0,:]

**else**:

positive\_weights\_adjusted **=** second\_weights [0,:]

negative\_weights\_adjusted **=** first\_weights [0,:]

**return** (positive\_weights\_adjusted, negative\_weights\_adjusted)

### 8.4 Function ComputeConstraintViolation

This function computes the constraint violation function $\psi\left(\bar{X},V,Risk^{Original},t\right)$ defined in equation (8). The function works to ensure that the risk of the rebalanced portfolio does not exceed that of the original portfolio by making use of penalty functions.

In [9]:

"""

Constraint violation function for rebalanced portfolio optimization model

-------------------------------------------------------------------------

Reference: Chapter 7 Metaheuristic Portfolio Rebalancing with Transaction Costs[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** ComputeConstraintViolation( weight\_mat, covariance\_mat, original\_portfolio\_risk, C\_param, Beta\_param, Alpha\_param, generation\_count ):

**import** numpy **as** np

[row\_mat, col\_mat]**=** np**.**shape(weight\_mat)

psi **=** np**.**zeros(row\_mat)

G **=** np**.**zeros(row\_mat)

EPSILON**=**0.01

**for** i **in** range(row\_mat):

*# Select each chromosome from the population*

X\_chromosome **=** weight\_mat[i,:]

*# Compute portfolio risk for the chromosome*

portfolio\_risk **=** np**.**sqrt(np**.**matmul((np**.**matmul(X\_chromosome,covariance\_mat)), np**.**transpose(X\_chromosome) ))

*# Compute phi, the rebalanced portfolio risk constraint*

phi **=** abs(portfolio\_risk**-**original\_portfolio\_risk)**-** EPSILON

*# compute penalties G*

g **=** 1**-**(phi **<=**0)

G[i] **=** g

*# compute constraint violation function psi shown in equation (8)*

penalty\_term**=** np**.**power(C\_param**\***generation\_count, Alpha\_param)

psi[i] **=** penalty\_term **\*** g **\***np**.**power(phi, Beta\_param)

**return** (G, psi)

### 8. 5 Function ComputeFitness

This function computes the fitness function values for the population of chromosomes making use of the penalized objective function described in equation (8).

In [10]:

"""

Computing Fitness function values

--------------------------------------------------------------

Reference: Chapter 7 Metaheuristic Portfolio Rebalancing with Transaction Costs[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** ComputeFitness(popln\_mat, mean\_returns, covariance\_data, psi, risk\_free\_rate):

**import** numpy **as** np

[popln\_size, col\_size ]**=** np**.**shape(popln\_mat)

objective\_function\_value **=** np**.**zeros(popln\_size)

**for** i **in** range(popln\_size):

weight **=** popln\_mat[i,:]

*#compute risk returns and max Sharpe Ratio of the optimal portfolio*

x\_optimal\_array **=** np**.**array(weight)

risk **=** np**.**matmul((np**.**matmul(x\_optimal\_array, covariance\_data)), np**.**transpose(x\_optimal\_array))

expected\_return **=** np**.**matmul(np**.**array(mean\_returns),x\_optimal\_array**.**T)

annualized\_risk **=** np**.**sqrt(risk**\***251)

annualized\_ret **=** 251**\***np**.**array(expected\_return)

objective\_function\_value[i]**=** ((annualized\_ret**-**risk\_free\_rate)**/**annualized\_risk) **-**psi[i]

**return** objective\_function\_value

### 8.6 Function RandomVariablePointArithmeticCrossover

This function undertakes the Random Variable Point Arithmetic Crossover [OSY 2002] that works over a pair of real coded parent chromosomes to yield a pair of offspring chromosomes.

In [11]:

"""

Executes random variable point arithmetic crossover operator over a parent population

-----------------------------------------------------------------------------------

Reference: Sec. 2.4.1 of Chapter 2 A Brief Primer on Metaheuristics [PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** RandomVariablePointArithmeticCrossover(parent\_population, popln\_size, genes):

**import** numpy **as** np

**import** random

parent1 **=** np**.**empty(shape **=**(genes), dtype**=**float)

parent2 **=** np**.**empty(shape **=** (genes), dtype **=**float)

crossover\_rate **=** 0.61

random\_arrangement **=** np**.**random**.**permutation(popln\_size)

selectedparent\_population **=** parent\_population[random\_arrangement,:]

offspring\_population **=** np**.**zeros((popln\_size, genes))

**for** i **in** range(0, popln\_size**-**1, 2) :

parent1 **=** selectedparent\_population[i,:]

parent2 **=** selectedparent\_population[i**+**1,:]

a **=** random**.**uniform(0,1)

**for** j **in** range(genes):

**if** (random**.**uniform(0,1) **<** crossover\_rate) :

temp1 **=** parent1[j]

temp2 **=** parent2[j]

parent1[j]**=** a **\*** temp1 **+** (1**-**a)**\*** temp2

parent2[j]**=** (1**-**a)**\*** temp1 **+** a **\*** temp2

offspring\_population[i,:] **=** parent1

offspring\_population[i**+**1,:]**=** parent2

**return** offspring\_population

### 8.7 Function RealNumberMutation

This function undertakes real number uniform mutation [OSY 2002], a mutation operator that contributes to the diversity of the population for a specific mutation rate that is kept very small.

In [12]:

"""

Executes Real Number Uniform Mutation over a population

-----------------------------------------------------------------------------------

Reference: Sec. 2.4.1 of Chapter 2 A Brief Primer on Metaheuristics [PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** RealNumberUniformMutation(population, popln\_size, genes):

**import** random

**import** numpy **as** np

mutated\_population **=** np**.**empty(shape **=** (popln\_size, genes), dtype **=** float)

mutation\_rate **=** 0.01

gene\_lowpoint **=** 0

gene\_highpoint **=** 1

**for** i **in** range(popln\_size):

**for** j **in** range(genes):

rno **=** random**.**uniform(0,1)

**if** (rno **<** mutation\_rate): *# mutate the gene*

mutated\_population[i,j] **=** gene\_lowpoint **+** rno **\*** (gene\_highpoint**-**gene\_lowpoint)

**else**:

mutated\_population[i,j] **=** population[i,j]

**return** mutated\_population

### 8.8 Function WeightNormalization

This function normalizes the mutated population of buy/sell weights to lie within their respective bounds.

In [13]:

"""

Normalization of buy/sell weights to lie within their respective bounds

-----------------------------------------------------------------------------------

Reference: Chapter 7 Metaheuristic Portfolio Rebalancing with Transaction Costs[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** WeightNormalization(mutated\_population,sell\_low , sell\_high, buy\_low, buy\_high):

**import** numpy **as** np

[popln\_rows, popln\_cols] **=** np**.**shape(mutated\_population)

positive\_weights **=** np**.**multiply(mutated\_population, (mutated\_population **>=**0)) *# Negative weights are marked 0*

negative\_weights **=** **-**(np**.**multiply(mutated\_population, (mutated\_population **<**0)) ) *# Positive weights are marked 0*

*# normalize positive and negative weights to [0,1]*

max\_positive **=** np**.**max(positive\_weights)

max\_negative **=** np**.**max(negative\_weights)

min\_positive **=** np**.**min(positive\_weights)

min\_negative **=** np**.**min(negative\_weights)

range\_positive **=** max\_positive **-** min\_positive

range\_negative **=** max\_negative **-** min\_negative

positive\_weights01Norm **=** np**.**zeros((popln\_rows, popln\_cols))

negative\_weights01Norm **=** np**.**zeros((popln\_rows, popln\_cols))

positive\_weights\_newnorm**=** np**.**zeros((popln\_rows, popln\_cols))

negative\_weights\_newnorm **=** np**.**zeros((popln\_rows, popln\_cols))

mutated\_population\_normalized **=** np**.**zeros((popln\_rows, popln\_cols))

**for** i **in** range(popln\_rows):

**for** j **in** range(popln\_cols):

a **=** positive\_weights[i,j]

b **=** negative\_weights[i,j]

**if** (a **>**0):

positive\_weights01Norm[i,j] **=** (a**-**min\_positive)**/**range\_positive

**else**:

positive\_weights01Norm[i,j] **=**0

**if** (b **>** 0):

negative\_weights01Norm[i,j] **=** (b**-**min\_negative)**/**range\_negative

**else**:

negative\_weights01Norm[i,j] **=**0

range\_new\_positive **=** buy\_high **-** buy\_low

range\_new\_negative **=** sell\_high **-** sell\_low

**for** i **in** range(popln\_rows):

**for** j **in** range(popln\_cols):

a **=** positive\_weights01Norm[i,j]

b **=** negative\_weights01Norm[i,j]

**if** (a **>**0):

positive\_weights\_newnorm[i,j]**=** buy\_low[0,j]**+**( a**\***range\_new\_positive[0,j])

**else**:

positive\_weights\_newnorm[i,j] **=**0

**if** (b **>** 0):

negative\_weights\_newnorm[i,j] **=** sell\_low[0,j]**+**( b**\***range\_new\_negative[0,j])

**else**:

negative\_weights\_newnorm[i,j] **=**0

mutated\_population\_normalized **=** positive\_weights\_newnorm **-** negative\_weights\_newnorm

**return** mutated\_population\_normalized

### 8.9 Function ConstructNewGeneration

This function prepares the population of offspring and parent chromosomes in the ratio $\mu$: $\lambda$, for the next generation. The $\mu$+$\lambda$ evolution strategy ensures exploration and exploitation of search space and therefore selects the $\mu$ best fit parent chromosomes and $\lambda$ best fit offspring chromosomes for the next generation.

In [14]:

"""

Construct the population of best fit offspring and parent chromosomes for the next generation

-----------------------------------------------------------------------------------

Extracts the best individuals from the parent and offspring population using mu+lambda strategy,

for the next generation.

The best fit individuals are selected based on the fitness function values, parent\_W\_fitness

associated with the parent\_W\_popln and OffspringWFitness associated with the offspring\_W\_popln.

The corresponding individuals from parent\_X\_popln and offspring\_X\_popln are conjoined as

NextGenPoolX, to prepare for the next generation cycle.

Reference:

Chapter 7 Metaheuristic Portfolio Rebalancing with Transaction Costs[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** ConstructNewGeneration(parent\_X\_popln, parent\_W\_popln, parent\_W\_fitness, parent\_W\_psi, offspring\_X\_popln, offspring\_W\_popln, OffspringWFitness, OffspringWPsi):

**import** numpy **as** np

[row\_mat, col\_mat] **=** np**.**shape(parent\_X\_popln)

parent\_W\_fitness\_sortindex **=** (np**.**argsort(parent\_W\_fitness))[:: **-**1]

offspring\_W\_fitness\_sortindex**=** (np**.**argsort(OffspringWFitness))[:: **-**1]

*# mu+lambda strategy based construction of next generation*

mu **=** int(np**.**round((1**/**3)**\*** row\_mat**+**1))

lamb **=** row\_mat**-**mu

select\_parent\_index **=** parent\_W\_fitness\_sortindex[0:mu]

select\_offspring\_index **=** offspring\_W\_fitness\_sortindex[0:lamb]

selectparents\_X **=** parent\_X\_popln[select\_parent\_index,:]

selectparents\_W**=** parent\_W\_popln[select\_parent\_index,:]

selectparents\_psi **=** parent\_W\_psi[select\_parent\_index]

selectparents\_fitness**=** parent\_W\_fitness[select\_parent\_index]

selectoffspring\_X **=** offspring\_X\_popln[select\_offspring\_index,:]

selectoffspring\_W **=** offspring\_W\_popln[select\_offspring\_index,:]

selectoffspring\_psi **=** OffspringWPsi[select\_offspring\_index]

selectoffspring\_fitness**=** OffspringWFitness[select\_offspring\_index]

nextgen\_pool\_X**=** np**.**append(selectparents\_X, selectoffspring\_X, axis **=**0)

nextgen\_pool\_W **=** np**.**append(selectparents\_W, selectoffspring\_W, axis **=**0)

nextgen\_pool\_psi **=** np**.**append(selectparents\_psi, selectoffspring\_psi)

nextgen\_pool\_fitness **=** np**.**append(selectparents\_fitness, selectoffspring\_fitness)

**return** (nextgen\_pool\_X, nextgen\_pool\_W, nextgen\_pool\_fitness, nextgen\_pool\_psi)

### 8.10 Function ComputePMeasure

In [15]:

"""

Compute P Measure to ascertain the convergence of the evolutionary algorithm

----------------------------------------------------------------------------

References: Vitaliy Feoktistov, Differential Evolution in Search of Solutions, Springer, 2006.

Chapter 7 Metaheuristic Portfolio Rebalancing with Transaction Costs[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version:

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

-----------------------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**import** scipy.spatial.distance **as** ssd

*# compute Population measure (Vitaliy Feoktistov, 2006) to ascertain the convergence of ES HOF*

**def** ComputePMeasure (popln):

[popln\_size, components]**=** np**.**shape(popln)

popln\_centre **=** np**.**mean(popln, axis **=**0)

augmented\_popln **=** np**.**vstack([popln\_centre, popln])

Cdist\_matrix **=** ssd**.**cdist(augmented\_popln, augmented\_popln,'euclidean')

P\_measure **=** np**.**max(Cdist\_matrix[0,:])

**return** (P\_measure)

### 8.11 Main Program for ES HOF based Portfolio Rebalancing

The main Python program for ES HOF is shown below. A concise and clear Process Flow Chart of ES HOF constructing the optimal rebalanced portfolio can be found in Sec. 7.3.2 of Chapter 7 Metaheuristic Portfolio Rebalancing with Transaction Costs in [PAI, 2018].

In [16]:

"""

Main Program

Optimal Rebalanced Portfolio Construction using Evolution Strategy with Hall of Fame

------------------------------------------------------------------------------------------------------------------

Equity Market: S&P BSE200(Bombay Stock Exchange, India)

Original portfolio (k-portfolio) of 30 assets, invested on April 02, 2019;

Portfolio kept untended from April 03, 2019 to June 01, 2020;

Rebalancing done on June 02, 2020;

Historical data set for portfolio rebalancing: S&P BSE200 (April 01, 2009 to June 01, 2020).

The asset labels of the k-portfolio followed by the mean returns of the assets and variance-covariance matrix of returns

of the assets for the historical period, are available in the input csv file S&PBSE200\_K30\_RebalPeriodMeanCovRetrns.csv

------------------------------------------------------------------ ----------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**import** numpy **as** np

**import** pandas **as** pd

PortfolioSize **=** 30

*# obtain mean returns and variance-covariance matrix of returns for the historical data set*

*# of the original portfolio, until the day of rebalancing*

PortfolioParametersFileName **=** 'S&PBSE200\_K30\_RebalPeriodMeanCovRetrns.csv'

Rows **=** 32

df **=** pd**.**read\_csv(PortfolioParametersFileName, nrows**=** Rows)

*# extract asset labels*

AssetLabels **=** df**.**columns**.**tolist()[0:PortfolioSize]

print(AssetLabels)

*# extract mean returns, variance-covariance matrix of returns and asset betas*

PortfolioParams **=** np**.**array(df**.**iloc[0:, 0:])

MeanData **=** np**.**array(PortfolioParams[0,:])

CovData **=** np**.**array(PortfolioParams[1:PortfolioSize**+**1, :])

*# obtain optimal weights, maximal diversification ratio, risk and return of the original portfolio invested in*

OriginalPortfolioOptimalWeights **=** np**.**reshape(np**.**array([0.088088,0.01,0.267138,0.01,0.01, 0.01,0.01,0.01,0.026345,0.01,0.01,0.01,0.01,0.01, 0.01,0.01, 0.064164,0.01,0.01,0.017946,0.190318,0.01,0.01,0.083153,0.01,0.01,0.042249,0.01,0.01,0.01 ]),(1, PortfolioSize))

OriginalPortfolioSharpeRatio **=** 2.104436

OriginalPortfolioDailyRisk **=** 0.014238

OriginalPortfolioDailyReturn **=** 0.002046

OriginalPortfolioAnnReturn**=** 53.406

OriginalPortfolioAnnRisk **=** 23.002

*# Set Transaction cost*

p **=** 0.0044

*# Set Risk Free Rate of Indian markets*

RiskFreeRate **=** 0.05

*# Set lower and upper bounds of buy and sell weights as defined by equations (2-3)*

BuyLowBounds **=** np**.**zeros((1,PortfolioSize), dtype **=** float)

BuyUpBounds **=** np**.**ones((1,PortfolioSize), dtype **=** float)**\***0.025

SellLowBounds **=** np**.**zeros((1,PortfolioSize), dtype **=** float)

SellUpBounds **=** OriginalPortfolioOptimalWeights

*# Set control parameters of Evolution Strategy with Hall of Fame*

PoplnSize **=** 300

ChromosomeLength **=** PortfolioSize

TotalGenerations **=** 800

*# The penalized objective function is an outcome of the application of Joines and Houck's (1994) dynamic penalty function strategy*

*# Set parameters (C, alpha, beta) described in equation (8)*

C **=** 0.5

BetaParam**=**2

AlphaParam**=**2

EPSILON **=** 0.0009

*# initialize Hall of Fame*

HOFFitness **=** **-**9999.99

*# generation counter*

GenerationCount **=** 0

*# initialize index for tracing HOF fitness*

i1**=**0

*# generate random initial population of buy / sell weights (xi+, xi-) subject to their respective bounds*

SourcePopln **=** GenerateBoundedPopln(PoplnSize, PortfolioSize, SellLowBounds, SellUpBounds, BuyLowBounds, BuyUpBounds)

*# determine the respective lower and upper bounds of the genes in the population of buy/sell weights and*

*# store it in a Python dictionary for further use*

LowUpBoundsInit **=**{}

LowUpBoundsInit **=** DetermineBounds(SourcePopln, BuyLowBounds, BuyUpBounds, SellLowBounds, SellUpBounds)

*# repair buy/sell weights (xi+, xi-)so that constraints described by equations (4-6) are satisfied and*

*# a feasible solution set is obtained*

StandardizedSourcePopln **=** PortfolioRebalancingWeightRepair(SourcePopln, LowUpBoundsInit,p)

*# obtain the population of rebalanced weights, which is the initial (parent) population*

InitialPopln **=** np**.**ones((PoplnSize,1)) **\*** OriginalPortfolioOptimalWeights **+** StandardizedSourcePopln

*# compute constraint violation function values of the parent population*

[ParentG, ParentPsi] **=** ComputeConstraintViolation(InitialPopln, CovData, OriginalPortfolioDailyRisk, C, BetaParam, AlphaParam, GenerationCount) *#compute constraint violation functions using Joines and Houck's dynamic penalty functions*

*# compute fitness function values of the parent population*

ParentFitness **=** np**.**zeros(PoplnSize)

ParentFitness**=** ComputeFitness(InitialPopln, MeanData, CovData, ParentPsi, RiskFreeRate )

*# set parent population parameters for the generation cycle*

*# (the buy/sell weights and the respective rebalanced portfolio weights)*

FeasParentPoplnX **=** StandardizedSourcePopln

FeasParentPoplnW **=** InitialPopln

FeasParentPoplnWFitness **=** ParentFitness

FeasParentPoplnWPsi **=** ParentPsi

*#set counter variable for P measure*

PMeasureCount **=**0

HOFGenarray **=** np**.**zeros(TotalGenerations)

HOFFitarray **=** np**.**zeros(TotalGenerations)

PerformanceAnalysisMeasure **=** np**.**zeros((TotalGenerations, 2))

*# ES HOF generation cycles begin*

**while** (GenerationCount **<=** TotalGenerations**-**1):

print("Generation:", GenerationCount )

*# perform crossover operation on the parent population*

OffspringPoplnXSource**=** RandomVariablePointArithmeticCrossover(FeasParentPoplnX, PoplnSize, ChromosomeLength)

*# perform mutation operation to yield offspring population*

OffspringPoplnXMutated **=** RealNumberUniformMutation(OffspringPoplnXSource, PoplnSize, ChromosomeLength)

*# normalize offspring population so that the buy/sell weights lie within their respective bounds*

OffspringPoplnXNormalized**=** WeightNormalization(OffspringPoplnXMutated, SellLowBounds , SellUpBounds, BuyLowBounds, BuyUpBounds)

*# determine lower and upper bounds of the population of buy /sell weights and store it in Python dictionary for future use*

LowUpBounds **=**{}

LowUpBounds **=** DetermineBounds(OffspringPoplnXNormalized, BuyLowBounds, BuyUpBounds, SellLowBounds , SellUpBounds)

*# repair weights of the offspring population*

OffspringPoplnX **=** PortfolioRebalancingWeightRepair(OffspringPoplnXNormalized, LowUpBounds, p )

*# obtain the population of rebalanced weights*

OffspringPoplnW **=** np**.**add(np**.**matmul(np**.**ones((PoplnSize,1)),(OriginalPortfolioOptimalWeights)) , OffspringPoplnX ) *# Xi*

*# compute constraint violation function values*

[OffspringWG, OffspringWPsi] **=** ComputeConstraintViolation(OffspringPoplnW, CovData, OriginalPortfolioDailyRisk, C, BetaParam, AlphaParam, GenerationCount) *#compute constraint violation functions using Joines and Houck's dynamic penalty functions*

*# compute fitness function values*

OffspringWFitness **=** ComputeFitness(OffspringPoplnW, MeanData, CovData, OffspringWPsi, RiskFreeRate)

*# construct the new generation of chromosomes, selecting the best from among the parent and offspring populations*

[NextGenPoolX, NextGenPoolW, NextGenPoolFitness, NextGenPoolPsi] **=** ConstructNewGeneration(FeasParentPoplnX, FeasParentPoplnW, FeasParentPoplnWFitness, FeasParentPoplnWPsi , OffspringPoplnX, OffspringPoplnW, OffspringWFitness, OffspringWPsi)

*# induct the best fit individual whose constraint violation function value is 0, from the new generation into*

*# the Hall of Fame,*

**for** i **in** range(PoplnSize):

**if** (NextGenPoolPsi[i] **==** 0):

**if** (NextGenPoolFitness[i] **>** HOFFitness):

HOFFitness **=** NextGenPoolFitness[i]

HOFIndividualW **=** NextGenPoolW[i,:]

HOFIndividualX **=** NextGenPoolX[i,:]

print('HOF Fitness', HOFFitness)

HOFGenarray[i1] **=** GenerationCount

HOFFitarray[i1] **=** HOFFitness

i1**=**i1**+**1

**else**:

**continue**

*# compute Population measure to study the convergence of the evolutionary algorithm*

GenPMeasure **=** ComputePMeasure(NextGenPoolW)

PerformanceAnalysisMeasure[PMeasureCount, :]**=** [GenerationCount, GenPMeasure]

PMeasureCount**=**PMeasureCount**+**1

*# set parent population and its parameters for the next generation cycle*

FeasParentPoplnX **=** NextGenPoolX

FeasParentPoplnW **=** NextGenPoolW

FeasParentPoplnWFitness **=** NextGenPoolFitness

FeasParentPoplnWPsi **=** NextGenPoolPsi

GenerationCount **=** GenerationCount **+** 1

*# extract the optimal solution (rebalanced weights) from the Hall of Fame*

OptimalRebalWeights **=** HOFIndividualW

print('Optimal Rebalanced Portfolio Weights', OptimalRebalWeights)

*# compute risk, return and Sharpe Ratio of the optimal rebalanced portfolio*

DailyRebalancedPortfolioReturn **=** np**.**sum(np**.**multiply(MeanData, OptimalRebalWeights))

*# number of trading days = 261*

AnnualRebalancedPortfolioReturn **=** 261 **\*** DailyRebalancedPortfolioReturn**\***100

print('Rebalanced portfolio annualized return ', AnnualRebalancedPortfolioReturn)

DailyRebalancedPortfolioRisk**=** np**.**sqrt(np**.**matmul( np**.**matmul(OptimalRebalWeights, CovData), OptimalRebalWeights**.**T))

AnnualRebalancedPortfolioRisk **=** np**.**sqrt(261)**\*** DailyRebalancedPortfolioRisk**\***100

print('Rebalanced portfolio annualized risk ', AnnualRebalancedPortfolioRisk)

SharpeRatio **=** (AnnualRebalancedPortfolioReturn**-**RiskFreeRate)**/**AnnualRebalancedPortfolioRisk

print('Rebalanced portfolio Sharpe Ratio', SharpeRatio)

*# compare results with risk, return and Sharpe Ratio of the original portfolio*

print('Original portfolio annualized return ', OriginalPortfolioAnnReturn)

print('Original portfolio annualized risk ', OriginalPortfolioAnnRisk)

print('Original portfolio Sharpe Ratio', OriginalPortfolioSharpeRatio)

print('Successful Execution!')

['3MINDIA', 'ASHOKLEY', 'BAJFINANCE', 'BHARATFORG', 'GAIL', 'GMRINFRA', 'HDFCBANK', 'HINDPETRO', 'HINDUNILVR', 'IOC', 'ITC', 'JINDALSTEL', 'JSWSTEEL', 'JUBILANT', 'LT', 'M&M', 'MINDTREE', 'NMDC', 'PFC', 'RAJESHEXPO', 'RELAXO', 'RELIANCE', 'SBIN', 'SHREECEM', 'SUNTV', 'TATASTEEL', 'TCS', 'ULTRACEMCO', 'VOLTAS', 'WIPRO']

Generation: 0

HOF Fitness 1.8913813844305976

Generation: 1

Generation: 2

Generation: 3

Generation: 4

Generation: 5

Generation: 6

Generation: 7

Generation: 8

Generation: 9

Generation: 10

Generation: 11

Generation: 12

Generation: 13

Generation: 14

Generation: 15

Generation: 16

Generation: 17

Generation: 18

Generation: 19

Generation: 20

Generation: 21

Generation: 22

Generation: 23

Generation: 24

Generation: 25

Generation: 26

Generation: 27

Generation: 28

Generation: 29

Generation: 30

Generation: 31

Generation: 32

Generation: 33

Generation: 34

Generation: 35

Generation: 36

Generation: 37

Generation: 38

Generation: 39

Generation: 40

Generation: 41

Generation: 42

Generation: 43

Generation: 44

Generation: 45

Generation: 46

Generation: 47

Generation: 48

Generation: 49

Generation: 50

Generation: 51

Generation: 52

Generation: 53

Generation: 54

Generation: 55

Generation: 56

Generation: 57

Generation: 58

Generation: 59

Generation: 60

Generation: 61

Generation: 62

Generation: 63

Generation: 64

Generation: 65

Generation: 66

Generation: 67

Generation: 68

Generation: 69

Generation: 70

Generation: 71

Generation: 72

Generation: 73

Generation: 74

Generation: 75

Generation: 76

Generation: 77

Generation: 78

Generation: 79

Generation: 80

Generation: 81

Generation: 82

Generation: 83

Generation: 84

Generation: 85

HOF Fitness 1.8994510779171148

Generation: 86

Generation: 87

HOF Fitness 1.9006401631758092

Generation: 88

HOF Fitness 1.9079327630145195

Generation: 89

HOF Fitness 1.912672446558536

Generation: 90

HOF Fitness 1.9164273352632746

Generation: 91

HOF Fitness 1.9184791941883113

Generation: 92

HOF Fitness 1.9340237031112453

Generation: 93

HOF Fitness 1.9355902506169365

Generation: 94

Generation: 95

HOF Fitness 1.9370118013441902

Generation: 96

HOF Fitness 1.9371182325016478

Generation: 97

HOF Fitness 1.9381623765689415

Generation: 98

HOF Fitness 1.9411213007287613

Generation: 99

HOF Fitness 1.9420679306090713

Generation: 100

HOF Fitness 1.9465513685600082

Generation: 101

HOF Fitness 1.9476570421097934

Generation: 102

HOF Fitness 1.950319433023251

Generation: 103

HOF Fitness 1.9524488599607952

Generation: 104

Generation: 105

Generation: 106

HOF Fitness 1.9550604104201816

Generation: 107

HOF Fitness 1.955353280754502

Generation: 108

HOF Fitness 1.9571751993154403

Generation: 109

Generation: 110

HOF Fitness 1.957664594274533

Generation: 111

HOF Fitness 1.9586856839901785

Generation: 112

HOF Fitness 1.9595044028985689

Generation: 113

HOF Fitness 1.9622308666613657

Generation: 114

HOF Fitness 1.9624909488298068

Generation: 115

HOF Fitness 1.966167753253205

Generation: 116

HOF Fitness 1.9667362952664535

Generation: 117

HOF Fitness 1.9673620719763685

Generation: 118

HOF Fitness 1.9696593391051764

Generation: 119

Generation: 120

HOF Fitness 1.9702400389785641

Generation: 121

HOF Fitness 1.9713858356131935

Generation: 122

Generation: 123

Generation: 124

HOF Fitness 1.9737599318682733

Generation: 125

Generation: 126

HOF Fitness 1.9743383563744359

Generation: 127

HOF Fitness 1.9749675572911785

Generation: 128

HOF Fitness 1.9752996912182776

Generation: 129

HOF Fitness 1.9760115176726396

Generation: 130

HOF Fitness 1.976869742584258

Generation: 131

HOF Fitness 1.977540199772637

Generation: 132

Generation: 133

Generation: 134

HOF Fitness 1.9784639808831503

Generation: 135

HOF Fitness 1.9786943037336249

Generation: 136

HOF Fitness 1.9795117288318245

Generation: 137

Generation: 138

HOF Fitness 1.9796526435094606

Generation: 139

HOF Fitness 1.9801471640503734

Generation: 140

HOF Fitness 1.9808742973827849

Generation: 141

Generation: 142

HOF Fitness 1.9809322233692712

Generation: 143

HOF Fitness 1.981269222837435

Generation: 144

HOF Fitness 1.9819325356154267

Generation: 145

Generation: 146

HOF Fitness 1.9827561439465957

Generation: 147

Generation: 148

HOF Fitness 1.9828041428584462

Generation: 149

Generation: 150

HOF Fitness 1.9837907719460053

Generation: 151

Generation: 152

Generation: 153

Generation: 154

HOF Fitness 1.983915859950058

Generation: 155

HOF Fitness 1.9845647146461765

Generation: 156

HOF Fitness 1.9847946419240394

Generation: 157

HOF Fitness 1.9858104074007437

Generation: 158

Generation: 159

Generation: 160

Generation: 161

HOF Fitness 1.9859864949888684

Generation: 162

HOF Fitness 1.9862296432218487

Generation: 163

HOF Fitness 1.9864030166434017

Generation: 164

HOF Fitness 1.9865523848301399

Generation: 165

HOF Fitness 1.9869261028121847

Generation: 166

HOF Fitness 1.987031140666182

Generation: 167

Generation: 168

HOF Fitness 1.9871865423594968

Generation: 169

HOF Fitness 1.9882218094032

Generation: 170

Generation: 171

Generation: 172

Generation: 173

HOF Fitness 1.9882863950107503

Generation: 174

HOF Fitness 1.9887588457096286

Generation: 175

Generation: 176

Generation: 177

Generation: 178

HOF Fitness 1.9893444627631929

Generation: 179

Generation: 180

HOF Fitness 1.9894635262573774

Generation: 181

HOF Fitness 1.9895755975505827

Generation: 182

Generation: 183

HOF Fitness 1.9897448032113798

Generation: 184

HOF Fitness 1.9899588198170723

Generation: 185

HOF Fitness 1.990067180613404

Generation: 186

HOF Fitness 1.9901922305113031

Generation: 187

HOF Fitness 1.9905691278475837

Generation: 188

HOF Fitness 1.9907443887384002

Generation: 189

Generation: 190

Generation: 191

HOF Fitness 1.9909225401488413

Generation: 192

Generation: 193

HOF Fitness 1.9912932792411469

Generation: 194

HOF Fitness 1.991404713776435

Generation: 195

Generation: 196

Generation: 197

HOF Fitness 1.9915421114033585

Generation: 198

Generation: 199

Generation: 200

HOF Fitness 1.9916153950360014

Generation: 201

Generation: 202

HOF Fitness 1.9917646120597035

Generation: 203

HOF Fitness 1.9918436591909987

Generation: 204

HOF Fitness 1.9919301891383534

Generation: 205

HOF Fitness 1.9919308780056662

Generation: 206

HOF Fitness 1.9921060527206578

Generation: 207

HOF Fitness 1.9922850024588046

Generation: 208

Generation: 209

HOF Fitness 1.99228621246891

Generation: 210

HOF Fitness 1.992403443651286

Generation: 211

HOF Fitness 1.9924187966924518

Generation: 212

HOF Fitness 1.9924663824878193

Generation: 213

HOF Fitness 1.992483950378479

Generation: 214

HOF Fitness 1.9925373699811124

Generation: 215

HOF Fitness 1.9927008826660684

Generation: 216

HOF Fitness 1.9927854769807172

Generation: 217

Generation: 218

HOF Fitness 1.9928131316317907

Generation: 219

Generation: 220

HOF Fitness 1.9929180145896928

Generation: 221

Generation: 222

HOF Fitness 1.992960542658324

Generation: 223

HOF Fitness 1.992987495958479

Generation: 224

HOF Fitness 1.9930349419568183

Generation: 225

HOF Fitness 1.9930563365986913

Generation: 226

HOF Fitness 1.9931081149047805

Generation: 227

HOF Fitness 1.993127994264172

Generation: 228

HOF Fitness 1.9931907196062897

Generation: 229

HOF Fitness 1.9932047089585196

Generation: 230

HOF Fitness 1.993206604612577

Generation: 231

HOF Fitness 1.9932933352174549

Generation: 232

HOF Fitness 1.9933163235384754

Generation: 233

Generation: 234

HOF Fitness 1.9934105513058962

Generation: 235

Generation: 236

Generation: 237

HOF Fitness 1.9935640309475606

Generation: 238

Generation: 239

Generation: 240

HOF Fitness 1.9936169978419052

Generation: 241

Generation: 242

HOF Fitness 1.9936178042508512

Generation: 243

HOF Fitness 1.9936312949792971

Generation: 244

HOF Fitness 1.9937305950960467

Generation: 245

Generation: 246

Generation: 247

Generation: 248

Generation: 249

HOF Fitness 1.9937733665086872

Generation: 250

HOF Fitness 1.9938830870634645

Generation: 251

Generation: 252

HOF Fitness 1.9939173357366864

Generation: 253

Generation: 254

Generation: 255

HOF Fitness 1.9939321220205055

Generation: 256

HOF Fitness 1.994003822363538

Generation: 257

Generation: 258

HOF Fitness 1.9940107477960016

Generation: 259

HOF Fitness 1.9940498405697122

Generation: 260

Generation: 261

HOF Fitness 1.994117597983387

Generation: 262

Generation: 263

Generation: 264

Generation: 265

HOF Fitness 1.9941572042092406

Generation: 266

HOF Fitness 1.9941840946518077

Generation: 267

HOF Fitness 1.994211611026383

Generation: 268

HOF Fitness 1.9942587434631744

Generation: 269

HOF Fitness 1.9942637547034756

Generation: 270

HOF Fitness 1.9942663319771787

Generation: 271

HOF Fitness 1.9944170538515544

Generation: 272

HOF Fitness 1.9944319661659458

Generation: 273

Generation: 274

Generation: 275

HOF Fitness 1.994484152157624

Generation: 276

Generation: 277

Generation: 278

HOF Fitness 1.9944889286217702

Generation: 279

HOF Fitness 1.9945060370616297

Generation: 280

HOF Fitness 1.9946283893039676

Generation: 281

HOF Fitness 1.9946309929704427

Generation: 282

Generation: 283

HOF Fitness 1.9946582497884529

Generation: 284

Generation: 285

HOF Fitness 1.9946633269706426

Generation: 286

Generation: 287

HOF Fitness 1.9947197625207573

Generation: 288

Generation: 289

HOF Fitness 1.9947718384427977

Generation: 290

Generation: 291

HOF Fitness 1.9947734056338753

Generation: 292

HOF Fitness 1.9948370590242428

Generation: 293

Generation: 294

HOF Fitness 1.994847139774813

Generation: 295

Generation: 296

Generation: 297

HOF Fitness 1.994928042737731

Generation: 298

Generation: 299

HOF Fitness 1.9949397031644738

Generation: 300

HOF Fitness 1.9949437722068195

Generation: 301

HOF Fitness 1.9949763810827585

Generation: 302

HOF Fitness 1.9949890360291789

Generation: 303

HOF Fitness 1.9950064767993638

Generation: 304

HOF Fitness 1.9950176030509248

Generation: 305

HOF Fitness 1.9950496862668998

Generation: 306

Generation: 307

HOF Fitness 1.9950729215411325

Generation: 308

HOF Fitness 1.9950978304690692

Generation: 309

HOF Fitness 1.9951257679797112

Generation: 310

HOF Fitness 1.9951289563346601

Generation: 311

HOF Fitness 1.9951509232784643

Generation: 312

HOF Fitness 1.9951635706499047

Generation: 313

HOF Fitness 1.9952011855997986

Generation: 314

HOF Fitness 1.9952079350946719

Generation: 315

Generation: 316

Generation: 317

Generation: 318

HOF Fitness 1.9952253857405413

Generation: 319

HOF Fitness 1.9952257067488741

Generation: 320

HOF Fitness 1.99523326402161

Generation: 321

Generation: 322

HOF Fitness 1.9952410373774438

Generation: 323

HOF Fitness 1.9952699630699313

Generation: 324

HOF Fitness 1.9952786033574414

Generation: 325

HOF Fitness 1.995284463779598

Generation: 326

HOF Fitness 1.9952955539020785

Generation: 327

Generation: 328

Generation: 329

HOF Fitness 1.9953041794933062

Generation: 330

HOF Fitness 1.9953133053032253

Generation: 331

Generation: 332

HOF Fitness 1.995321093858803

Generation: 333

Generation: 334

HOF Fitness 1.9953307766860218

Generation: 335

HOF Fitness 1.9953328282376366

Generation: 336

HOF Fitness 1.9953357068607944

Generation: 337

HOF Fitness 1.9953427660086567

Generation: 338

HOF Fitness 1.9953451268326803

Generation: 339

HOF Fitness 1.9954128558759674

Generation: 340

Generation: 341

Generation: 342

Generation: 343

Generation: 344

HOF Fitness 1.995422548454568

Generation: 345

Generation: 346

HOF Fitness 1.9954407862881767

Generation: 347

HOF Fitness 1.995443172727979

Generation: 348

Generation: 349

HOF Fitness 1.9954431823234435

Generation: 350

HOF Fitness 1.9954561894450868

Generation: 351

Generation: 352

Generation: 353

HOF Fitness 1.9954690321959667

Generation: 354

HOF Fitness 1.9954709438711806

Generation: 355

Generation: 356

HOF Fitness 1.9954740550131085

Generation: 357

HOF Fitness 1.9954822350081964

Generation: 358

Generation: 359

Generation: 360

HOF Fitness 1.99548758453688

Generation: 361

HOF Fitness 1.9954876999481719

Generation: 362

HOF Fitness 1.9954896030061022

Generation: 363

HOF Fitness 1.9954920947782484

Generation: 364

Generation: 365

HOF Fitness 1.9954934308978198

Generation: 366

HOF Fitness 1.9955007886816236

Generation: 367

HOF Fitness 1.9955019029266539

Generation: 368

Generation: 369

HOF Fitness 1.9955029212384439

Generation: 370

HOF Fitness 1.9955087596295897

Generation: 371

HOF Fitness 1.9955103715416838

Generation: 372

HOF Fitness 1.9955109037934238

Generation: 373

HOF Fitness 1.9955142877374277

Generation: 374

Generation: 375

HOF Fitness 1.9955173655830027

Generation: 376

Generation: 377

HOF Fitness 1.9955176985911607

Generation: 378

HOF Fitness 1.9955181493883523

Generation: 379

HOF Fitness 1.9955213503099012

Generation: 380

HOF Fitness 1.9955227096042114

Generation: 381

Generation: 382

HOF Fitness 1.995527777791405

Generation: 383

Generation: 384

HOF Fitness 1.9955283489298077

Generation: 385

HOF Fitness 1.9955295554120887

Generation: 386

HOF Fitness 1.9955296052734448

Generation: 387

HOF Fitness 1.995532910942251

Generation: 388

HOF Fitness 1.9955378337316014

Generation: 389

Generation: 390

Generation: 391

Generation: 392

HOF Fitness 1.9955386722987052

Generation: 393

HOF Fitness 1.9955426124176678

Generation: 394

Generation: 395

HOF Fitness 1.995543953461057

Generation: 396

HOF Fitness 1.9955452940608702

Generation: 397

HOF Fitness 1.9955479488622425

Generation: 398

HOF Fitness 1.9955512030516713

Generation: 399

HOF Fitness 1.99555131250809

Generation: 400

HOF Fitness 1.9955520219817584

Generation: 401

HOF Fitness 1.9955542883283672

Generation: 402

HOF Fitness 1.995586672073803

Generation: 403

Generation: 404

Generation: 405

Generation: 406

HOF Fitness 1.9955880281151592

Generation: 407

HOF Fitness 1.9955904087503988

Generation: 408

HOF Fitness 1.9955919717913253

Generation: 409

HOF Fitness 1.9955947946872465

Generation: 410

HOF Fitness 1.995595523856138

Generation: 411

HOF Fitness 1.9955983247878444

Generation: 412

HOF Fitness 1.9956044441842327

Generation: 413

Generation: 414

HOF Fitness 1.9956050868808306

Generation: 415

HOF Fitness 1.9956068917908647

Generation: 416

HOF Fitness 1.995608285724261

Generation: 417

HOF Fitness 1.9956087857668108

Generation: 418

HOF Fitness 1.9956097862822784

Generation: 419

HOF Fitness 1.9956104607461742

Generation: 420

HOF Fitness 1.9956120565543563

Generation: 421

HOF Fitness 1.9956155496590273

Generation: 422

HOF Fitness 1.995616304418327

Generation: 423

Generation: 424

HOF Fitness 1.9956172641549375

Generation: 425

HOF Fitness 1.9956189282714711

Generation: 426

HOF Fitness 1.995619405049622

Generation: 427

HOF Fitness 1.9956204064685255

Generation: 428

HOF Fitness 1.9956225641457002

Generation: 429

HOF Fitness 1.9956318133717692

Generation: 430

Generation: 431

Generation: 432

HOF Fitness 1.9956323857024174

Generation: 433

Generation: 434

HOF Fitness 1.9956356499737624

Generation: 435

Generation: 436

HOF Fitness 1.995639923957122

Generation: 437

Generation: 438

Generation: 439

Generation: 440

HOF Fitness 1.9956450011313662

Generation: 441

Generation: 442

HOF Fitness 1.9956488396556296

Generation: 443

HOF Fitness 1.9956496642208796

Generation: 444

HOF Fitness 1.9956496877886025

Generation: 445

HOF Fitness 1.9956516902298436

Generation: 446

HOF Fitness 1.9956518846678195

Generation: 447

HOF Fitness 1.9956525780598025

Generation: 448

HOF Fitness 1.995654377869386

Generation: 449

HOF Fitness 1.9956554872876158

Generation: 450

HOF Fitness 1.9956584906455148

Generation: 451

Generation: 452

HOF Fitness 1.9956586488445551

Generation: 453

HOF Fitness 1.9956598467543467

Generation: 454

HOF Fitness 1.9956612443209722

Generation: 455

HOF Fitness 1.9956614019666161

Generation: 456

HOF Fitness 1.9956632701022994

Generation: 457

Generation: 458

HOF Fitness 1.9956641925289953

Generation: 459

HOF Fitness 1.9956657904063817

Generation: 460

Generation: 461

HOF Fitness 1.995666037566053

Generation: 462

HOF Fitness 1.9956667434238016

Generation: 463

HOF Fitness 1.99566691795459

Generation: 464

HOF Fitness 1.9956682860684238

Generation: 465

Generation: 466

Generation: 467

Generation: 468

Generation: 469

HOF Fitness 1.995717855379887

Generation: 470

Generation: 471

Generation: 472

Generation: 473

Generation: 474

Generation: 475

Generation: 476

HOF Fitness 1.995728328203675

Generation: 477

HOF Fitness 1.9957694857187203

Generation: 478

HOF Fitness 1.9957748931996555

Generation: 479

Generation: 480

Generation: 481

Generation: 482

Generation: 483

Generation: 484

Generation: 485

HOF Fitness 1.9957921326456813

Generation: 486

Generation: 487

HOF Fitness 1.9958154989441796

Generation: 488

Generation: 489

HOF Fitness 1.9958159094804093

Generation: 490

Generation: 491

HOF Fitness 1.9958164761630852

Generation: 492

Generation: 493

Generation: 494

HOF Fitness 1.9958267732666763

Generation: 495

Generation: 496

HOF Fitness 1.9958306140892956

Generation: 497

HOF Fitness 1.9958332429327483

Generation: 498

Generation: 499

Generation: 500

HOF Fitness 1.9958346369191462

Generation: 501

HOF Fitness 1.99584868564455

Generation: 502

Generation: 503

HOF Fitness 1.9958488155947627

Generation: 504

HOF Fitness 1.9958518494354613

Generation: 505

HOF Fitness 1.9958520258774295

Generation: 506

Generation: 507

HOF Fitness 1.9958570061935546

Generation: 508

HOF Fitness 1.9958603107406176

Generation: 509

HOF Fitness 1.9958624857786664

Generation: 510

HOF Fitness 1.99586315590654

Generation: 511

HOF Fitness 1.995868776704902

Generation: 512

HOF Fitness 1.995873803056182

Generation: 513

Generation: 514

Generation: 515

HOF Fitness 1.995876141678638

Generation: 516

HOF Fitness 1.9958775015027073

Generation: 517

HOF Fitness 1.9958807571337016

Generation: 518

HOF Fitness 1.9958841156388754

Generation: 519

Generation: 520

HOF Fitness 1.9958905953064727

Generation: 521

Generation: 522

HOF Fitness 1.995896310617745

Generation: 523

Generation: 524

Generation: 525

HOF Fitness 1.995896775341448

Generation: 526

Generation: 527

HOF Fitness 1.9958979864370165

Generation: 528

HOF Fitness 1.9958995639127561

Generation: 529

Generation: 530

HOF Fitness 1.9958999943205158

Generation: 531

Generation: 532

HOF Fitness 1.9959001279294164

Generation: 533

HOF Fitness 1.9959048385455584

Generation: 534

Generation: 535

HOF Fitness 1.995905467665754

Generation: 536

Generation: 537

Generation: 538

Generation: 539

HOF Fitness 1.9959064761652416

Generation: 540

Generation: 541

HOF Fitness 1.9959068950013048

Generation: 542

HOF Fitness 1.9959080640416178

Generation: 543

HOF Fitness 1.9959081739610383

Generation: 544

HOF Fitness 1.9959111234966553

Generation: 545

Generation: 546

Generation: 547

Generation: 548

Generation: 549

HOF Fitness 1.9959122304204262

Generation: 550

Generation: 551

HOF Fitness 1.995912463812256

Generation: 552

HOF Fitness 1.9959132886361377

Generation: 553

HOF Fitness 1.9959140902313395

Generation: 554

Generation: 555

Generation: 556

HOF Fitness 1.9959146108380013

Generation: 557

HOF Fitness 1.9959147409253641

Generation: 558

Generation: 559

HOF Fitness 1.9959149998658736

Generation: 560

HOF Fitness 1.995915004742113

Generation: 561

HOF Fitness 1.9959150862375512

Generation: 562

HOF Fitness 1.9959159378289202

Generation: 563

Generation: 564

HOF Fitness 1.9959162104978685

Generation: 565

Generation: 566

HOF Fitness 1.9959163722320634

Generation: 567

HOF Fitness 1.9959164930304996

Generation: 568

Generation: 569

HOF Fitness 1.995916556571077

Generation: 570

Generation: 571

HOF Fitness 1.9959168739833895

Generation: 572

HOF Fitness 1.995917043790673

Generation: 573

HOF Fitness 1.9959170989085964

Generation: 574

Generation: 575

HOF Fitness 1.9959171718595894

Generation: 576

HOF Fitness 1.9959172840918364

Generation: 577

Generation: 578

HOF Fitness 1.9959174766266128

Generation: 579

HOF Fitness 1.9959176506418608

Generation: 580

Generation: 581

Generation: 582

HOF Fitness 1.9959176670769412

Generation: 583

HOF Fitness 1.995917670795162

Generation: 584

HOF Fitness 1.9959178504004744

Generation: 585

HOF Fitness 1.995917854975959

Generation: 586

HOF Fitness 1.9959178834288873

Generation: 587

HOF Fitness 1.9959179312776456

Generation: 588

HOF Fitness 1.9959179488749084

Generation: 589

HOF Fitness 1.9959179630876827

Generation: 590

HOF Fitness 1.9959179769647197

Generation: 591

HOF Fitness 1.9959179902801472

Generation: 592

HOF Fitness 1.9959273738353542

Generation: 593

Generation: 594

Generation: 595

Generation: 596

Generation: 597

HOF Fitness 1.9959277252127137

Generation: 598

Generation: 599

Generation: 600

HOF Fitness 1.9959301084930987

Generation: 601

Generation: 602

Generation: 603

Generation: 604

Generation: 605

Generation: 606

Generation: 607

HOF Fitness 1.9959305418026805

Generation: 608

Generation: 609

HOF Fitness 1.9959318342626093

Generation: 610

HOF Fitness 1.9959320307422665

Generation: 611

HOF Fitness 1.9959321010813784

Generation: 612

HOF Fitness 1.9959321168170432

Generation: 613

HOF Fitness 1.9959323152128852

Generation: 614

HOF Fitness 1.9959325179598917

Generation: 615

HOF Fitness 1.9959331828319073

Generation: 616

HOF Fitness 1.9959334197960978

Generation: 617

Generation: 618

Generation: 619

Generation: 620

Generation: 621

HOF Fitness 1.9959338004336413

Generation: 622

HOF Fitness 1.9959365716376214

Generation: 623

Generation: 624

Generation: 625

HOF Fitness 1.9959366955298807

Generation: 626

Generation: 627

Generation: 628

HOF Fitness 1.9959369147115742

Generation: 629

Generation: 630

Generation: 631

HOF Fitness 1.9959374229956304

Generation: 632

HOF Fitness 1.9959376499615193

Generation: 633

HOF Fitness 1.9959384507214235

Generation: 634

HOF Fitness 1.9959390116180924

Generation: 635

HOF Fitness 1.9959394993979673

Generation: 636

HOF Fitness 1.9959395060738363

Generation: 637

HOF Fitness 1.995940036998323

Generation: 638

HOF Fitness 1.9959404500843727

Generation: 639

Generation: 640

HOF Fitness 1.9959406071058543

Generation: 641

HOF Fitness 1.9959407226447614

Generation: 642

HOF Fitness 1.99594100905426

Generation: 643

Generation: 644

Generation: 645

HOF Fitness 1.9959411250433294

Generation: 646

HOF Fitness 1.9959413569544524

Generation: 647

HOF Fitness 1.9959419330193735

Generation: 648

HOF Fitness 1.9959420628245348

Generation: 649

HOF Fitness 1.9959422242359977

Generation: 650

HOF Fitness 1.9959422333309766

Generation: 651

HOF Fitness 1.9959426364859787

Generation: 652

HOF Fitness 1.9959426955457409

Generation: 653

HOF Fitness 1.9959444539910296

Generation: 654

Generation: 655

Generation: 656

HOF Fitness 1.9959445000180727

Generation: 657

HOF Fitness 1.9959449911785283

Generation: 658

HOF Fitness 1.9959449912505127

Generation: 659

HOF Fitness 1.9959453081964085

Generation: 660

Generation: 661

Generation: 662

Generation: 663

HOF Fitness 1.995945563310233

Generation: 664

Generation: 665

Generation: 666

HOF Fitness 1.9959470375097097

Generation: 667

Generation: 668

Generation: 669

HOF Fitness 1.9959472126344522

Generation: 670

HOF Fitness 1.9959472975453747

Generation: 671

HOF Fitness 1.9959475107962454

Generation: 672

Generation: 673

Generation: 674

HOF Fitness 1.9959475681235652

Generation: 675

HOF Fitness 1.9959477110731039

Generation: 676

Generation: 677

HOF Fitness 1.995947804737376

Generation: 678

HOF Fitness 1.995947848104018

Generation: 679

HOF Fitness 1.9959478965528297

Generation: 680

HOF Fitness 1.995947984002171

Generation: 681

HOF Fitness 1.995948130512604

Generation: 682

HOF Fitness 1.9959481594052244

Generation: 683

HOF Fitness 1.9959481735431204

Generation: 684

HOF Fitness 1.9959483213473963

Generation: 685

HOF Fitness 1.995948368245605

Generation: 686

HOF Fitness 1.9959483954363142

Generation: 687

HOF Fitness 1.9959484470925841

Generation: 688

HOF Fitness 1.9959484619351153

Generation: 689

HOF Fitness 1.9959484623507757

Generation: 690

HOF Fitness 1.9959485075817185

Generation: 691

HOF Fitness 1.9959485355678468

Generation: 692

HOF Fitness 1.9959485577361178

Generation: 693

HOF Fitness 1.99594857109199

Generation: 694

Generation: 695

Generation: 696

HOF Fitness 1.9959485726729373

Generation: 697

HOF Fitness 1.995948609197878

Generation: 698

HOF Fitness 1.9959486341779287

Generation: 699

HOF Fitness 1.9959486977016057

Generation: 700

Generation: 701

HOF Fitness 1.9959487453904416

Generation: 702

HOF Fitness 1.995948751532029

Generation: 703

HOF Fitness 1.995948771059793

Generation: 704

HOF Fitness 1.995948781032285

Generation: 705

Generation: 706

HOF Fitness 1.995948785173635

Generation: 707

Generation: 708

Generation: 709

HOF Fitness 1.995948800311291

Generation: 710

HOF Fitness 1.9959488096169693

Generation: 711

HOF Fitness 1.9959488261221874

Generation: 712

HOF Fitness 1.9959488791323297

Generation: 713

HOF Fitness 1.9959488905577054

Generation: 714

HOF Fitness 1.995948902857228

Generation: 715

HOF Fitness 1.9959489081922424

Generation: 716

HOF Fitness 1.9959489337916183

Generation: 717

Generation: 718

Generation: 719

Generation: 720

HOF Fitness 1.9959489374379893

Generation: 721

HOF Fitness 1.9959554212934552

Generation: 722

Generation: 723

Generation: 724

HOF Fitness 1.9959567337447466

Generation: 725

Generation: 726

HOF Fitness 1.995957068062302

Generation: 727

Generation: 728

Generation: 729

Generation: 730

HOF Fitness 1.9959572321792083

Generation: 731

Generation: 732

HOF Fitness 1.9959573655948377

Generation: 733

Generation: 734

Generation: 735

Generation: 736

HOF Fitness 1.9959575971309167

Generation: 737

HOF Fitness 1.9959578654973547

Generation: 738

HOF Fitness 1.9959579411156902

Generation: 739

Generation: 740

HOF Fitness 1.9959579469832585

Generation: 741

HOF Fitness 1.9959581331363963

Generation: 742

HOF Fitness 1.9959582516687684

Generation: 743

HOF Fitness 1.9959583069326983

Generation: 744

HOF Fitness 1.995958393979239

Generation: 745

HOF Fitness 1.9959584228664913

Generation: 746

HOF Fitness 1.9959584737973302

Generation: 747

HOF Fitness 1.9959585253994234

Generation: 748

HOF Fitness 1.9959586708478207

Generation: 749

HOF Fitness 1.9959587481930872

Generation: 750

HOF Fitness 1.9959588110343451

Generation: 751

HOF Fitness 1.9959588987813652

Generation: 752

Generation: 753

Generation: 754

HOF Fitness 1.995959021085024

Generation: 755

Generation: 756

HOF Fitness 1.995959021827134

Generation: 757

HOF Fitness 1.9959590432132444

Generation: 758

HOF Fitness 1.9959591113362587

Generation: 759

Generation: 760

HOF Fitness 1.9959591529882608

Generation: 761

HOF Fitness 1.9959591723514516

Generation: 762

HOF Fitness 1.995959176907216

Generation: 763

HOF Fitness 1.995959193455455

Generation: 764

HOF Fitness 1.9959592370270907

Generation: 765

HOF Fitness 1.9959592581924706

Generation: 766

Generation: 767

HOF Fitness 1.995959287464845

Generation: 768

HOF Fitness 1.9959592876393826

Generation: 769

HOF Fitness 1.9959592908254242

Generation: 770

HOF Fitness 1.9959593132963018

Generation: 771

HOF Fitness 1.995959322725494

Generation: 772

HOF Fitness 1.995959365283498

Generation: 773

HOF Fitness 1.9959593779603806

Generation: 774

HOF Fitness 1.9959594008971446

Generation: 775

HOF Fitness 1.9959594059482246

Generation: 776

HOF Fitness 1.9959594174555666

Generation: 777

HOF Fitness 1.995966227961732

Generation: 778

HOF Fitness 1.995966241492907

Generation: 779

Generation: 780

HOF Fitness 1.9959663895875703

Generation: 781

Generation: 782

Generation: 783

Generation: 784

Generation: 785

Generation: 786

HOF Fitness 1.9959664859809434

Generation: 787

Generation: 788

HOF Fitness 1.995966687273276

Generation: 789

HOF Fitness 1.995966961941924

Generation: 790

HOF Fitness 1.995967298751389

Generation: 791

Generation: 792

Generation: 793

HOF Fitness 1.995967417133786

Generation: 794

HOF Fitness 1.9959674387339652

Generation: 795

HOF Fitness 1.995967681539067

Generation: 796

Generation: 797

Generation: 798

HOF Fitness 1.995967752298861

Generation: 799

Optimal Rebalanced Portfolio Weights [9.80860235e-02 4.30030537e-04 2.92137422e-01 0.00000000e+00

0.00000000e+00 0.00000000e+00 1.99820404e-02 9.82360463e-03

4.98625985e-02 1.02777711e-04 1.14406177e-02 0.00000000e+00

0.00000000e+00 1.39749293e-02 0.00000000e+00 0.00000000e+00

7.41518618e-02 0.00000000e+00 0.00000000e+00 2.85041348e-02

2.05779961e-01 7.17534564e-03 0.00000000e+00 9.31317887e-02

1.68239282e-03 0.00000000e+00 5.22282918e-02 1.44110597e-02

1.02757192e-02 1.49870047e-02]

Rebalanced portfolio annualized return 49.31072822380834

Rebalanced portfolio annualized risk 21.67280678844272

Rebalanced portfolio Sharpe Ratio 2.2729279462813836

Original portfolio annualized return 53.406

Original portfolio annualized risk 23.002

Original portfolio Sharpe Ratio 2.104436

Successful Execution!

## 9. Optimal Rebalanced Portfolio - Analysis of Results

### 9.1 Characteristics of the Optimal Rebalanced Portfolio

Typical of metaheuristic strategies, ES HOF yields multiple optimal solutions during its various runs. The interpretation of the results obtained by ES HOF during one of its runs is as shown in Fig. 3.

The optimal buy/sell weights obtained by ES HOF (buy shown in muted green, sell shown in muted orange and hold shown in yellow) and the corresponding rebalanced portfolio weights have been shown in the figure. The Sharpe Ratios, Annualized Risk(%) and Expected Portfolio Return(%) of the original and rebalanced portfolios have also been listed. It can be seen that the risk of the optimal rebalanced portfolio does not exceed that of the original portfolio.

The rebalanced portfolio now comprises only 17 assets from out of 30 assets invested in the original portfolio. It can be seen in Fig. 3 that the assets [ASHOKLEY], [BHARATFORG], [GAIL], [GMRINFRA], [IOC], [JINDALSTEL], [JSWSTEEL], [LT], [M&M], [NMDC], [PFC],[SBIN], [TATASTEEL] have been divested of during optimal rebalancing.

A picture containing calendar

Description automatically generated

#### Fig. 3 Characteristics of the rebalanced portfolio (rebalanced on June 02, 2020) as compared with those of the original portfolio (invested on April 02, 2019) and kept untended up till June 01, 2020.

### 9.2 Multiple solutions - Making a Choice!

It needs to be emphasized that due to the stochastic behavior of exploring solutions over large search spaces by generating a random population of individuals, which evolve over the generations into optimal solutions, metaheuristic strategies are endowed with inherent capabilities to deliver multiple solutions that are either optimal or acceptable or near-optimal. Thus metaheuristic strategies generate multiple solutions to the problem in hand for various runs. This characteristic of metaheuristic algorithms needs to be viewed meritoriously and exploited to one's own advantage.

Table 1 shows the summarized characteristics of optimal rebalanced portfolios delivered by ES HOF during various runs.

#### Table 1 Summarized characteristics of optimal rebalanced portfolios of S&PBSE200 index obtained by ES HOF

Table

Description automatically generated

### 9.3 Convergence characteristics of ES HOF

Observing the convergence behaviour of metaheuristic algorithms is an essential part of the study. A distribution based criterion viz., P Measure [FEO 2006] has been employed to observe the convergence behaviour of ES HOF. The generation number and P Measure value of the new generation population, comprising the best fit parent and offspring chromosomes, has been recorded. The Numpy array PerformanceAnalysisMeasure in the main program of ES HOF records these pair of values in each generation.

Fig. 4 illustrates the trace of the P Measure values during the generation cycles of a few sample runs. It can be observed that P Measure values of ES HOF in a matter of 200~300 generations, have begun converging towards 0, indicating faster and better convergence.

Chart

Description automatically generated

#### Fig. 4 Convergence of ES HOF during a few sample runs

### 9.4 Relentless rebalancing!

The S&PBSE200 index portfolio rebalanced on June 02, 2020 by ES HOF resulted in a portfolio comprising 17 assets. This portfolio was kept untended until Jan 26 2021 and rebalanced again on Jan 27 2021 using ES HOF. Fig 4 illustrates the optimal buy/sell weights, the rebalanced weights, the Sharpe Ratio, the annualized risk(%) and the expected portfolio annualized return(%) of the rebalanced portfolio by ES HOF.

It can be observed that the optimal rebalanced portfolio satisfies all the constraints imposed on it.

The muted green cells indicate "buy stock", muted orange cells indicate "sell stock" and the yellow cells indicate "hold stock" decisions.

Chart

Description automatically generated with medium confidence

#### Fig. 5 Characteristics of the rebalanced portfolio (rebalanced on Jan 27, 2021) as compared with those of the already once rebalanced portfolio (rebalanced on June 02, 2020) and kept untended until Jan 26, 2021

## 10. Conclusion

This work demonstrates how a metaheuristic strategy (ES HOF) is able to obtain optimal rebalanced portfolios. An S&P BSE200 equity portfolio invested on April 02, 2019 and kept untended up till June 01, 2020 due to the pandemic, was rebalanced on June 02, 2020. Again, the rebalanced portfolio was kept untended up till Jan 26, 2021 and rebalanced again on Jan 27, 2021. It can be observed that ES HOF was able to obtain optimal rebalanced portfolios that satisfied all the constraints imposed on it by the investor. The buy/sell/hold decisions put forth by ES HOF satisfied their respective constraints and the risk of the rebalanced portfolio did not exceed that of the original portfolio as well.

At this juncture, it pays to recall Qian's [QIA 2014] intriguing question "To rebalance or not to rebalance?" and the significant assertions that he made in this regard. Sec. 7.5.2 of Chapter 7 Metaheuristic Portfolio Rebalancing with Transaction Costs [PAI 2018] details investigation of Qian's assertions on a high risk portfolio invested in S&P BSE 200.

## Companion Reading

[1] Chapter 2 A Brief Primer on Metaheuristics [PAI 2018]

[2] MATLAB Demonstration of Metaheuristic Portfolio Rebalancing with Transaction Costs in Mathworks Central File Exchange

<https://in.mathworks.com/matlabcentral/fileexchange/64507-metaheuristic-portfolio-optimization-models>

[3] Sharpe Ratio based Portfolio Optimization

<https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson6_SharpeRatioOptimization/Lesson6_MainContent.ipynb>

[4] Heuristic Portfolio Selection

<https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson3_HeuristicPortfolioSelection/Lesson3_MainContent.ipynb>

[5] Fundamentals of Risk and Return of a Portfolio

<https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson1_FundaRiskReturnPortfolio/Lesson1_MainContent.ipynb>

## References

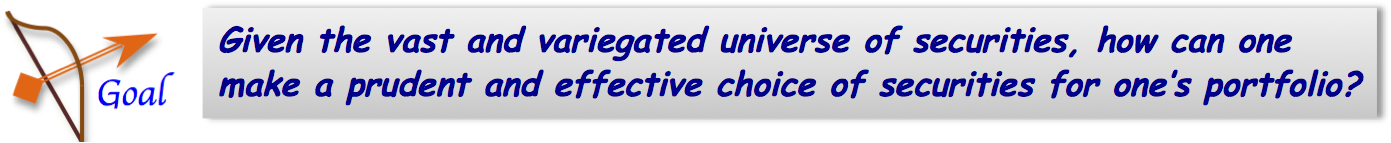
[FEO 2006] Feoktistov V, "Differential Evolution in search of solutions", Springer, 2006.

[JOI 1994] Joines J A and C R Houck, "On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs", Proceedings of the First IEEE Conference on Evolutionary Computation, pp.579-584, 1994.

[PAI 2018] Vijayalakshmi Pai G. A., "Metaheuristics for Portfolio Optimization- An Introduction using MATLAB", Wiley-ISTE, 2018. <https://www.mathworks.com/academia/books/metaheuristics-for-portfolio-optimization-pai.html>

[QIA 2014] Qian E E, "To rebalance or not to rebalance: A statistical comparison of terminal wealth of fixed weight and buy-and-hold portfolios", <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2402679>, Jan 26, 2014.

# Heuristic Portfolio Selection



## 3.1 Introduction

The universe of stocks, both local and global, can truly baffle investors who wish to make a modest selection of stocks for their portfolios. The vastness of the choices and the diverse behavioural characteristics of each of these stocks with respect to itself and to one another, can render a prudent selection of stocks to be a daunting task indeed!

Fig. 3.1 illustrates an investor's perception of the challenging task of portfolio selection.

Diagram

Description automatically generated

#### Fig. 3.1 The challenging task of Portfolio Selection - an investor's perception

Also, there is this adage, "never put all your eggs in the same basket"! So, how would an investor know that the stocks picked do not perform similarly, warranting a risk when all stocks react similarly to market events thereby impacting the performance of the portfolio?

There are several answers available to these questions independently and collectively from traditional investment finance, but a commonly agreed upon answer is Diversification. Diversification involves investments in different assets or asset classes or markets. A portfolio that comprises such a diversified set of securities can go a long way in mitigating risk since the securities would react differently to market events.

The choice of assets in an investor's portfolio reflects the investor's risk appetite or the willingness to take risk. It is an established investment principle that investments giving high returns are always associated with a probability of high risks and those that are less risky are associated with a probability of low returns.

Therefore, while risk averse investors, as a matter of fact, would choose to invest a larger share of their capital in risk-free instruments - government bonds, for example, which are notionally risk-free, risk-seeking investors on the other hand, would choose to invest a lion's share of their capital in risky instruments - equities or currencies or commodities, for example, which are prone to yield quick and high returns and therefore are inherently risky.

Investors therefore, may hold portfolios constituted by a mix of securities that can render the portfolio investments to range from less risky through moderately risky to highly risky, as measured by a "riskometer".

Fig. 3.2 graphically illustrates some investors' portfolio "doughnuts" with the investors' risk appetites and approach to investments symbolically described by the icons superscribed over the investor labels. Thus, Investor C seems to be risk averse considering the larger investments made in bonds, which are risk-free, when compared to equities which are generally risky. Investor B turns out to be the opposite owning a high risk portfolio, that invests a lion's share of the capital in equities. Investor D and Investor A, both owning chequered portfolios that invest lesser in bonds and more on other securities, could be deemed to be holding moderately high and high risk portfolios, respectively.

Diagram

Description automatically generated

#### Fig. 3.2 Portfolio selection vs Investors' risk appetites

## 3.2 Diversification Index

A Diversification Index quantifies diversification. There are several diversification indices discussed in the literature. Diversification Ratio proposed and patented by Yves Choueifaty in 2008 [CHO 08, CHO 13], is a diversification index of recent origin, built on the inter-dependence between assets of a portfolio. Diversification Ratio is the ratio of the weighted sum of individual asset volatilities to the portfolio's volatility.

Let N be the number of assets in the portfolio spanning different asset classes or belonging to a specific class. Let $(\bar{w}=(w\_1,w\_2,...w\_N) )$ be the weights or the proportion of capital to be invested in individual assets in the portfolio and $\bar{w}'$ its transpose. Let $(\bar{\sigma}=(\sigma\_1,\sigma\_2,...\sigma\_N))$ be the standard deviations of returns on the assets and V, the variance-covariance matrix of returns on the assets. The Diversification Ratio of a portfolio is given as follows:

Shape

Description automatically generated with medium confidence

##### ..........(3.1)

A portfolio that is most diversified would yield the maximal Diversification Ratio.

## 3.3 Clustering

A one-shot solution to ensure prudent selection of assets from a stock universe, which will ensure benefits of Diversification is Clustering or Cluster Analysis. Clustering deals with the task of grouping a set of physical or abstract objects into classes such that objects within a class exhibit close similarity to one another, while simultaneously expressing a strong dissimilarity to objects with other classes.

Fig. 3.3 graphically illustrates clustering of objects represented by points, into three clusters. It can be easily seen that those points that are in proximity to one another have grouped themselves into a cluster. Marking the points in each of these clusters with different symbols, visibly shows within-class similarity of objects in the cluster and out-of-class dissimilarity to objects in other clusters.

Thus, clustering exploits the characteristic of objects with similar features or behaviour, gravitating towards one another, while moving away or repelling the influence of objects which are dissimilar to them in features or behaviour, thereby satisfying the adage, "birds of a feather flock together".

Chart, diagram, bubble chart

Description automatically generated

#### Fig. 3.3 Graphical illustration of clustering objects represented by points, into groups

Several clustering methods have been explored. $k$-means clustering is a cluster analysis technique which groups $N$ objects into **k** clusters, where $k$ is an input decided by the user. In the case of portfolio selection, if $N$ were to indicate the size of the stock universe, $k$ could indicate the portfolio size and the investor can use the input $k$ to his or her advantage to make a choice of small portfolios (typically $k\le30$) or a large portfolio (typically $k >30$).

## 3.4 Case Study

Let us restrict our discussion to selection of equity stocks, after all any asset allocation plan involves significant portion of the capital being invested in equities.

We shall consider a "mini" stock universe of 29 equity stocks of Dow Jones Industrial Average (DJIA) Index viz., Apple (AAPL), American Express (AXP), Boeing (BA), Caterpillar (CAT), Cisco Systems (CSCO), Chevron (CVX), Walt Disney (DIS), Goldman Sachs (GS), The Home Depot (HD), IBM (IBM), Intel (INTC), Johnson & Johnson (JNJ), JP Morgan Chase (JPM), Coca-Cola (KO), McDonald's (MCD), 3M(MMM), Merck & Co (MRK), Microsoft (MSFT), Nike (NKE), Pfizer (PFE), Procter & Gamble (PG), Travelers (TRV), United Health Group (UNH), United Technologies (UTX), Verizon (V), Verizon (VZ), Walgreens Boots Alliance (WBA), Walmart (WMT), Exxon Mobil (XOM).

The data set considered is from April 11, 2014 to April 11, 2019. Fig. 3.4 illustrates a snapshot of the DJIA dataset.

Table

Description automatically generated

#### Fig. 3.4 Snapshot of the DJIA dataset

An investor desires to invest in 15 stocks from this "mini" universe. The following questions arise:

How can the investor decide on which combination of assets among the 29 stocks, is the best?  
How can diversification be ensured, when the assets belong to different sectors and therefore behave differently under varying market conditions?

$k$-means clustering provides solutions to both questions at once. The following steps and the companion Python code illustrate this heuristic portfolio selection process. The Python code employs NumPy, Pandas and scikit-learn to effect portfolio selection using $k$-means clustering.

Step 1: **Undertake data wrangling of the original stock dataset to keep it fit for further processing**.

(Refer Lesson2 Some glimpses of financial data wrangling to learn about aspects of financial data wrangling).  
We assume that the DJIA dataset for the "mini" universe of 29 stocks is already cleaned and the Python code shown below reads the CSV file concerned. clusters = 15 represents the portfolio size desired by the investor.

In [15]:

*#read stock prices from a cleaned DJIA dataset*

*#Dependencies*

**import** numpy **as** np

**import** pandas **as** pd

**from** sklearn.cluster **import** KMeans

*#input stock prices data set*

stockFileName **=** 'DJIA\_Apr112014\_Apr112019.csv'

originalRows **=** 1259 *#excluding header*

originalColumns **=** 29 *#excluding date*

clusters **=** 15

*#read stock dataset into a dataframe*

df **=** pd**.**read\_csv(stockFileName, nrows**=** originalRows)

*#extract asset labels*

assetLabels **=** df**.**columns[1:originalColumns**+**1]**.**tolist()

print(assetLabels)

*#extract stock prices excluding header and trading dates*

dfStockPrices **=** df**.**iloc[0:, 1:]

*#store stock prices as an array*

arStockPrices **=** np**.**asarray(dfStockPrices)

[rows, cols]**=** arStockPrices**.**shape

print(rows, cols)

print(arStockPrices)

['AAPL', 'AXP', 'BA', 'CAT', 'CSCO', 'CVX', 'DIS', 'GS', 'HD', 'IBM', 'INTC', 'JNJ', 'JPM', 'KO', 'MCD', 'MMM', 'MRK', 'MSFT', 'NKE', 'PFE', 'PG', 'TRV', 'UNH', 'UTX', 'V', 'VZ', 'WBA', 'WMT', 'XOM']

1259 29

[[ 74.23 84.54 122.07 ... 64.26 76.5 96.72 ]

[ 74.52571 85.5 123.25 ... 65.67 77.38 97.86 ]

[ 73.99429 86.04 124.27 ... 66.01 76.88 98.68 ]

...

[199.5 109.85 369.04001 ... 54.5 98.69 81.93 ]

[200.61999 110.16 364.94 ... 54.51 99.6 81.56 ]

[198.95 109.85 370.16 ... 53.44 100.8 81.95 ]]

Step 2: **Compute asset returns of the stocks**

(Refer Lesson 1 Fundamentals of Risk and Return of a Portfolio to know about stock returns computing)  
Function StockReturnsComputing computes the daily return of stocks in the DJIA index and stores it in the array arReturns, as shown in the Python code fragment given below.

In [16]:

*#function for Stock Returns computing*

**def** StockReturnsComputing(StockPrice, Rows, Columns):

**import** numpy **as** np

StockReturn **=** np**.**zeros([Rows**-**1, Columns])

**for** j **in** range(Columns): *# j: Assets*

**for** i **in** range(Rows**-**1): *#i: Daily Prices*

StockReturn[i,j]**=**((StockPrice[i**+**1, j]**-**StockPrice[i,j])**/**StockPrice[i,j])

**return** StockReturn

In [17]:

*#compute daily returns of all stocks in the mini universe*

arReturns **=** StockReturnsComputing(arStockPrices, rows, cols)

print('Size of the array of daily returns of stocks:\n', arReturns**.**shape)

print('Array of daily returns of stocks\n', arReturns)

Size of the array of daily returns of stocks:

(1258, 29)

Array of daily returns of stocks

[[ 0.00398 0.01136 0.00967 ... 0.02194 0.0115 0.01179]

[-0.00713 0.00632 0.00828 ... 0.00518 -0.00646 0.00838]

[ 0.00203 0.01581 0.01424 ... 0.00227 0.00442 0.01277]

...

[-0.003 -0.00768 -0.01463 ... -0.01017 -0.00544 -0.01289]

[ 0.00561 0.00282 -0.01111 ... 0.00018 0.00922 -0.00452]

[-0.00832 -0.00281 0.0143 ... -0.01963 0.01205 0.00478]]

Step 3: **Compute mean returns and variance-covariance matrix of returns**

(Refer Lesson 1 Fundamentals of Risk and Return of a Portfolio to know about computing the mean returns and variance-covariance matrix of stock returns)  
meanReturns and covReturns store the outputs of the respective computations.

In [18]:

*#compute mean returns and variance covariance matrix of returns*

meanReturns **=** np**.**mean(arReturns, axis **=** 0)

print('Mean returns:\n', meanReturns)

covReturns **=** np**.**cov(arReturns, rowvar**=False**)

*#set precision for printing results*

np**.**set\_printoptions(precision**=**5, suppress **=** **True**)

print('Size of Variance-Covariance matrix of returns:\n', covReturns**.**shape)

print('Variance-Covariance matrix of returns:\n', covReturns)

Mean returns:

[ 0.0009 0.00029 0.001 0.00039 0.00081 0.00016 0.0004 0.00033

0.00085 -0.00016 0.00073 0.00032 0.0006 0.00019 0.00057 0.00044

0.00036 0.001 0.0008 0.00034 0.00025 0.00043 0.00095 0.00019

0.00101 0.00023 -0.00002 0.00029 -0.00006]

Size of Variance-Covariance matrix of returns:

(29, 29)

Variance-Covariance matrix of returns:

[[0.00024 0.00007 0.0001 0.0001 0.0001 0.00007 0.00007 0.0001 0.00007

0.00007 0.00011 0.00005 0.00008 0.00003 0.00005 0.00007 0.00005 0.00012

0.00008 0.00005 0.00004 0.00005 0.00008 0.00007 0.0001 0.00003 0.00007

0.00004 0.00006]

[0.00007 0.00016 0.00008 0.0001 0.00007 0.00006 0.00006 0.00011 0.00007

0.00007 0.00007 0.00005 0.0001 0.00003 0.00004 0.00007 0.00006 0.00008

0.00007 0.00005 0.00003 0.00005 0.00007 0.00007 0.00008 0.00003 0.00007

0.00004 0.00005]

[0.0001 0.00008 0.00023 0.00013 0.00009 0.00008 0.00007 0.00011 0.00007

0.00008 0.0001 0.00006 0.0001 0.00004 0.00005 0.00009 0.00006 0.00009

0.00008 0.00005 0.00004 0.00006 0.00007 0.00009 0.00009 0.00004 0.00007

0.00005 0.00007]

[0.0001 0.0001 0.00013 0.00027 0.0001 0.00012 0.00007 0.00013 0.00008

0.00009 0.00012 0.00005 0.00012 0.00004 0.00005 0.00011 0.00006 0.00011

0.00008 0.00006 0.00004 0.00007 0.00007 0.0001 0.0001 0.00004 0.00007

0.00004 0.00011]

[0.0001 0.00007 0.00009 0.0001 0.00018 0.00008 0.00007 0.00009 0.00007

0.00008 0.00012 0.00006 0.00009 0.00004 0.00005 0.00008 0.00006 0.00011

0.00008 0.00006 0.00004 0.00006 0.00007 0.00007 0.00009 0.00005 0.00007

0.00006 0.00007]

[0.00007 0.00006 0.00008 0.00012 0.00008 0.00019 0.00006 0.00009 0.00006

0.00007 0.00008 0.00005 0.00009 0.00004 0.00004 0.00007 0.00006 0.00008

0.00005 0.00005 0.00004 0.00006 0.00006 0.00006 0.00007 0.00005 0.00006

0.00004 0.00013]

[0.00007 0.00006 0.00007 0.00007 0.00007 0.00006 0.00014 0.00008 0.00006

0.00006 0.00007 0.00004 0.00007 0.00003 0.00004 0.00006 0.00005 0.00007

0.00007 0.00005 0.00004 0.00005 0.00006 0.00005 0.00007 0.00004 0.00006

0.00004 0.00006]

[0.0001 0.00011 0.00011 0.00013 0.00009 0.00009 0.00008 0.00021 0.00008

0.00008 0.0001 0.00005 0.00016 0.00003 0.00005 0.00008 0.00007 0.0001

0.00008 0.00006 0.00004 0.00008 0.00008 0.00008 0.0001 0.00004 0.00008

0.00004 0.00008]

[0.00007 0.00007 0.00007 0.00008 0.00007 0.00006 0.00006 0.00008 0.00014

0.00006 0.00007 0.00005 0.00008 0.00003 0.00005 0.00006 0.00005 0.00008

0.00008 0.00005 0.00004 0.00005 0.00007 0.00006 0.00008 0.00004 0.00007

0.00006 0.00005]

[0.00007 0.00007 0.00008 0.00009 0.00008 0.00007 0.00006 0.00008 0.00006

0.00016 0.00008 0.00005 0.00008 0.00004 0.00004 0.00007 0.00006 0.00009

0.00006 0.00005 0.00004 0.00005 0.00006 0.00007 0.00008 0.00004 0.00005

0.00004 0.00006]

[0.00011 0.00007 0.0001 0.00012 0.00012 0.00008 0.00007 0.0001 0.00007

0.00008 0.00025 0.00006 0.00009 0.00004 0.00004 0.00009 0.00006 0.00013

0.00007 0.00006 0.00004 0.00006 0.00007 0.00008 0.00009 0.00005 0.00007

0.00005 0.00007]

[0.00005 0.00005 0.00006 0.00005 0.00006 0.00005 0.00004 0.00005 0.00005

0.00005 0.00006 0.0001 0.00005 0.00004 0.00004 0.00006 0.00006 0.00006

0.00005 0.00006 0.00004 0.00005 0.00006 0.00005 0.00005 0.00004 0.00006

0.00004 0.00005]

[0.00008 0.0001 0.0001 0.00012 0.00009 0.00009 0.00007 0.00016 0.00008

0.00008 0.00009 0.00005 0.00017 0.00003 0.00005 0.00008 0.00007 0.00009

0.00007 0.00006 0.00004 0.00008 0.00008 0.00008 0.00009 0.00004 0.00007

0.00004 0.00008]

[0.00003 0.00003 0.00004 0.00004 0.00004 0.00004 0.00003 0.00003 0.00003

0.00004 0.00004 0.00004 0.00003 0.00008 0.00004 0.00004 0.00004 0.00004

0.00004 0.00003 0.00005 0.00004 0.00003 0.00003 0.00004 0.00004 0.00004

0.00003 0.00003]

[0.00005 0.00004 0.00005 0.00005 0.00005 0.00004 0.00004 0.00005 0.00005

0.00004 0.00004 0.00004 0.00005 0.00004 0.00011 0.00004 0.00004 0.00006

0.00005 0.00003 0.00004 0.00004 0.00004 0.00004 0.00005 0.00004 0.00004

0.00004 0.00004]

[0.00007 0.00007 0.00009 0.00011 0.00008 0.00007 0.00006 0.00008 0.00006

0.00007 0.00009 0.00006 0.00008 0.00004 0.00004 0.00012 0.00006 0.00008

0.00006 0.00006 0.00004 0.00006 0.00006 0.00007 0.00007 0.00004 0.00006

0.00004 0.00006]

[0.00005 0.00006 0.00006 0.00006 0.00006 0.00006 0.00005 0.00007 0.00005

0.00006 0.00006 0.00006 0.00007 0.00004 0.00004 0.00006 0.00015 0.00006

0.00005 0.00008 0.00004 0.00005 0.00006 0.00005 0.00006 0.00005 0.00006

0.00005 0.00006]

[0.00012 0.00008 0.00009 0.00011 0.00011 0.00008 0.00007 0.0001 0.00008

0.00009 0.00013 0.00006 0.00009 0.00004 0.00006 0.00008 0.00006 0.00021

0.00008 0.00007 0.00005 0.00006 0.00008 0.00008 0.00012 0.00005 0.00008

0.00005 0.00007]

[0.00008 0.00007 0.00008 0.00008 0.00008 0.00005 0.00007 0.00008 0.00008

0.00006 0.00007 0.00005 0.00007 0.00004 0.00005 0.00006 0.00005 0.00008

0.00022 0.00005 0.00004 0.00005 0.00007 0.00006 0.00008 0.00004 0.00007

0.00005 0.00005]

[0.00005 0.00005 0.00005 0.00006 0.00006 0.00005 0.00005 0.00006 0.00005

0.00005 0.00006 0.00006 0.00006 0.00003 0.00003 0.00006 0.00008 0.00007

0.00005 0.00012 0.00004 0.00005 0.00007 0.00005 0.00006 0.00004 0.00006

0.00004 0.00005]

[0.00004 0.00003 0.00004 0.00004 0.00004 0.00004 0.00004 0.00004 0.00004

0.00004 0.00004 0.00004 0.00004 0.00005 0.00004 0.00004 0.00004 0.00005

0.00004 0.00004 0.00009 0.00004 0.00004 0.00004 0.00004 0.00004 0.00004

0.00004 0.00004]

[0.00005 0.00005 0.00006 0.00007 0.00006 0.00006 0.00005 0.00008 0.00005

0.00005 0.00006 0.00005 0.00008 0.00004 0.00004 0.00006 0.00005 0.00006

0.00005 0.00005 0.00004 0.00011 0.00006 0.00006 0.00006 0.00004 0.00006

0.00004 0.00005]

[0.00008 0.00007 0.00007 0.00007 0.00007 0.00006 0.00006 0.00008 0.00007

0.00006 0.00007 0.00006 0.00008 0.00003 0.00004 0.00006 0.00006 0.00008

0.00007 0.00007 0.00004 0.00006 0.00017 0.00006 0.00007 0.00004 0.00008

0.00005 0.00005]

[0.00007 0.00007 0.00009 0.0001 0.00007 0.00006 0.00005 0.00008 0.00006

0.00007 0.00008 0.00005 0.00008 0.00003 0.00004 0.00007 0.00005 0.00008

0.00006 0.00005 0.00004 0.00006 0.00006 0.00013 0.00007 0.00004 0.00006

0.00004 0.00006]

[0.0001 0.00008 0.00009 0.0001 0.00009 0.00007 0.00007 0.0001 0.00008

0.00008 0.00009 0.00005 0.00009 0.00004 0.00005 0.00007 0.00006 0.00012

0.00008 0.00006 0.00004 0.00006 0.00007 0.00007 0.00017 0.00003 0.00007

0.00004 0.00006]

[0.00003 0.00003 0.00004 0.00004 0.00005 0.00005 0.00004 0.00004 0.00004

0.00004 0.00005 0.00004 0.00004 0.00004 0.00004 0.00004 0.00005 0.00005

0.00004 0.00004 0.00004 0.00004 0.00004 0.00004 0.00003 0.00012 0.00005

0.00004 0.00004]

[0.00007 0.00007 0.00007 0.00007 0.00007 0.00006 0.00006 0.00008 0.00007

0.00005 0.00007 0.00006 0.00007 0.00004 0.00004 0.00006 0.00006 0.00008

0.00007 0.00006 0.00004 0.00006 0.00008 0.00006 0.00007 0.00005 0.00026

0.00006 0.00005]

[0.00004 0.00004 0.00005 0.00004 0.00006 0.00004 0.00004 0.00004 0.00006

0.00004 0.00005 0.00004 0.00004 0.00003 0.00004 0.00004 0.00005 0.00005

0.00005 0.00004 0.00004 0.00004 0.00005 0.00004 0.00004 0.00004 0.00006

0.00015 0.00004]

[0.00006 0.00005 0.00007 0.00011 0.00007 0.00013 0.00006 0.00008 0.00005

0.00006 0.00007 0.00005 0.00008 0.00003 0.00004 0.00006 0.00006 0.00007

0.00005 0.00005 0.00004 0.00005 0.00005 0.00006 0.00006 0.00004 0.00005

0.00004 0.00014]]

Step 4: **Prepare parameters for k-means clustering**

Every asset $A\_i$ is characterized by its mean return and the variance-covariance vector of its returns with those of other assets $A\_j$. For i = j, it would indicate its own variance of returns. Thus the characteristic vector for asset $A\_i$ is given by $\left[\mu\_i, \sigma\_{i1},\sigma\_{i2},...\sigma\_{ii}, ...\sigma\_{iN}) \right]$, where $\mu\_i$ indicates the mean return of asset $A\_i$ and $\sigma\_{i1},\sigma\_{i2},...\sigma\_{ii}, ...\sigma\_{iN}$ are the variance and covariance of its returns with other assets. It can be seen that $\left[ \sigma\_{i1},\sigma\_{i2},...\sigma\_{ii}, ...\sigma\_{iN} \right]$ is nothing but row $i$ of the variance-covariance matrix $V$ of $N$ assets in the stock universe. $\sigma\_{ii}$ which is the variance of the asset return, is the diagonal element of matrix $V$ in row i.

The following Python code shows the gathering of parameters for each of the 29 assets in the stock universe. The parameters are to be provided as inputs to the $k$-means clustering method. Each characteristic vector of the asset comprises 30 components viz., its own mean return as the first element of the vector followed by its covariance/variance of returns with the rest of the 29 assets. Thus assetParameter holds the the characteristic vectors of all the 29 assets in the stock universe.

In [19]:

*#prepare asset parameters for k-means clustering*

*#reshape for concatenation*

meanReturns **=** meanReturns**.**reshape(len(meanReturns),1)

assetParameters **=** np**.**concatenate([meanReturns, covReturns], axis **=** 1)

print('Size of the asset parameters for clustering:\n', assetParameters**.**shape)

print('Asset parameters for clustering:\n', assetParameters)

Size of the asset parameters for clustering:

(29, 30)

Asset parameters for clustering:

[[ 0.0009 0.00024 0.00007 0.0001 0.0001 0.0001 0.00007 0.00007

0.0001 0.00007 0.00007 0.00011 0.00005 0.00008 0.00003 0.00005

0.00007 0.00005 0.00012 0.00008 0.00005 0.00004 0.00005 0.00008

0.00007 0.0001 0.00003 0.00007 0.00004 0.00006]

[ 0.00029 0.00007 0.00016 0.00008 0.0001 0.00007 0.00006 0.00006

0.00011 0.00007 0.00007 0.00007 0.00005 0.0001 0.00003 0.00004

0.00007 0.00006 0.00008 0.00007 0.00005 0.00003 0.00005 0.00007

0.00007 0.00008 0.00003 0.00007 0.00004 0.00005]

[ 0.001 0.0001 0.00008 0.00023 0.00013 0.00009 0.00008 0.00007

0.00011 0.00007 0.00008 0.0001 0.00006 0.0001 0.00004 0.00005

0.00009 0.00006 0.00009 0.00008 0.00005 0.00004 0.00006 0.00007

0.00009 0.00009 0.00004 0.00007 0.00005 0.00007]

[ 0.00039 0.0001 0.0001 0.00013 0.00027 0.0001 0.00012 0.00007

0.00013 0.00008 0.00009 0.00012 0.00005 0.00012 0.00004 0.00005

0.00011 0.00006 0.00011 0.00008 0.00006 0.00004 0.00007 0.00007

0.0001 0.0001 0.00004 0.00007 0.00004 0.00011]

[ 0.00081 0.0001 0.00007 0.00009 0.0001 0.00018 0.00008 0.00007

0.00009 0.00007 0.00008 0.00012 0.00006 0.00009 0.00004 0.00005

0.00008 0.00006 0.00011 0.00008 0.00006 0.00004 0.00006 0.00007

0.00007 0.00009 0.00005 0.00007 0.00006 0.00007]

[ 0.00016 0.00007 0.00006 0.00008 0.00012 0.00008 0.00019 0.00006

0.00009 0.00006 0.00007 0.00008 0.00005 0.00009 0.00004 0.00004

0.00007 0.00006 0.00008 0.00005 0.00005 0.00004 0.00006 0.00006

0.00006 0.00007 0.00005 0.00006 0.00004 0.00013]

[ 0.0004 0.00007 0.00006 0.00007 0.00007 0.00007 0.00006 0.00014

0.00008 0.00006 0.00006 0.00007 0.00004 0.00007 0.00003 0.00004

0.00006 0.00005 0.00007 0.00007 0.00005 0.00004 0.00005 0.00006

0.00005 0.00007 0.00004 0.00006 0.00004 0.00006]

[ 0.00033 0.0001 0.00011 0.00011 0.00013 0.00009 0.00009 0.00008

0.00021 0.00008 0.00008 0.0001 0.00005 0.00016 0.00003 0.00005

0.00008 0.00007 0.0001 0.00008 0.00006 0.00004 0.00008 0.00008

0.00008 0.0001 0.00004 0.00008 0.00004 0.00008]

[ 0.00085 0.00007 0.00007 0.00007 0.00008 0.00007 0.00006 0.00006

0.00008 0.00014 0.00006 0.00007 0.00005 0.00008 0.00003 0.00005

0.00006 0.00005 0.00008 0.00008 0.00005 0.00004 0.00005 0.00007

0.00006 0.00008 0.00004 0.00007 0.00006 0.00005]

[-0.00016 0.00007 0.00007 0.00008 0.00009 0.00008 0.00007 0.00006

0.00008 0.00006 0.00016 0.00008 0.00005 0.00008 0.00004 0.00004

0.00007 0.00006 0.00009 0.00006 0.00005 0.00004 0.00005 0.00006

0.00007 0.00008 0.00004 0.00005 0.00004 0.00006]

[ 0.00073 0.00011 0.00007 0.0001 0.00012 0.00012 0.00008 0.00007

0.0001 0.00007 0.00008 0.00025 0.00006 0.00009 0.00004 0.00004

0.00009 0.00006 0.00013 0.00007 0.00006 0.00004 0.00006 0.00007

0.00008 0.00009 0.00005 0.00007 0.00005 0.00007]

[ 0.00032 0.00005 0.00005 0.00006 0.00005 0.00006 0.00005 0.00004

0.00005 0.00005 0.00005 0.00006 0.0001 0.00005 0.00004 0.00004

0.00006 0.00006 0.00006 0.00005 0.00006 0.00004 0.00005 0.00006

0.00005 0.00005 0.00004 0.00006 0.00004 0.00005]

[ 0.0006 0.00008 0.0001 0.0001 0.00012 0.00009 0.00009 0.00007

0.00016 0.00008 0.00008 0.00009 0.00005 0.00017 0.00003 0.00005

0.00008 0.00007 0.00009 0.00007 0.00006 0.00004 0.00008 0.00008

0.00008 0.00009 0.00004 0.00007 0.00004 0.00008]

[ 0.00019 0.00003 0.00003 0.00004 0.00004 0.00004 0.00004 0.00003

0.00003 0.00003 0.00004 0.00004 0.00004 0.00003 0.00008 0.00004

0.00004 0.00004 0.00004 0.00004 0.00003 0.00005 0.00004 0.00003

0.00003 0.00004 0.00004 0.00004 0.00003 0.00003]

[ 0.00057 0.00005 0.00004 0.00005 0.00005 0.00005 0.00004 0.00004

0.00005 0.00005 0.00004 0.00004 0.00004 0.00005 0.00004 0.00011

0.00004 0.00004 0.00006 0.00005 0.00003 0.00004 0.00004 0.00004

0.00004 0.00005 0.00004 0.00004 0.00004 0.00004]

[ 0.00044 0.00007 0.00007 0.00009 0.00011 0.00008 0.00007 0.00006

0.00008 0.00006 0.00007 0.00009 0.00006 0.00008 0.00004 0.00004

0.00012 0.00006 0.00008 0.00006 0.00006 0.00004 0.00006 0.00006

0.00007 0.00007 0.00004 0.00006 0.00004 0.00006]

[ 0.00036 0.00005 0.00006 0.00006 0.00006 0.00006 0.00006 0.00005

0.00007 0.00005 0.00006 0.00006 0.00006 0.00007 0.00004 0.00004

0.00006 0.00015 0.00006 0.00005 0.00008 0.00004 0.00005 0.00006

0.00005 0.00006 0.00005 0.00006 0.00005 0.00006]

[ 0.001 0.00012 0.00008 0.00009 0.00011 0.00011 0.00008 0.00007

0.0001 0.00008 0.00009 0.00013 0.00006 0.00009 0.00004 0.00006

0.00008 0.00006 0.00021 0.00008 0.00007 0.00005 0.00006 0.00008

0.00008 0.00012 0.00005 0.00008 0.00005 0.00007]

[ 0.0008 0.00008 0.00007 0.00008 0.00008 0.00008 0.00005 0.00007

0.00008 0.00008 0.00006 0.00007 0.00005 0.00007 0.00004 0.00005

0.00006 0.00005 0.00008 0.00022 0.00005 0.00004 0.00005 0.00007

0.00006 0.00008 0.00004 0.00007 0.00005 0.00005]

[ 0.00034 0.00005 0.00005 0.00005 0.00006 0.00006 0.00005 0.00005

0.00006 0.00005 0.00005 0.00006 0.00006 0.00006 0.00003 0.00003

0.00006 0.00008 0.00007 0.00005 0.00012 0.00004 0.00005 0.00007

0.00005 0.00006 0.00004 0.00006 0.00004 0.00005]

[ 0.00025 0.00004 0.00003 0.00004 0.00004 0.00004 0.00004 0.00004

0.00004 0.00004 0.00004 0.00004 0.00004 0.00004 0.00005 0.00004

0.00004 0.00004 0.00005 0.00004 0.00004 0.00009 0.00004 0.00004

0.00004 0.00004 0.00004 0.00004 0.00004 0.00004]

[ 0.00043 0.00005 0.00005 0.00006 0.00007 0.00006 0.00006 0.00005

0.00008 0.00005 0.00005 0.00006 0.00005 0.00008 0.00004 0.00004

0.00006 0.00005 0.00006 0.00005 0.00005 0.00004 0.00011 0.00006

0.00006 0.00006 0.00004 0.00006 0.00004 0.00005]

[ 0.00095 0.00008 0.00007 0.00007 0.00007 0.00007 0.00006 0.00006

0.00008 0.00007 0.00006 0.00007 0.00006 0.00008 0.00003 0.00004

0.00006 0.00006 0.00008 0.00007 0.00007 0.00004 0.00006 0.00017

0.00006 0.00007 0.00004 0.00008 0.00005 0.00005]

[ 0.00019 0.00007 0.00007 0.00009 0.0001 0.00007 0.00006 0.00005

0.00008 0.00006 0.00007 0.00008 0.00005 0.00008 0.00003 0.00004

0.00007 0.00005 0.00008 0.00006 0.00005 0.00004 0.00006 0.00006

0.00013 0.00007 0.00004 0.00006 0.00004 0.00006]

[ 0.00101 0.0001 0.00008 0.00009 0.0001 0.00009 0.00007 0.00007

0.0001 0.00008 0.00008 0.00009 0.00005 0.00009 0.00004 0.00005

0.00007 0.00006 0.00012 0.00008 0.00006 0.00004 0.00006 0.00007

0.00007 0.00017 0.00003 0.00007 0.00004 0.00006]

[ 0.00023 0.00003 0.00003 0.00004 0.00004 0.00005 0.00005 0.00004

0.00004 0.00004 0.00004 0.00005 0.00004 0.00004 0.00004 0.00004

0.00004 0.00005 0.00005 0.00004 0.00004 0.00004 0.00004 0.00004

0.00004 0.00003 0.00012 0.00005 0.00004 0.00004]

[-0.00002 0.00007 0.00007 0.00007 0.00007 0.00007 0.00006 0.00006

0.00008 0.00007 0.00005 0.00007 0.00006 0.00007 0.00004 0.00004

0.00006 0.00006 0.00008 0.00007 0.00006 0.00004 0.00006 0.00008

0.00006 0.00007 0.00005 0.00026 0.00006 0.00005]

[ 0.00029 0.00004 0.00004 0.00005 0.00004 0.00006 0.00004 0.00004

0.00004 0.00006 0.00004 0.00005 0.00004 0.00004 0.00003 0.00004

0.00004 0.00005 0.00005 0.00005 0.00004 0.00004 0.00004 0.00005

0.00004 0.00004 0.00004 0.00006 0.00015 0.00004]

[-0.00006 0.00006 0.00005 0.00007 0.00011 0.00007 0.00013 0.00006

0.00008 0.00005 0.00006 0.00007 0.00005 0.00008 0.00003 0.00004

0.00006 0.00006 0.00007 0.00005 0.00005 0.00004 0.00005 0.00005

0.00006 0.00006 0.00004 0.00005 0.00004 0.00014]]

Step 5: **Group the assets into clusters using k-means clustering where k =15, which is the portfolio size selected by the investor.**

The Python code shows the invocation of the function KMeans from the scikit-learn library. The centroids (special points in multidimensional space) towards which the the other physical points representing the asset parameters gravitated to, based on their similarity measure and hence formed a cluster with the centroid as its nucleus, has been listed in the output. Observe that 15 centroids are obtained for $k=15$. Each point in the multi-dimensional space including the centroids, are of dimension 30. The labels indicate the cluster to which point $i$ among $N$ points or asset $i$ of the $N$-stock universe in reality, belong to.

In [20]:

*#kmeans clustering of assets using the characteristic vector of*

*#mean return and variance-covariance vector of returns*

assetsCluster**=** KMeans(algorithm**=**'auto', max\_iter**=**600, n\_clusters**=**clusters)

print('Clustering of assets completed!')

assetsCluster**.**fit(assetParameters)

centroids **=** assetsCluster**.**cluster\_centers\_

labels **=** assetsCluster**.**labels\_

print('Centroids:\n', centroids)

print('Labels:\n', labels)

Clustering of assets completed!

Centroids:

[[ 0.00029 0.00004 0.00004 0.00005 0.00004 0.00006 0.00004 0.00004

0.00004 0.00006 0.00004 0.00005 0.00004 0.00004 0.00003 0.00004

0.00004 0.00005 0.00005 0.00005 0.00004 0.00004 0.00004 0.00005

0.00004 0.00004 0.00004 0.00006 0.00015 0.00004]

[ 0.00099 0.0001 0.00008 0.00012 0.0001 0.00009 0.00007 0.00007

0.0001 0.00007 0.00007 0.0001 0.00006 0.00009 0.00004 0.00005

0.00008 0.00006 0.00013 0.00008 0.00006 0.00004 0.00006 0.0001

0.00008 0.00011 0.00004 0.00008 0.00005 0.00006]

[-0.00011 0.00007 0.00006 0.00008 0.0001 0.00008 0.0001 0.00006

0.00008 0.00006 0.00011 0.00008 0.00005 0.00008 0.00004 0.00004

0.00007 0.00006 0.00008 0.00005 0.00005 0.00004 0.00005 0.00005

0.00006 0.00007 0.00004 0.00005 0.00004 0.0001 ]

[ 0.00034 0.00005 0.00005 0.00006 0.00006 0.00006 0.00005 0.00004

0.00006 0.00005 0.00005 0.00006 0.00007 0.00006 0.00004 0.00004

0.00006 0.0001 0.00006 0.00005 0.00009 0.00004 0.00005 0.00006

0.00005 0.00006 0.00004 0.00006 0.00004 0.00005]

[ 0.00082 0.00008 0.00007 0.00008 0.00008 0.00008 0.00006 0.00006

0.00008 0.00011 0.00006 0.00007 0.00005 0.00007 0.00004 0.00005

0.00006 0.00005 0.00008 0.00015 0.00005 0.00004 0.00005 0.00007

0.00006 0.00008 0.00004 0.00007 0.00005 0.00005]

[ 0.00039 0.0001 0.0001 0.00013 0.00027 0.0001 0.00012 0.00007

0.00013 0.00008 0.00009 0.00012 0.00005 0.00012 0.00004 0.00005

0.00011 0.00006 0.00011 0.00008 0.00006 0.00004 0.00007 0.00007

0.0001 0.0001 0.00004 0.00007 0.00004 0.00011]

[ 0.00077 0.0001 0.00007 0.00009 0.00011 0.00015 0.00008 0.00007

0.0001 0.00007 0.00008 0.00018 0.00006 0.00009 0.00004 0.00005

0.00008 0.00006 0.00012 0.00008 0.00006 0.00004 0.00006 0.00007

0.00008 0.00009 0.00005 0.00007 0.00005 0.00007]

[ 0.00017 0.00007 0.00007 0.00009 0.00011 0.00008 0.00013 0.00006

0.00009 0.00006 0.00007 0.00008 0.00005 0.00008 0.00004 0.00004

0.00007 0.00005 0.00008 0.00006 0.00005 0.00004 0.00006 0.00006

0.0001 0.00007 0.00004 0.00006 0.00004 0.00009]

[ 0.00042 0.00006 0.00006 0.00007 0.00008 0.00007 0.00006 0.00008

0.00008 0.00006 0.00006 0.00007 0.00005 0.00008 0.00004 0.00004

0.00008 0.00005 0.00007 0.00006 0.00005 0.00004 0.00007 0.00006

0.00006 0.00007 0.00004 0.00006 0.00004 0.00006]

[ 0.00022 0.00003 0.00003 0.00004 0.00004 0.00004 0.00004 0.00004

0.00004 0.00004 0.00004 0.00004 0.00004 0.00004 0.00006 0.00004

0.00004 0.00004 0.00005 0.00004 0.00004 0.00006 0.00004 0.00003

0.00004 0.00004 0.00007 0.00004 0.00004 0.00004]

[ 0.00057 0.00005 0.00004 0.00005 0.00005 0.00005 0.00004 0.00004

0.00005 0.00005 0.00004 0.00004 0.00004 0.00005 0.00004 0.00011

0.00004 0.00004 0.00006 0.00005 0.00003 0.00004 0.00004 0.00004

0.00004 0.00005 0.00004 0.00004 0.00004 0.00004]

[-0.00002 0.00007 0.00007 0.00007 0.00007 0.00007 0.00006 0.00006

0.00008 0.00007 0.00005 0.00007 0.00006 0.00007 0.00004 0.00004

0.00006 0.00006 0.00008 0.00007 0.00006 0.00004 0.00006 0.00008

0.00006 0.00007 0.00005 0.00026 0.00006 0.00005]

[ 0.00031 0.00008 0.00014 0.00009 0.00011 0.00008 0.00008 0.00007

0.00016 0.00007 0.00007 0.00009 0.00005 0.00013 0.00003 0.00004

0.00007 0.00006 0.00009 0.00007 0.00006 0.00004 0.00007 0.00008

0.00008 0.00009 0.00004 0.00007 0.00004 0.00007]

[ 0.0006 0.00008 0.0001 0.0001 0.00012 0.00009 0.00009 0.00007

0.00016 0.00008 0.00008 0.00009 0.00005 0.00017 0.00003 0.00005

0.00008 0.00007 0.00009 0.00007 0.00006 0.00004 0.00008 0.00008

0.00008 0.00009 0.00004 0.00007 0.00004 0.00008]

[ 0.0009 0.00024 0.00007 0.0001 0.0001 0.0001 0.00007 0.00007

0.0001 0.00007 0.00007 0.00011 0.00005 0.00008 0.00003 0.00005

0.00007 0.00005 0.00012 0.00008 0.00005 0.00004 0.00005 0.00008

0.00007 0.0001 0.00003 0.00007 0.00004 0.00006]]

Labels:

[14 12 1 5 6 7 8 12 4 2 6 3 13 9 10 8 3 1 4 3 9 8 1 7

1 9 11 0 2]

Step 6: **Fix asset labels to points in each cluster**

In [21]:

*#fixing asset labels to cluster points*

print('Stocks in each of the clusters:\n',)

assets **=** np**.**array(assetLabels)

**for** i **in** range(clusters):

print('Cluster', i**+**1)

clt **=** np**.**where(labels **==** i)

assetsCluster **=** assets[clt]

print(assetsCluster)

Stocks in each of the clusters:

Cluster 1

['WMT']

Cluster 2

['BA' 'MSFT' 'UNH' 'V']

Cluster 3

['IBM' 'XOM']

Cluster 4

['JNJ' 'MRK' 'PFE']

Cluster 5

['HD' 'NKE']

Cluster 6

['CAT']

Cluster 7

['CSCO' 'INTC']

Cluster 8

['CVX' 'UTX']

Cluster 9

['DIS' 'MMM' 'TRV']

Cluster 10

['KO' 'PG' 'VZ']

Cluster 11

['MCD']

Cluster 12

['WBA']

Cluster 13

['AXP' 'GS']

Cluster 14

['JPM']

Cluster 15

['AAPL']

It can be seen that the 29 assets of the stock universe have been grouped into 15 clusters. The idea conveyed is that all assets in the same cluster behave similar (inter-cluster similarity) with regard to their mean and variance-covariance of returns, and are dissimilar (intra-cluster dissimilarity) with regard to the same characteristics with those assets in other clusters.

Therefore picking one asset from each cluster to gather a portfolio of 15 assets would ensure that the portfolio is well-diversified with regard to these characteristics. The choice could be random or preferential, but restricted to one asset from each cluster. For ease of reference, we term the choices made as $k$-portfolio. Needless to say, multiple $k$-portfolios can be generated from these clusters.

Since $k$-means clustering is a heuristic method which produces clusters that are sensitive to the randomly chosen initial centroids, it may yield different cluster configurations with each run. In practice, an aggregation of the clusters yielded during various runs can be studied to make the appropriate choice of assets one each from each cluster to construct the well-diversified portfolio.

## 3.5 $k$-portfolios - a brief note

Let us suppose that a specific run of the $k$-means clustering algorithm yielded the following clusters for the DJIA index "mini" universe: [KO PG VZ WMT], [UNH], [DIS MMM TRV], [IBM, XOM], [CSCO INTC], [JPM], [GS], [WBA], [AAPL MSFT V], [HD NKE], [AXP CVX UTX], [MCD],[JNJ MRK PFE], [BA], [CAT]

The investor is now free to make a choice of one asset each from each of the clusters. The choice could be random or guided by individual preferences. The following $k$-portfolios are some sample choices that can be made by investors.

$k$-portfolio 1:  
{ [KO], [UNH], [DIS], [IBM], [CSCO], [JPM], [GS], [WBA], [AAPL], [HD], [AXP], [MCD], [MRK], [BA], [CAT] }

$k$-portfolio 2:  
{ [VZ], [UNH], [MMM], [XOM], [INTC], [JPM], [GS], [WBA], [MSFT], [NKE], [CVX], [MCD], [PFE], [BA], [CAT]}

$k$-portfolio 3:  
{ [WMT], [UNH], [TRV], [IBM], [CSCO], [JPM], [GS], [WBA], [V], [NKE], [UTX], [MCD], [JNJ], [BA], [CAT]}

An investor can opt to invest in any one of the $k$-portfolios. Fig. 3.5 illustrates heuristic selection of portfolios where the portfolio of assets is diversified in behaviour adhering to the adage " never put all eggs in the basket".

Diagram

Description automatically generated

#### Fig. 3.5 Heuristic selection of assets to construct a diversified portfolio

Indeed, several questions do arise on the behavior of the $k$-portfolios, their risk-return tradeoffs and their performance when time tested portfolio construction techniques are applied over them. These shall be covered in detail in the ensuing lessons.

### Companion Reading

This work is an abridged adaptation of concepts discussed in Chapter 3 of [PAI 18] to Dow Jones dataset (DJIA Index: April, 2014-April, 2019) and implemented in Python using NumPy, Pandas and scikit-learn libraries. Readers (read "worker bees") seeking more information may refer to the corresponding chapter in the book.

### References

[CHO 08] Choueifaty Yves and Y Coignard, Toward Maximum Diversification, The Journal of Portfolio Management, pp. 40-51, 2008.

[CHO 13] Choueifaty Yves, T Froidure and J Reynier, Properties of the Most Diversified Portfolio, Journal of Investment Strategies, 2(2), pp. 49-70, 2013.

[PAI 18] Vijayalakshmi Pai G. A., Metaheuristics for Portfolio Optimization- An Introduction using MATLAB, Wiley-ISTE, 2018. <https://www.mathworks.com/academia/books/metaheuristics-for-portfolio-optimization-pai.html>

# 130-30 Portfolio Construction using Metaheuristics

Text, letter

Description automatically generated

# 1. What is a 130-30 portfolio?

130-30 strategy is a portfolio construction method which ensures investment exposure and market protection at the same time. The strategy adopts leveraging by shorting poor performing stocks to the tune of 30% of the portfolio value and diverting the funds to invest in the long, on better performing stocks to the tune of 130% of the portfolio value. 130-30 portfolios are long-short portfolios and by and large have been observed to perform better than long-only portfolios.

# 2. 130-30 Portfolio Construction

Let us consider an investor who desires to construct a 130-30 portfolio, optimal with regard to an objective function and subject to constraints that reflect restrictions on asset allocation and the investor's preferences.

(1) The objective is to maximize Sharpe Ratio of the portfolio.

(See <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson6_SharpeRatioOptimization/Lesson6_MainContent.ipynb> to know about Sharpe Ratio based portfolio optimization)

The following constraints are imposed on the portfolio by the investor:

(2) A budget constraint on the portfolio, where the capital is fully invested, (i.e.) the sum of the portfolio

weights equals 1.

(3) A budget constraint of 130% on the capital invested in the long positions, which automatically implies a

budget constraint of 30% on the capital invested in the short positions of the portfolio.

(4) Leveraged bounds on each long position to lie between [0, 1.3]

(5) Bounds on short positions to lie between [-0.3,0]

(6) Portfolio beta needs to be 1 to ensure that the volatility of the portfolio matches with that of the

market.

# 3. Mathematical Formulation of the Investor's 130-30 Portfolio Model

Let us suppose that the investor desires to construct a portfolio P comprising assets $A\_1, A\_2, ...A\_N$, with $\bar \mu\_{(1 X N)} = [\mu\_1, \mu\_2, ...\mu\_N]$ as the asset returns and $\bar W\_{(N X1)}=[W\_1, W\_2, ...W\_N]'$ as the weights.

Let $R\_f$ be the risk free rate of return.

The portfolio return $r$ determined by a weighted summation of its individual asset returns is given by $\bar \mu . \bar W = \sum\left({\mu\_i.W\_i}\right)$ and the risk is given by $\sqrt{{\bar W}'.V.\bar W}$, where V is the variance-covariance matrix of returns.

With regard to portfolios with equity stocks, portfolio beta is the weighted sum of the asset betas,where the weights are represented by the portfolio weights. Portfolio beta is given by, $\sum{\beta\_i.W\_i}$, where $\beta\_i$ are the asset betas and $W\_i$ are the portfolio weights.

(See <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson1_FundaRiskReturnPortfolio/Lesson1_MainContent.ipynb>, to know about risk, return and asset betas of a portfolio)

The mathematical model for the 130-30 strategy based Portfolio Optimization whose objective function and constraints were defined in the previous section is as shown below:

1.Objective function for maximizing Sharpe Ratio of the portfolio



subject to the constraints that describe constraints (2)-(6) listed in Sec. 2. viz.,

Shape

Description automatically generated with medium confidence

In the above system of equations, $W^+$ and $W^-$ indicate the weights of the long and short positions of the optimal portfolio. Equations [1.3] to [1.5] illustrate the budget and leveraged bound constraints imposed on the long and short positions. Equation [1.6] denotes the portfolio beta constraint and equation [1.2] describes a fully invested long-short portfolio.

## 3.1 Long-Plus-Short Portfolio vs Integrated Long-Short Portfolio

The 130-30 portfolio model discussed here, is not a mere combination of a long-only portfolio with a short-only portfolio, where the investor earmarks some positions in the portfolio to be longed and the rest to be shorted. Jacobs et al., [JAC 1999] were critical of this approach and while terming it as a long-plus-short portfolio, asserted that such a portfolio has lesser merits when compared to a real long-short portfolio that undertakes an integrated combination of long and short positions. They claimed that a long-short portfolio construction is not a two-portfolio strategy but "...a one-portfolio strategy in which the long and short positions are determined jointly within an optimization that takes into account the expected returns of the individual securities, the standard deviation of those returns and the correlations between them, as well as the investor's tolerance for risk" [JAC 1999].

The strategy adopted to construct such a long-short 130-30 portfolio is evolved using metaheuristics. The metaheuristic strategy undertakes an integrated optimization of the assets in the portfolio to determine the long and short positions of the portfolio that yield maximal Sharpe Ratio, subject to all the inherent constraints laid down by the investment strategy as well as the investor.

# 4. Metaheuristic solution strategy

We proceed to undertake the integrated optimization of the 130-30 portfolio model described by equations [1.1] to [1.6] employing a robust metaheuristic strategy viz., Differential Evolution with Hall of Fame (DE HOF). DE HOF dynamically arrives at the optimal composition of long and short positions of the portfolio that yield the maximal Sharpe Ratio, subject to all the constraints enforced over the assets and the portfolio.

(See Chapter 2: A Brief Primer on Metaheuristics [Pai 2018], to know more about metaheuristics and Differential Evolution)

Metaheuristic strategies build feasible solution sets that satisfy the constraints imposed, during the course of their execution process. However constraint handling has been a major problem in the application of metaheuristic strategies to complex constrained optimization problems, to tackle which several methods such as Penalty Function Strategy and Repair Strategy have been proposed.

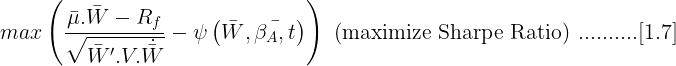
For the 130-30 portfolio optimization, DE HOF makes use of Joines and Houck's Dynamic Penalty Function Strategy [JOI 1994] to handle the portfolio beta constraint (equation [1.6]) and Repair Strategies to handle the budget constraints on the portfolio and long positions as well as the leveraged bounds constraints on the long and short positions (equations [1.2] to [1.5]).

Repair strategies are custom made to suit the requirements of the problem and evolving such a strategy that will help satisfy one or more constraints at one go, can turn out to be difficult. Nevertheless, once the strategy is evolved, it can help churn out populations of feasible solution sets leading to faster convergence of the metaheuristic strategy.

(See Sec. 6.4.2 of Chapter 6: Metaheuristic 130-30 Portfolio Construction [Pai 2018], to know about the Repair Strategies evolved for the 130-30 portfolio optimization model)

# 5. Transformation of the Mathematical Model

Joines and Houck's dynamic penalty function strategy is used to tackle the portfolio beta constraint represented by equation [1.6]. The portfolio beta constraint is accomodated in the "penalized" objective function by defining appropriate penalty functions. The transformed mathematical model is shown below.



where $ \psi \left ( \bar{W}, {\bar{\beta}\_A}, t \right )$ is the constraint violation function that tackles the portfolio beta constraint using dynamic penalty functions and is defined as shown below.

Shape

Description automatically generated with medium confidence

In the system of equations [1.8], $(C, \alpha, \beta)$ are all constants and the penalty term $(C.t)^\alpha$ increases constantly with each generation count t of the metaheuristic strategy DE HOF. $\varphi\left ( \bar{W}, \bar{\beta\_A}\right )$ tackles the portfolio beta constraint as $ \left | \sum\_{k=1}^{N} \left ( \beta \_A^k.W\_k \right )-1\right | \leq \epsilon$, for a tolerance limit of $\epsilon$.

DE HOF now strives to solve the optimization model whose penalized objective function is described by equations [1.7]-[1.8] subject to the budget and leveraged bounds constraints imposed on the long and short positions of the portfolio, described by equations [1.2]-[1.5] only.

To reiterate, DE HOF employs repair strategies to tackle these constraints resulting in faster convergence.

# 6. Differential Evolution with Hall of fame - a brief roundup

DE HOF is a population based metaheuristic strategy that can efficiently solve complex constrained optimization problems. We restrict the discussion to the DE HOF designed to solve the 130-30 portfolio optimization problem model. The input, process and output of the DE HOF strategy are as follows:

### Inputs

It is essential that the portfolio parameters and the DE HOF strategy parameters are clearly set before the optimization process begins.

The portfolio parameters are (1) assets in the portfolio (2) asset betas (3) mean asset returns (4) covariance of returns (the variance-covariance matrix of returns) and (5) risk free rate of return.

The DE HOF parameters are (1) population size (2) number of generations (3) dynamic penalty function parameters $(C, \alpha, \beta)$ (4) Scale factor $\beta\_S$ and (5) probability of recombination $p\_r$.

### Process

DE HOF begins its execution by generating an initial random population of individuals that represent the candidate solution sets to the optimization problem concerned. Each individual in the population represents a collection of N genes that represent random weights generated in the range [-0.3, 1.3] for the N assets in the portfolio.

The candidate solution sets are transformed into feasible solution sets by calling the weight repair strategies which repair each individual in the population, so that it satisfies constraints represented by equations [1.2] to [1.5]. At the end of the process the initial population is transformed into a population that is a feasible solution set and is termed as the parent population

Typical of metaheuristic strategies, DE HOF now computes the fitness function values of each individual in the parent population making use of the penalized objective function described by equations [1.7]-[1.8].

Making use of mutation and crossover operators ( Rand4/Best/Dir5 mutation operator and Binomial Crossover operator) and the scale factor $\beta\_A$, DE HOF generates what are called trial vectors which eventually leads to the production of the offspring population of individuals. (See [FEO 2006] to know more about Rand4/Best/Dir5 operator)

The offspring population of individuals is standardized to satisfy the respective constraints, by calling the weight repair strategies once again and their fitness function values are computed as was done for the parent population.

Based on the fitness function values of the individuals in the parent population and offspring population, the best fit individuals are selected using the Deterministic Selection operator and pushed to the next generation. The best among the better individuals selected for the next generation with maximal fitness function value and zero penalty function value (no constraints violated) is inducted into the Hall of Fame.

At the end of the first generation, the next generation pool of individuals get set as the parent population and the second generation begins. The second generation offspring population is now generated by repeating the reproduction process and the best among the better individuals selected for the third generation competes with the individual in the Hall of Fame. DE HOF ensures that only the best individual by way of maximal fitness function value and zero penalty function value generated this far, is inducted into the Hall of Fame.

The generation cycles progress until the termination criterion is met with, at which stage the individual in the Hall of Fame is declared to be the optimal solution to the problem concerned. Since it is integrated optimization, those positions that are to be longed or shorted to arrive at the optimal 130-30 portfolio, is known only at this stage.

### Output

The genes of the HOF individual represent the optimal weights or proportion of capital to be invested in the assets of the portfolio. Positive weights indicate assets that are to be longed and negative weights indicate assets that are to be shorted. Given the mean returns and the covariance of returns, the risk and return of the optimal 130-30 portfolio can be easily computed. It can be verified that all the long and short positions in the optimal portfolio satisfy their respective budgets and leveraged bounds constraints and the portfolio beta equals 1, ensuring that the volatility of the optimal 130-30 portfolio matches that of the market.

(See Sec. 6.4.3 of Chapter 6 Metaheuristic 130-30 Portfolio Construction [PAI 2018] to know more about the design, process flow chart and execution of Differential Evolution with Hall of Fame based 130-30 Portfolio construction)

# 7. Case Study

We proceed to demonstrate the construction of 130-30 long-short portfolios over S&P BSE200 (April 2009- July 2020) data set of Bombay Stock Exchange, India, which includes the Covid 19 crisis period as well. To keep the story short, we assume that the investor has already made a technically diverse choice of assets in the portfolio (a k-portfolio, in fact) and is ready with the mean returns $\mu$, the variance-covariance matrix of returns V and the betas of the assets constituting the portfolio.

Note that a k-portfolio is an outcome of a heuristic portfolio selection strategy, where the universe of stocks is grouped into clusters that display intra-class similarity and inter-class dissimilarity with regard to the mean-returns and covariance of returns. Since assets belonging to a cluster are similar in behavior, the investor now makes a choice of one asset each from each cluster to ensure diversification of assets in the portfolio. A clustering technique such as k-means algorithm can be used to group the stock universe into k clusters with the investor exercising the choice of k.

(See <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson3_HeuristicPortfolioSelection/Lesson3_MainContent.ipynb> and Chapter 3 Heuristic Portfolio Selection[PAI 2018], to know more about the construction of k-portfolios and their merits)

Let the k-portfolio selected by the investor comprise the following 20 assets, after making a heuristic portfolio selection for k = 20:

Ajanta Pharma Ltd. ["'AJANTPHARM'"], Amara Raja Batteries Ltd.["'AMARAJABAT'"], Ashok Leyland Ltd.["'ASHOKLEY'"], Bharat Forge Ltd. ["'BHARATFORG'"], Cipla Ltd. ["'CIPLA'"], GAIL (India) Ltd. ["'GAIL'"], HDFC Bank Ltd.["'HDFCBANK'"], Hindustan Petroleum Corporation Ltd. [ "'HINDPETRO'"], Vodafone Idea Ltd. ["'IDEA'"], Indraprastha Gas Ltd. ["'IGL'"], Indian Oil Corporation Ltd.[ "'IOC'"], ITC Ltd. ["'ITC'"], JSW Steel Ltd.[ "'JSWSTEEL'"], Motilal Oswal Financial Services Ltd. ["'MOTILALOFS'"], Petronet LNG Ltd. ["'PETRONET'"], State Bank of India Ltd. ["'SBIN'"], Shree Cement Ltd. ["'SHREECEM'"], Tata Steel Ltd. ["'TATASTEEL'"], Tata Consultancy Services Ltd. ["'TCS'"], Wipro Ltd. ["'WIPRO'"]

The objective is to construct a 130-30 portfolio that will yield maximal Sharpe Ratio subject to the constraints listed in equations [1.2]-[1.6].

A fragment of the CSV file S&PBSE200\_kPortfolio.csv, which describes the asset labels, mean returns, variance-covariance matrix of returns and asset betas, to be used by DE HOF for the construction of optimal 130-30 portfolio is shown below:

Table

Description automatically generated

#### Fig.1.1 Structure of the input file to DE HOF that captures details about assets in the portfolio, their mean and covariance of returns (daily %) and asset betas

# 8. Python coding of DE HOF for 130-30 Portfolio Construction

The DE HOF program is a conglomeration of functions typical of any metaheuristic strategy. The functions are listed first followed by the main program, with a brief description of the task accomplished by the function code.

### 8.1 Function WeightStdzn130\_30BoundsConstr

This function implements Phase 1 of the weight repair strategy that DE HOF uses to standardize or repair the weights of a population individual, so that all the asset weights lie between [-0.3, 1.3] and their sum equals 1. That is, the budget constraint on the portfolio described by equation [1.2] and the bounds stipulated for a 130-30 portfolio which is [-0.3, 1.3] are met with before Phase 2 of the weight repair strategy, proceeds to repair them further to make them satisfy the constraints described by equations [1.3]-[1.5].

The population of individuals each of which represents the weights of the N assets in the portfolio is in reality an array (InpWeightMat) of size (Population Size X N). LowUpBounds is an array of size (2 X N) where LowUpBounds[0, :] is initialized to -0.3 and LowUpBounds[1,:] is initialized to 1.3.

The function returns the standardized weight matrix StdWeightMatrix, which represents the population of individuals whose weights lie in the interval [-0.3, 1.3] and satisfy the portfolio budget constraint of sum of weights equaling 1 (equation [1.2])

In [1]:

"""

Weight Repair Strategy - Phase 1

-----------------------------------------------------------------------------

Standardization of weights of each individual in the population so that

they satisfy the portfolio budget constraint of

sum of weights equals 1 while each weight lies between [-0.3, 1.3]

Reference: Sec. 6.4.2, Chapter 6 Metaheuristic 130-30 Portfolio Construction of

[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** WeightStdzn130\_30BoundsConstr(InpWeightMat, LowUpBounds):

*#dependencies*

**import** numpy **as** np

WeightMatrix **=** InpWeightMat

[RowsMat, ColsMat]**=**np**.**shape(WeightMatrix)

[LowBound, UpBound] **=** np**.**shape(LowUpBounds)

*# Steps 1 and 2 of Weight Repair Strategy Phase 1*

*# Standardize weights to satisfy their lower bounds while ensuring*

*# that their sum equals 1*

**for** i **in** range(RowsMat):

*#R: those weights which are less than their respective lower bounds*

R **=** []

**for** j **in** range(ColsMat):

**if** (WeightMatrix[i,j]**<** LowUpBounds[0,j]):

WeightMatrix[i,j]**=** LowUpBounds[0,j]

R**.**append(j)

*#Q: those weights which satisfy their lower bounds*

Q **=** list(set(range(ColsMat))**-**set(R))

F **=** 1.0 **-** np**.**sum(LowUpBounds[0,R])**-** np**.**sum(LowUpBounds[0,Q])

L **=** np**.**sum(np**.**abs(WeightMatrix[i,Q]))

**if** (L**==**0):

term **=** F **/** len(Q)

WeightMatrix[i,Q]**=** LowUpBounds[0,Q]**+** term

**else**:

term **=** F **/** L

WeightMatrix[i,Q] **=** LowUpBounds[0,Q]**+** np**.**abs(WeightMatrix[i,Q])**\*** term

*# Steps 3-6 of Weight Repair Strategy Phase 1*

*# standardize upper bounds so that weights ultimately satisfy both upper*

*# and lower bounds while their sum equals 1*

**for** i **in** range(RowsMat):

r **=** []

ExitFlag **=** **True**

q **=** list(set(range(ColsMat))**-**set(r))

**while** (ExitFlag **==** **True**):

ExitFlag **=** **False**

**for** j **in** range(len(q)):

**if** ( WeightMatrix[i, q[j]] **<=** LowUpBounds[1, q[j]] ):

**continue**

**else**:

ExitFlag **=** **True**

r**.**append(q[j])

q **=** list(set(range(ColsMat))**-**set(r))

**if** ( ExitFlag **==** **True**):

L **=**np**.**sum(np**.**abs(WeightMatrix[i,q]))

F **=** 1.0 **-** ( np**.**sum(LowUpBounds[0,q])**+** np**.**sum( LowUpBounds[1,r]) )

**if** (L**==**0):

term **=** F

WeightMatrix[i,q[0]]**=** term

**else**:

term **=** F**/**L

WeightMatrix[i,q] **=** LowUpBounds[0,q] **+** (np**.**abs(WeightMatrix[i,q])**\*** term)

WeightMatrix[i,r]**=**LowUpBounds[1,r]

StdWeightMatrix **=** WeightMatrix

**return** StdWeightMatrix

### 8.2 Function WeightStdzn130\_30BudgetConstr

This function implements Phase 2 of the Weight Repair Strategy. The output of Phase 1 of the Weight Repair Strategy is fed as input to this function. This population of individuals is now repaired to satisfy the leveraged bounds and budget constraints on the long and short positions of the portfolio described by equations [1.3]-[1.5]. The output array StdzWeightMatrix of this function represents a feasible solution set to the transformed mathematical model, that satisfies all the constraints described by equations [1.2]-[1.5].

In [2]:

"""

Weight Repair Strategy - Phase 2

-----------------------------------------------------------------------------

Pre-requisite: The input population of individuals to this function

must have undergone Weight Repair Strategy Phase 1 processing, implemented

as function WeightStdzn130\_30BoundsConstr.

Standardization of weights of each individual in the population undertaken so that

they satisfy the budget and leveraged bounds constraints on long positions and

bounds constraints on short positions, subject to the fully invested

constraint of sum of weights equals 1.

Reference: Sec. 6.4.2, Chapter 6 Metaheuristic 130-30 Portfolio Construction of

[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** WeightStdzn130\_30BudgetConstr(InputWeightMatrix, LongLowUpBounds, ShortLowUpBounds):

*#dependencies*

**import** numpy **as** np

WeightMatrix **=** InputWeightMatrix

[RowsMat, ColsMat]**=** np**.**shape(WeightMatrix)

StdzWeightMatrix **=** np**.**zeros([RowsMat, ColsMat], dtype **=** float)

*# budgets for long positions*

eta **=** 1.3

gamma **=** 0

**for** p **in** range(RowsMat):

*# identify long and short positions in H, where H[0] denotes*

*# indices of long positions and H[1] denotes indices of short positions*

H **=** GroupAssets130\_30(ColsMat, WeightMatrix[p,:])

*# Adjust weights representing long positions*

DepositWeights **=** 0.0

h **=** 0

SumWeights **=** np**.**sum(WeightMatrix[p,H[h]])

**if** ((SumWeights **<=** eta ) **&** (SumWeights **>=** gamma)):

**continue**

**else**:

DepositWeights **=** DepositWeights **+** (SumWeights**-**eta)

MP **=** eta

WeightMatrix[p,:] **=** StdznAssetWeights130\_30LongPositions(WeightMatrix[p,:], H[h], SumWeights, MP, LongLowUpBounds)

*# Adjust weights representing short positions*

**if** (DepositWeights **==**0):

**continue**

**else**:

h **=** 1

SumWeights **=** np**.**sum(WeightMatrix[p,H[h]])

AbsSumWeights **=** np**.**sum(np**.**abs(WeightMatrix[p,H[h]]) )

DepositWeights **=** DepositWeights **+** SumWeights

WeightMatrix[p,:] **=** StdznAssetWeights130\_30ShortPositions(WeightMatrix[p,:], H[h], AbsSumWeights, DepositWeights, ShortLowUpBounds)

StdzWeightMatrix **=** WeightMatrix

**return** StdzWeightMatrix

### 8.2.1 Function GroupAssets130\_30

Since DE HOF undertakes integrated optimization of long-short portfolio, there is no knowing which assets are long or short in each individual of the population. This sub function groups assets in each individual of the population into long and short classes, so that the respective budgets and leveraged bounds can be applied on the respective classes.

In [3]:

"""

Group assets into long and short positions, given a chromosome of weights

-------------------------------------------------------------------------

@author: Dr G Vijayalakshmi Pai

"""

**def** GroupAssets130\_30(PortfolioSize, chromosome):

*#dependencies*

**import** numpy **as** np

*#initialization*

LongPositions **=** []

ShortPositions **=** []

**for** c **in** range(PortfolioSize):

**if** (chromosome[c] **>=** 0):

LongPositions**.**append(c)

**else**:

ShortPositions**.**append(c)

GroupAssets **=** [LongPositions, ShortPositions]

**return** GroupAssets

### 8.2.2 Function StdznAssetWeights130\_30LongPositions

This sub function takes the weight vector of each individual in the population and repairs them so that the long positions of the corresponding weight vector gathered in M, satisfy the constraints imposed on them viz., the leveraged bounds and budget constraint represented by equations [1.3] and [1.4], subject to the constraint represented by equation [1.2]. TW denotes the budget imposed on the long positions.

In [4]:

"""

Weight Repair Strategy - Phase 2

-----------------------------------------------------------------------------

Sub function:

The weights of Long positions of an individual in the population

are repaired, to satisfy their respective constraints of leveraged bounds

and budget.

Reference: Sec. 6.4.2, Chapter 6 Metaheuristic 130-30 Portfolio Construction of

[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** StdznAssetWeights130\_30LongPositions(InputWeightVector, M, WeightSum, TW, LowUpBounds):

*#dependencies*

**import** numpy **as** np

WeightVector **=** InputWeightVector

*# adjust lower bounds of assets*

L **=** WeightSum

F **=** TW **-** np**.**sum(LowUpBounds[0,M])

term **=** F **/**L

WeightVector[M] **=** LowUpBounds[0,M]**+** (WeightVector[M]**\*** term)

*# adjust upper bounds of assets*

r **=** []

ExitFlag **=** **True**

q **=** M

**while** (ExitFlag **==** **True**) :

ExitFlag **=** **False**

**for** j **in** range(len(q)):

**if** (WeightVector[q[j]]**>** LowUpBounds[1,q[j]]):

ExitFlag **=** **True**

r**.**append(q[j])

q **=** list(set(M)**-**set(r))

**if** (ExitFlag **==** **True**):

L **=** np**.**sum(WeightVector[q])

F **=** TW **-** (np**.**sum(LowUpBounds[0,q])**+** np**.**sum( LowUpBounds[1, r]) )

term **=** F**/**L

WeightVector[q] **=** LowUpBounds[0,q] **+** (WeightVector[q]**\*** term)

WeightVector[r] **=** LowUpBounds[1,r]

**return** WeightVector

### 8.2.3 Function StdznAssetWeights130\_30ShortPositions

This sub function takes the weight vector whose long position weights have been repaired and proceeds to do a similar activity with regard to the weights of the short positions indicated by M. TW denotes the budget imposed on the short positions.

In [5]:

"""

Weight Repair Strategy - Phase 2

-----------------------------------------------------------------------------

Sub function:

The weights of Short positions of an individual in the population

are repaired, to satisfy their respective constraints of leveraged bounds

and budget.

Reference: Sec. 6.4.2, Chapter 6 Metaheuristic 130-30 Portfolio Construction of

[PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** StdznAssetWeights130\_30ShortPositions(InpWeightVector, M, AbsWeightSum, TW, LowUpBounds):

*#dependencies*

**import** numpy **as** np

WeightVector **=** InpWeightVector

*# adjust lower bounds of assets, where TW is the budget imposed on*

*# short positions*

L **=** AbsWeightSum

F **=** TW **-** np**.**sum(LowUpBounds[0,M])

**if** (L**==**0):

term **=** F**/**len(M)

WeightVector[M] **=** LowUpBounds[0,M]**+** term

**else**:

term **=** F **/**L

WeightVector[M] **=** LowUpBounds[0, M]**+** (np**.**abs(WeightVector[M])**\*** term)

*# adjust upper bounds of assets*

r **=** []

ExitFlag **=** **True**

q **=** M

**while** (ExitFlag **==** **True**):

ExitFlag **=** **False**

**for** j **in** range(len(q)):

**if** (WeightVector[q[j]] **>** LowUpBounds[1,q[j]]):

ExitFlag **=** **True**

r**.**append (q[j])

q **=** list(set(M)**-**set(r))

**if** (ExitFlag **==** **True**):

L **=** np**.**sum(np**.**abs(WeightVector[q]))

F **=** TW **-** (np**.**sum(LowUpBounds[0,q])**+** np**.**sum(LowUpBounds[1,r]) )

**if** (len(q) **<>**0):

**if** (L**==**0):

term **=** F

WeightVector[q[0]]**=** term

**else**:

term **=** F**/**L

WeightVector[q] **=** LowUpBounds[0,q] **+** (np**.**abs(WeightVector[q])**\*** term)

WeightVector[r] **=** LowUpBounds[1, r]

**return** WeightVector

### 8.3 Function ConstrViolnFunction130\_30

This function computes the constraint violation function defined by equation [1.8] and returns $\varphi\left ( \bar{W}, \bar{\beta\_A}\right )$ and G as output (optional).

In [6]:

"""

Constraint violation function for 130-30 portfolio construction described by

equation [1.8] in the tranformed mathematical model.

Reference: Chapter 6 Metaheuristic 130-30 Portfolio Construction of [PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** ConstrViolnFunction130\_30( WeightMat, BetasAssets, C\_param, AlphaParam, beta\_param, GenerationCount):

*#dependencies*

**import** numpy **as** np

[RowMat, ColMat]**=** np**.**shape(WeightMat)

epsilon **=** 0.001

G **=** np**.**zeros(shape **=**(RowMat))

psi **=** np**.**zeros(shape **=** (RowMat))

**for** i **in** range(RowMat):

ChromosomeX **=** WeightMat[i,:]

*# compute penalty function G*

PortfolioBetaTerm **=** np**.**sum(np**.**multiply(BetasAssets, ChromosomeX))

g1Term **=** np**.**abs((PortfolioBetaTerm) **-** 1.0)**-**epsilon

**if** (g1Term **<=**0 ):

G[i]**=**0

**else**:

G[i]**=**1

*#compute constraint violation function*

PenaltyTerm**=** np**.**power((C\_param **\*** GenerationCount), AlphaParam)

psi[i] **=** PenaltyTerm **\***( G[i] **\*** np**.**power(g1Term, beta\_param))

**return** [psi, G]

### 8.4 Function ComputeFitness130\_30

This function computes the fitness function values of the population using the penalized objective function described by equation [1.7].

In [7]:

"""

Compute fitness function values for the population of individuals using the

penalized objective function defined by equation [1.7]

Reference: Chapter 6 Metaheuristic 130-30 Portfolio Construction of [PAI, 2018]

[PAI, 2018] G A Vijayalakshmi Pai, Metaheuristics for Portfolio Optimization-An

Introduction using MATLAB, ISTE-Wiley, 2018.

MATLAB Version

https://in.mathworks.com/matlabcentral/profile/authors/2806050-dr-g-a-vijayalakshmi-pai

----------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** ComputeFitness130\_30(PoplnMat, ReturnData, CovarianceData, RiskFreeData, PsiFunction):

*#dependencies*

**import** numpy **as** np

[popln\_size, cols]**=** np**.**shape(PoplnMat)

weight **=** np**.**zeros(cols)

PoplnFitness **=** np**.**zeros(popln\_size)

**for** i **in** range(popln\_size):

weight **=** PoplnMat[i,:]

PoplnFitness[i] **=** ((np**.**matmul(ReturnData, weight**.**T)**-**RiskFreeData)**/**np**.**sqrt(np**.**matmul( np**.**matmul(weight, CovarianceData), weight**.**T)) ) **-** PsiFunction[i]

**return** PoplnFitness

### 8.5 Function DEOperatorRand4BestDir5

This function implements the standard Rand4BestDir5 operator of Differential Evolution strategy, which helps generate trial vectors.

(See [FEO 06] to learn more about Rand4BestDir5 operator)

In [8]:

"""

Differential Evolution's Rand4BestDir5 operator to generate trial vector population

Reference: Vitaliy Feoktistov, Differential Evolution in Search of Solutions, Springer, 2006.

-----------------------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** DEOperatorRand4BestDir5(popln, FitnessValue, BetaValue, PoplnSize, IndividualLength):

*#dependencies*

**import** numpy **as** np

**import** array **as** arr

*#initialization*

TrialVectorPopln **=** np**.**zeros([PoplnSize, IndividualLength])

**for** i **in** range(PoplnSize):

DifferentialVecIndex **=** [0]**\***5

V **=** np**.**zeros([4,IndividualLength])

Vb **=** np**.**zeros(IndividualLength)

*#set IND the current individual in the population indicated by i*

IND **=** popln[i,:]

*# prepare RandomIndex, random number indices for each population*

*# individual to enable it choose five random individuals from the*

*# population, without repeating itself.*

RandomIndex **=** arr**.**array('i', np**.**random**.**permutation(PoplnSize))

RandomIndex**.**remove(i)

*# select five random individuals from the population*

**for** u **in** range (5):

DifferentialVecIndex[u] **=** int(RandomIndex[u])

*# Obtain Vb the best individual with the*

*# maximal objective function value and designate the rest as array V*

individuals **=** [FitnessValue[DifferentialVecIndex[0]], FitnessValue[DifferentialVecIndex[1]], FitnessValue[DifferentialVecIndex[2]], FitnessValue[DifferentialVecIndex[3]], FitnessValue[DifferentialVecIndex[4]]]

max\_obj\_indx **=** individuals**.**index (np**.**max(individuals))

j **=** 0

**for** z **in** range(5):

**if** (DifferentialVecIndex[z] **==** DifferentialVecIndex[max\_obj\_indx] ):

Vb **=** popln[DifferentialVecIndex[z], : ]

**else**:

V[j,:] **=** popln[DifferentialVecIndex[z],: ]

j**=**j**+**1

*# obtain trial vector for each of the parent vector individual*

TrialVectorPopln[i,:] **=** Vb **+** BetaValue**/**5 **\*** ( 5**\***Vb **-** IND **-** V[0,:]**-**V[1,:]**-**V[2,:]**-**V[3,:])

**return** TrialVectorPopln

### 8.6 Function DEOperatorBinCrossOver

This function implements the standard Binomial Crossover operator of Differential Evolution.

In [9]:

"""

Differential Evolution Binomial Cross over operator

----------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** DEOperatorBinCrossover (ParentPopln, TargetVecPopln, ProbabilityRecombn, Components):

*#dependencies*

**import** numpy **as** np

[row, col] **=** np**.**shape(ParentPopln)

tau **=** DEComputeTau (Components, ProbabilityRecombn)

OffspringPopln **=** np**.**empty(shape **=**(row, col), dtype **=** float)

**for** i **in** range(col):

**if** i **in** tau:

OffspringPopln[:, i]**=** TargetVecPopln[:,i]

**else**:

OffspringPopln[:, i] **=** ParentPopln[:,i]

**return** OffspringPopln

### 8.6.1 Function DEComputeTau

This sub function computes a parameter (Tau) of the standard Binomial Crossover operator of Differential Evolution algorithm.

In [10]:

"""

This function computes Tau, a parameter required by Differential Evolution's Binomial Crossover operator

--------------------------------------------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** DEComputeTau(GeneSize, ProbabilityRecombn):

*#dependencies*

**import** numpy **as** np

**import** random

h **=** list(np**.**random**.**permutation(range(GeneSize)))

*#set jStar to a random index*

jStar **=** h[0]

tau**=**[jStar]

h**.**remove(jStar)

**for** i **in** range(GeneSize**-**1):

**if** (random**.**random()**<** ProbabilityRecombn):

tau**.**append(h[i])

**return** tau

### 8.7 Function DEOperatorDetermSelection

This function implements the standard Deterministic Selection operator of Differential Evolution, which selects the best fit amongst the parent and offspring population and prepares the population of individuals for the next generation.

In [11]:

"""

The standard Deterministic Selection operator of Differential Evolution

-----------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

**def** DEOperatorDetermSelection(FeasParentPopln, FeasParentPoplnFitness,PsiParent, FeasOffsprngPopln,FeasOffsprngPoplnFitness, PsiOffsprng, PoplnSize):

*#dependencies*

**import** numpy **as** np

[row, col] **=** np**.**shape(FeasParentPopln)

*#initialization*

NextGenPool **=** np**.**zeros(shape **=** (PoplnSize, col), dtype **=** float)

NextGenPoolFitness **=** np**.**zeros(shape **=** (PoplnSize), dtype **=** float)

NextGenPoolPsi **=** np**.**zeros(shape **=** (PoplnSize), dtype **=** float)

**for** i **in** range (PoplnSize):

**if** (FeasParentPoplnFitness[i] **>=** FeasOffsprngPoplnFitness[i]):

NextGenPool[i,:]**=** FeasParentPopln[i,:]

NextGenPoolPsi[i]**=** PsiParent[i]

NextGenPoolFitness[i] **=** FeasParentPoplnFitness[i]

**else**:

NextGenPool[i,:]**=** FeasOffsprngPopln [i,:]

NextGenPoolPsi[i]**=**PsiOffsprng[i]

NextGenPoolFitness[i] **=** FeasOffsprngPoplnFitness[i]

**return** [NextGenPool, NextGenPoolFitness, NextGenPoolPsi ]

### 8.8 Main program for DE HOF

The main Python program for DE HOF is shown below. A concise and clear Process Flow Chart of DE HOF constructing the optimal 130-30 portfolio can be found in Chapter 6 Metaheuristic 130-30 Portfolio Construction in [PAI, 2018].

In [12]:

"""

Main Program

130-30 portfolio Construction using Differential Evoution with Hall of Fame Metaheuristic strategy

--------------------------------------------------------------------------------------------------

Dataset: S&P BSE200 (April 03, 2009 - July 03, 2020)

Input Data File: S&PBSE200\_kPortfolio.csv

The asset labels of the k-portfolio followed by the mean returns of the

assets, variance-covariance matrix of returns of the assets

and betas of the assets in the order stated, are available in the

input csv file.

------------------------------------------------------------------

@author: Dr G A Vijayalakshmi Pai

"""

*#dependencies*

**import** numpy **as** np

**import** pandas **as** pd

**import** array **as** arr

**import** csv

**import** random

*# Read csv file to obtain asset labels, mean returns,*

*# variance-covariance matrix of asset returns and asset betas*

portfolioSize **=** 20

rows **=** 23

stockParamsFileName **=** 'S&PBSE200\_kPortfolio.csv'

df **=** pd**.**read\_csv(stockParamsFileName, nrows**=** rows)

*# extract asset labels*

assetLabels **=** df**.**columns**.**tolist()[0:portfolioSize]

print(assetLabels)

*# extract mean returns, variance-covariance matrix of returns and asset betas*

stockParamsData **=** np**.**array(df**.**iloc[0:, 0:])

meanData **=** np**.**array(stockParamsData[0,:])

covData **=** np**.**array(stockParamsData[1:portfolioSize**+**1, :])

betaAssets **=** np**.**array(stockParamsData[portfolioSize**+**1,:])

*# general bounds*

bounds **=** np**.**vstack( (np**.**repeat(**-**0.3, portfolioSize), np**.**repeat(1.3, portfolioSize)))

*# set bounds for the long and short positions in the portfolio as (0, 1.3) and (-0.3, 0) respectively*

longPosBounds **=** np**.**vstack( (np**.**repeat(0, portfolioSize), np**.**repeat(1.3, portfolioSize)))

shortPosBounds **=** np**.**vstack( (np**.**repeat(**-**0.3, portfolioSize), np**.**repeat(0, portfolioSize)))

*# Risk free rate*

annRiskFree **=** 6.5**/**100

riskfree **=** (np**.**power((1**+**annRiskFree),(1.0**/**360)) **-**1.0)**\***100

*# Differential Evolution strategy parameters*

poplnSize **=** 400

chromosomeLength **=** portfolioSize

totalGenerations **=** 600

beta **=** 0.5

probabilityRecombination **=** 0.87

*# Joines and Houck's (1994) dynamic penalty (dp) function strategy*

*# Set constants C, alpha, beta*

CConstant **=** 0.5

alphaConstant**=**2

betaConstant**=**2

*# initialize Hall of Fame which will finally hold the optimal weights*

HOFFitness **=** **-**999.0

HOFIndividual **=** np**.**zeros(shape**=**(portfolioSize), dtype **=** float)

i1 **=** 0

*# generation counter*

generationCount **=** 1

*# generate initial random population within the range [-0.3, 1.3]*

initialPoplnSeed **=** np**.**random**.**uniform(low **=** **-**0.3, high **=** 1.3, size **=**(poplnSize, chromosomeLength))

*# repair weights*

initialPoplnBound **=** WeightStdzn130\_30BoundsConstr(initialPoplnSeed, bounds)

initialPopln **=** WeightStdzn130\_30BudgetConstr(initialPoplnBound, longPosBounds, shortPosBounds)

*# compute constraint violation function*

[initialPoplnPsi, initialPoplnG1]**=** ConstrViolnFunction130\_30(initialPopln, betaAssets, CConstant, alphaConstant, betaConstant, generationCount)

*# compute fitness function*

initialPoplnFitness **=** ComputeFitness130\_30(initialPopln, meanData, covData, riskfree, initialPoplnPsi)

*# create parent population and compute fitness*

feasParentPopln **=** initialPopln

feasParentPoplnFitness **=** initialPoplnFitness

feasParentPoplnPsi **=** initialPoplnPsi

*#initialize HOF*

HOFGenerationArray **=** np**.**zeros(poplnSize)

HOFFitnessArray **=** np**.**zeros(poplnSize)

*# while loop for generation cycles begins*

**while** (generationCount **<=** totalGenerations):

print('generation:',generationCount)

dynamicBeta **=** 0.0

*# dynamic beta for each generation*

dynamicBeta **=** np**.**random**.**uniform(low **=**0.5, high **=**1.0)

*# obtain trial vector population*

trialVectorPopln **=** np**.**zeros(shape **=**(poplnSize, chromosomeLength), dtype **=** float)

trialVectorPopln **=** DEOperatorRand4BestDir5(feasParentPopln, feasParentPoplnFitness, dynamicBeta, poplnSize, chromosomeLength)

*# obtain offspring population*

offspringPopln **=** np**.**zeros(shape **=**(poplnSize, chromosomeLength), dtype **=** float)

offspringPopln **=** DEOperatorBinCrossover(feasParentPopln, trialVectorPopln, probabilityRecombination, chromosomeLength)

*# undertake weight repair strategy Phase 1*

mutatedPoplnBound **=** np**.**zeros(shape **=**(poplnSize, chromosomeLength), dtype **=** float)

mutatedPoplnBound **=** WeightStdzn130\_30BoundsConstr(offspringPopln, bounds)

*# undertake weight repair strategy Phase 2*

feasMutatedPopln **=** np**.**zeros(shape **=**(poplnSize, chromosomeLength), dtype **=** float)

feasMutatedPopln **=** WeightStdzn130\_30BudgetConstr(mutatedPoplnBound, longPosBounds, shortPosBounds)

*# compute constraint violation function*

feasMutatedPoplnPsi **=** np**.**zeros(shape **=**(poplnSize), dtype **=** float)

feasMutatedPopln\_G1 **=** np**.**zeros(shape **=**(poplnSize), dtype **=** float)

[feasMutatedPoplnPsi, feasMutatedPopln\_G1]**=** ConstrViolnFunction130\_30(feasMutatedPopln, betaAssets, CConstant, alphaConstant, betaConstant, generationCount)

*# compute fitness function values*

feasMutatedPoplnFitness **=** np**.**zeros(shape **=**(poplnSize, chromosomeLength), dtype **=** float)

feasMutatedPoplnFitness **=** ComputeFitness130\_30(feasMutatedPopln, meanData, covData, riskfree, feasMutatedPoplnPsi)

*# set the population for the next generation*

nextGenPool **=** np**.**zeros(shape **=**(poplnSize, chromosomeLength), dtype **=** float)

nextGenPoolFitness **=** np**.**zeros(shape **=**(poplnSize), dtype **=** float)

psiFun**=** np**.**zeros(shape **=**(poplnSize), dtype **=** float)

[nextGenPool, nextGenPoolFitness, psiFun ] **=** DEOperatorDetermSelection(feasParentPopln, feasParentPoplnFitness, feasParentPoplnPsi, feasMutatedPopln, feasMutatedPoplnFitness, feasMutatedPoplnPsi, poplnSize)

*# induct best individual into Hall of Fame*

**for** i **in** range(poplnSize):

**if** (psiFun[i] **==** 0):

**if** (nextGenPoolFitness[i] **>** HOFFitness):

HOFFitness **=** nextGenPoolFitness[i]

HOFIndividual **=** nextGenPool[i,:]

HOFGenerationArray[i1] **=** generationCount

HOFFitnessArray[i1] **=** HOFFitness

print('HOF updated Generation', generationCount)

print('HOF fitness', HOFFitness)

i1**=**i1**+**1

*# increment generation counter*

generationCount **=** generationCount **+** 1

feasParentPopln **=** np**.**zeros(shape **=**(poplnSize, chromosomeLength), dtype **=** float)

feasParentPoplnFitness **=** np**.**zeros(shape **=**(poplnSize), dtype **=** float)

*# set the parent population for the next generation*

feasParentPopln **=** nextGenPool

feasParentPoplnFitness **=** nextGenPoolFitness

*# while loop for DE HOF generations ends*

*# obtain optimal weights from the Hall of Fame*

xStar **=** HOFIndividual

*# compute optimal portfolio annualized risk and return*

portfolioReturn **=** 261 **\*** np**.**sum(np**.**multiply(meanData, xStar))

risk **=** np**.**sqrt(261 **\*** np**.**matmul( np**.**matmul(xStar, covData), xStar**.**T))

print("Integrated 130-30 Portfolio Optimization results:")

print("Annualized risk ( %) ", risk, " Expected Portfolio Annualized return ( %)", portfolioReturn)

sharpeRatio **=** (portfolioReturn**-**annRiskFree**\***100.0)**/** risk

print("Sharpe Ratio", sharpeRatio)

longIndex **=** np**.**where(xStar **>** 0)

shortIndex **=** np**.**where(xStar **<** 0)

print("Long positions of the optimal 130-30 portfolio: ", np**.**array(assetLabels)[longIndex])

print("Short positions of the optimal 130-30 portfolio: ", np**.**array(assetLabels)[shortIndex])

print("Optimal Weights: long positions: ", xStar[longIndex])

print("Optimal Weights: short positions: ", xStar[shortIndex])

print("Sum of long position weights: ", np**.**sum(xStar[longIndex]))

print("Sum of short position weights: ", np**.**sum(xStar[shortIndex]))

print("Sum of weights: " , np**.**sum(xStar))

portfoliobeta **=** np**.**sum(np**.**multiply(xStar, betaAssets))

print("Portfolio Beta: ", portfoliobeta)

print('Successful execution!')

["'AJANTPHARM'", " 'AMARAJABAT'", " 'ASHOKLEY'", " 'BHARATFORG'", " 'CIPLA'", " 'GAIL'", " 'HDFCBANK'", " 'HINDPETRO'", " 'IDEA'", "'IGL'", " 'IOC'", " 'ITC'", " 'JSWSTEEL'", " 'MOTILALOFS'", " 'PETRONET'", " 'SBIN'", " 'SHREECEM'", " 'TATASTEEL'", " 'TCS'", "'WIPRO'"]

('generation:', 1)

('HOF updated Generation', 1)

('HOF fitness', 0.05152434461565194)

('HOF updated Generation', 1)

('HOF fitness', 0.06703461897099006)

('generation:', 2)

('generation:', 3)

('HOF updated Generation', 3)

('HOF fitness', 0.08646358575372767)

('generation:', 4)

('generation:', 5)

('generation:', 6)

('generation:', 7)

('generation:', 8)

('generation:', 9)

('generation:', 10)

('generation:', 11)

('generation:', 12)

('generation:', 13)

('generation:', 14)

('generation:', 15)

('generation:', 16)

('generation:', 17)

('generation:', 18)

('generation:', 19)

('generation:', 20)

('generation:', 21)

('generation:', 22)

('generation:', 23)

('generation:', 24)

('generation:', 25)

('generation:', 26)

('generation:', 27)

('generation:', 28)

('generation:', 29)

('generation:', 30)

('generation:', 31)

('generation:', 32)

('generation:', 33)

('generation:', 34)

('generation:', 35)

('generation:', 36)

('generation:', 37)

('generation:', 38)

('HOF updated Generation', 38)

('HOF fitness', 0.08676048309116716)

('generation:', 39)

('generation:', 40)

('generation:', 41)

('generation:', 42)

('generation:', 43)

('generation:', 44)

('generation:', 45)

('generation:', 46)

('generation:', 47)

('generation:', 48)

('generation:', 49)

('generation:', 50)

('generation:', 51)

('generation:', 52)

('generation:', 53)

('generation:', 54)

('generation:', 55)

('generation:', 56)

('generation:', 57)

('generation:', 58)

('generation:', 59)

('generation:', 60)

('generation:', 61)

('generation:', 62)

('generation:', 63)

('generation:', 64)

('generation:', 65)

('generation:', 66)

('generation:', 67)

('generation:', 68)

('generation:', 69)

('generation:', 70)

('generation:', 71)

('generation:', 72)

('generation:', 73)

('generation:', 74)

('generation:', 75)

('generation:', 76)

('generation:', 77)

('generation:', 78)

('generation:', 79)

('generation:', 80)

('generation:', 81)

('generation:', 82)

('generation:', 83)

('generation:', 84)

('generation:', 85)

('generation:', 86)

('generation:', 87)

('generation:', 88)

('generation:', 89)

('generation:', 90)

('generation:', 91)

('generation:', 92)

('generation:', 93)

('generation:', 94)

('generation:', 95)

('generation:', 96)

('generation:', 97)

('generation:', 98)

('generation:', 99)

('generation:', 100)

('generation:', 101)

('HOF updated Generation', 101)

('HOF fitness', 0.08738288876439008)

('generation:', 102)

('generation:', 103)

('generation:', 104)

('generation:', 105)

('generation:', 106)

('generation:', 107)

('generation:', 108)

('generation:', 109)

('generation:', 110)

('generation:', 111)

('generation:', 112)

('generation:', 113)

('generation:', 114)

('generation:', 115)

('HOF updated Generation', 115)

('HOF fitness', 0.08820773443846115)

('generation:', 116)

('generation:', 117)

('generation:', 118)

('generation:', 119)

('generation:', 120)

('generation:', 121)

('generation:', 122)

('generation:', 123)

('generation:', 124)

('generation:', 125)

('generation:', 126)

('generation:', 127)

('generation:', 128)

('generation:', 129)

('generation:', 130)

('generation:', 131)

('generation:', 132)

('generation:', 133)

('generation:', 134)

('generation:', 135)

('generation:', 136)

('generation:', 137)

('generation:', 138)

('generation:', 139)

('generation:', 140)

('generation:', 141)

('generation:', 142)

('generation:', 143)

('generation:', 144)

('generation:', 145)

('generation:', 146)

('generation:', 147)

('generation:', 148)

('generation:', 149)

('generation:', 150)

('generation:', 151)

('generation:', 152)

('generation:', 153)

('generation:', 154)

('generation:', 155)

('generation:', 156)

('generation:', 157)

('generation:', 158)

('generation:', 159)

('generation:', 160)

('generation:', 161)

('generation:', 162)

('generation:', 163)

('generation:', 164)

('generation:', 165)

('generation:', 166)

('generation:', 167)

('generation:', 168)

('generation:', 169)

('generation:', 170)

('generation:', 171)

('generation:', 172)

('generation:', 173)

('generation:', 174)

('generation:', 175)

('generation:', 176)

('generation:', 177)

('generation:', 178)

('generation:', 179)

('generation:', 180)

('generation:', 181)

('generation:', 182)

('generation:', 183)

('generation:', 184)

('generation:', 185)

('generation:', 186)

('generation:', 187)

('generation:', 188)

('generation:', 189)

('HOF updated Generation', 189)

('HOF fitness', 0.09196478614276965)

('generation:', 190)

('generation:', 191)

('generation:', 192)

('generation:', 193)

('generation:', 194)

('generation:', 195)

('generation:', 196)

('generation:', 197)

('generation:', 198)

('generation:', 199)

('generation:', 200)

('generation:', 201)

('generation:', 202)

('generation:', 203)

('generation:', 204)

('generation:', 205)

('generation:', 206)

('generation:', 207)

('generation:', 208)

('generation:', 209)

('generation:', 210)

('generation:', 211)

('generation:', 212)

('generation:', 213)

('generation:', 214)

('generation:', 215)

('generation:', 216)

('generation:', 217)

('generation:', 218)

('generation:', 219)

('generation:', 220)

('generation:', 221)

('generation:', 222)

('generation:', 223)

('generation:', 224)

('generation:', 225)

('generation:', 226)

('generation:', 227)

('generation:', 228)

('generation:', 229)

('generation:', 230)

('generation:', 231)

('generation:', 232)

('generation:', 233)

('generation:', 234)

('generation:', 235)

('generation:', 236)

('generation:', 237)

('generation:', 238)

('generation:', 239)

('generation:', 240)

('generation:', 241)

('generation:', 242)

('generation:', 243)

('generation:', 244)

('generation:', 245)

('generation:', 246)

('generation:', 247)

('generation:', 248)

('generation:', 249)

('generation:', 250)

('generation:', 251)

('generation:', 252)

('generation:', 253)

('generation:', 254)

('generation:', 255)

('generation:', 256)

('generation:', 257)

('generation:', 258)

('generation:', 259)

('generation:', 260)

('generation:', 261)

('generation:', 262)

('generation:', 263)

('generation:', 264)

('generation:', 265)

('generation:', 266)

('generation:', 267)

('generation:', 268)

('generation:', 269)

('generation:', 270)

('generation:', 271)

('generation:', 272)

('generation:', 273)

('generation:', 274)

('generation:', 275)

('generation:', 276)

('generation:', 277)

('generation:', 278)

('generation:', 279)

('generation:', 280)

('generation:', 281)

('generation:', 282)

('generation:', 283)

('generation:', 284)

('generation:', 285)

('generation:', 286)

('generation:', 287)

('generation:', 288)

('generation:', 289)

('generation:', 290)

('generation:', 291)

('generation:', 292)

('generation:', 293)

('generation:', 294)

('generation:', 295)

('generation:', 296)

('generation:', 297)

('generation:', 298)

('generation:', 299)

('generation:', 300)

('generation:', 301)

('generation:', 302)

('generation:', 303)

('generation:', 304)

('generation:', 305)

('generation:', 306)

('generation:', 307)

('generation:', 308)

('generation:', 309)

('generation:', 310)

('generation:', 311)

('generation:', 312)

('generation:', 313)

('generation:', 314)

('generation:', 315)

('generation:', 316)

('generation:', 317)

('generation:', 318)

('generation:', 319)

('generation:', 320)

('generation:', 321)

('generation:', 322)

('generation:', 323)

('generation:', 324)

('generation:', 325)

('generation:', 326)

('generation:', 327)

('generation:', 328)

('generation:', 329)

('generation:', 330)

('generation:', 331)

('generation:', 332)

('generation:', 333)

('generation:', 334)

('generation:', 335)

('generation:', 336)

('generation:', 337)

('generation:', 338)

('generation:', 339)

('generation:', 340)

('generation:', 341)

('generation:', 342)

('generation:', 343)

('generation:', 344)

('generation:', 345)

('generation:', 346)

('generation:', 347)

('generation:', 348)

('generation:', 349)

('generation:', 350)

('generation:', 351)

('generation:', 352)

('generation:', 353)

('generation:', 354)

('generation:', 355)

('generation:', 356)

('generation:', 357)

('generation:', 358)

('generation:', 359)

('generation:', 360)

('generation:', 361)

('generation:', 362)

('generation:', 363)

('generation:', 364)

('generation:', 365)

('generation:', 366)

('generation:', 367)

('generation:', 368)

('generation:', 369)

('generation:', 370)

('generation:', 371)

('generation:', 372)

('generation:', 373)

('generation:', 374)

('generation:', 375)

('generation:', 376)

('generation:', 377)

('generation:', 378)

('generation:', 379)

('generation:', 380)

('generation:', 381)

('generation:', 382)

('generation:', 383)

('generation:', 384)

('generation:', 385)

('generation:', 386)

('generation:', 387)

('generation:', 388)

('generation:', 389)

('generation:', 390)

('generation:', 391)

('generation:', 392)

('generation:', 393)

('generation:', 394)

('generation:', 395)

('generation:', 396)

('generation:', 397)

('generation:', 398)

('generation:', 399)

('generation:', 400)

('generation:', 401)

('generation:', 402)

('generation:', 403)

('generation:', 404)

('generation:', 405)

('generation:', 406)

('generation:', 407)

('generation:', 408)

('generation:', 409)

('generation:', 410)

('generation:', 411)

('generation:', 412)

('generation:', 413)

('generation:', 414)

('generation:', 415)

('generation:', 416)

('generation:', 417)

('generation:', 418)

('generation:', 419)

('generation:', 420)

('generation:', 421)

('generation:', 422)

('generation:', 423)

('generation:', 424)

('generation:', 425)

('generation:', 426)

('generation:', 427)

('generation:', 428)

('generation:', 429)

('generation:', 430)

('generation:', 431)

('generation:', 432)

('generation:', 433)

('generation:', 434)

('generation:', 435)

('generation:', 436)

('generation:', 437)

('generation:', 438)

('generation:', 439)

('generation:', 440)

('generation:', 441)

('generation:', 442)

('generation:', 443)

('generation:', 444)

('generation:', 445)

('generation:', 446)

('generation:', 447)

('generation:', 448)

('generation:', 449)

('generation:', 450)

('generation:', 451)

('generation:', 452)

('generation:', 453)

('generation:', 454)

('generation:', 455)

('generation:', 456)

('generation:', 457)

('generation:', 458)

('generation:', 459)

('generation:', 460)

('generation:', 461)

('generation:', 462)

('generation:', 463)

('generation:', 464)

('generation:', 465)

('generation:', 466)

('generation:', 467)

('generation:', 468)

('generation:', 469)

('generation:', 470)

('generation:', 471)

('generation:', 472)

('generation:', 473)

('generation:', 474)

('generation:', 475)

('generation:', 476)

('generation:', 477)

('generation:', 478)

('generation:', 479)

('generation:', 480)

('generation:', 481)

('generation:', 482)

('generation:', 483)

('generation:', 484)

('generation:', 485)

('generation:', 486)

('generation:', 487)

('generation:', 488)

('generation:', 489)

('generation:', 490)

('generation:', 491)

('generation:', 492)

('generation:', 493)

('generation:', 494)

('generation:', 495)

('generation:', 496)

('generation:', 497)

('generation:', 498)

('generation:', 499)

('generation:', 500)

('generation:', 501)

('generation:', 502)

('generation:', 503)

('generation:', 504)

('generation:', 505)

('generation:', 506)

('generation:', 507)

('generation:', 508)

('generation:', 509)

('generation:', 510)

('generation:', 511)

('generation:', 512)

('generation:', 513)

('generation:', 514)

('generation:', 515)

('generation:', 516)

('generation:', 517)

('generation:', 518)

('generation:', 519)

('generation:', 520)

('generation:', 521)

('generation:', 522)

('generation:', 523)

('generation:', 524)

('generation:', 525)

('generation:', 526)

('generation:', 527)

('generation:', 528)

('generation:', 529)

('generation:', 530)

('generation:', 531)

('generation:', 532)

('generation:', 533)

('generation:', 534)

('generation:', 535)

('generation:', 536)

('generation:', 537)

('generation:', 538)

('generation:', 539)

('generation:', 540)

('generation:', 541)

('generation:', 542)

('generation:', 543)

('generation:', 544)

('generation:', 545)

('generation:', 546)

('generation:', 547)

('generation:', 548)

('generation:', 549)

('generation:', 550)

('generation:', 551)

('generation:', 552)

('generation:', 553)

('generation:', 554)

('generation:', 555)

('generation:', 556)

('generation:', 557)

('generation:', 558)

('generation:', 559)

('generation:', 560)

('generation:', 561)

('generation:', 562)

('generation:', 563)

('generation:', 564)

('generation:', 565)

('generation:', 566)

('generation:', 567)

('generation:', 568)

('generation:', 569)

('generation:', 570)

('generation:', 571)

('generation:', 572)

('generation:', 573)

('generation:', 574)

('generation:', 575)

('generation:', 576)

('generation:', 577)

('generation:', 578)

('generation:', 579)

('generation:', 580)

('generation:', 581)

('generation:', 582)

('generation:', 583)

('generation:', 584)

('generation:', 585)

('generation:', 586)

('generation:', 587)

('generation:', 588)

('HOF updated Generation', 588)

('HOF fitness', 0.09199168964039035)

('generation:', 589)

('generation:', 590)

('generation:', 591)

('generation:', 592)

('generation:', 593)

('generation:', 594)

('generation:', 595)

('generation:', 596)

('generation:', 597)

('generation:', 598)

('generation:', 599)

('generation:', 600)

Integrated 130-30 Portfolio Optimization results:

('Annualized risk ( %) ', 26.44353441923466, ' Expected Portfolio Annualized return ( %)', 43.86569234073356)

('Sharpe Ratio', 1.4130369922696218)

('Long positions of the optimal 130-30 portfolio: ', array(["'AJANTPHARM'", " 'AMARAJABAT'", " 'ASHOKLEY'", "'IGL'",

" 'MOTILALOFS'", " 'SHREECEM'"], dtype='|S13'))

('Short positions of the optimal 130-30 portfolio: ', array([" 'CIPLA'", " 'ITC'", " 'TATASTEEL'"], dtype='|S13'))

('Optimal Weights: long positions: ', array([0.16035239, 0.30002306, 0.20909457, 0.29617098, 0.17622758,

0.15813142]))

('Optimal Weights: short positions: ', array([-0.15296674, -0.00730503, -0.13972824]))

('Sum of long position weights: ', 1.3)

('Sum of short position weights: ', -0.29999999999999943)

('Sum of weights: ', 1.0000000000000007)

('Portfolio Beta: ', 1.0003742124207258)

Successful execution!

## 9. Optimal 130-30 Portfolio - Analysis of Results

Typical of metaheuristic strategies, DE HOF yields multiple optimal solutions during its various runs. The interpretation of the results obtained by DE HOF during one of its runs is as follows:

Optimal 130-30 Portfolio

Maximal Sharpe Ratio: 1.54  
Annualized Risk(%): 26.32  
Expected Portfolio Annualized Return(%): 46.90

Portfolio Beta: 1.00

Optimal Weights delivered by DE HOF for the k-portfolio (k = 20):

(0.225, 0.094, 0.208, 0, -0.063, -0.237, 0, 0.092, 0, 0.162, 0, 0.055, 0.038, 0, 0.0796, 0, 0.346, 0, 0, 0)

Long Short Portfolio Composition:

Fig. 1.2 illustrates the composition of the optimal 130-30 long short portofilio obtained by DE HOF in one of its runs. The description of the tickers have been listed below.

Long Positions: (Ajanta Pharma Ltd. ["'AJANTPHARM'"] 22.5%, Amara Raja Batteries Ltd.["'AMARAJABAT'"] 9.4%, Ashok Leyland Ltd.["'ASHOKLEY'"] 20.8%, Hindustan Petroleum Corporation Ltd. [ "'HINDPETRO'"] 9.2%, Indraprastha Gas Ltd. ["'IGL'"] 16.2%, ITC Ltd. ["'ITC'"] 5.5%, JSW Steel Ltd.[ "'JSWSTEEL'"] 3.8%, Petronet LNG Ltd. ["'PETRONET'"] 7.96%, Shree Cement Ltd. ["'SHREECEM'"] 34.6%.

Short Positions: (Cipla Ltd. ["'CIPLA'"] 6.3%, GAIL (India) Ltd. ["'GAIL'"] 23.7%)

Leveraged Budget on long positions: 130% Budget on short positions: 30% Sum of optimal weights: 100%

Chart

Description automatically generated

#### (a) Long Positions

Chart, bar chart

Description automatically generated

#### (b) Short Positions

#### Fig.1.2 Optimal 130-30 Long Short Portfolio Composition over S&P BSE200 index

### 9.1 Multiple solutions - Making a choice!

Table 1.1 shows the summarized characteristics of optimal 130-30 portfolios delivered by DE HOF during various runs. At this juncture it needs to be emphasized that due to the stochastic behavior of exploring solutions over large search spaces by generating a random population of individuals which evolve over the generations into optimal solutions, metaheuristic strategies are endowed with inherent capabilities to deliver multiple solutions that are either optimal or acceptable or near-optimal. Thus metaheuristic strategies generate multiple solutions to the problem in hand for various runs. This characteristic of metaheuristic algorithms needs to be viewed meritoriously and exploited to one's own advantage.

#### Table 1.1 Summarized characteristics of optimal 130-30 portfolios of S&PBSE200 index obtained by DE HOF

Table

Description automatically generated

Thus in the case of 130-30 portfolio construction, though the various runs of DE HOF yield optimal portfolios with similar characteristics of risk, return and Sharpe Ratio, the choice of long short positions and the allocation of weights to the assets in the optimal portfolios concerned, are different and diverse for each run. With the risk, return and Sharpe Ratio of the optimal portfolio guaranteed, it is now left to the discerning investor to make a choice of the portfolio with a composition of assets and allocation of weights that best suits his or her investment sentiments.

Fig. 1.3 illustrates another optimal 130-30 portfolio constructed by DE HOF during one of its runs. The characteristics of the portfolio are as listed below:

Another Optimal 130-30 Portfolio

Maximal Sharpe Ratio: 1.51  
Annualized Risk(%): 27.61  
Expected Portfolio Annualized Return(%): 48.05

Portfolio Beta: 1.00

Optimal Weights delivered by DE HOF for the k-portfolio (k = 20):

(0.378, 0.040, 0.086, 0.124, -0.164, -0.136, 0, 0.074, 0, 0.067, 0, 0, 0, 0.142, 0, 0, 0.390, 0, 0, 0)

Long Short Portfolio Composition:

Long Positions: Ajanta Pharma Ltd. ["'AJANTPHARM'"] 37.8%, Amara Raja Batteries Ltd.["'AMARAJABAT'"] 4%, Ashok Leyland Ltd.["'ASHOKLEY'"] 8.6%, Bharat Forge Ltd. ["'BHARATFORG'"] 12.4%, Hindustan Petroleum Corporation Ltd. [ "'HINDPETRO'"] 7.4%, Indraprastha Gas Ltd. ["'IGL'"] 6.7%, Motilal Oswal Financial Services Ltd. ["'MOTILALOFS'"] 14.2%, Shree Cement Ltd. ["'SHREECEM'"] 39%.

Short Positions: (Cipla Ltd. ["'CIPLA'"] 16.4%, GAIL (India) Ltd. ["'GAIL'"] 13.6%)

Leveraged Budget on long positions: 130% Budget on short positions: 30% Sum of optimal weights: 100%

Chart

Description automatically generated

#### (a) Long Positions

Chart, bar chart

Description automatically generated

#### (b) Short Positions

#### Fig.1.3 Another Optimal 130-30 Long Short Portfolio Composition over S&P BSE 200 index

### 9.2 Convergence characteristics of DE HOF

The convergence behaviour of DE HOF, where the fitness function values of individuals inducted into the Hall of Fame (that records the best solution obtained this far) during the course of the generation cycles, for various runs was observed. Fig. 1.4 illustrates the graph and it can be seen that the fitness function values stabilized around 300 generations and thereafter there was no siginificant rise in the fitness function values of individuals inducted into the Hall of Fame.

Chart, line chart, scatter chart

Description automatically generated

#### Fig. 1.4 Convergence of DE HOF

## 10. Conclusion

130-30 portfolios have served as good instruments to ensure investment exposure and market protection at the same time, especially during times when market behaviour has been erratic - the downturns during Covid-19 crisis being a good example in recent times.

An integrated optimization of long-short portfolios can help construct optimal 130-30 portfolios with leveraged returns and maximal Sharpe ratios and a metaheuristic strategy such as Differential Evolution with Hall of Fame can serve to construct such a portfolio.

The metaheuristic strategy is inherently endowed with the potential to deliver multiple near-optimal 130-30 portfolios, providing investors the luxury of making choices that follow their investment sentiments.

## Companion Reading

[1] Chapter 6 Metaheuristic 130-30 Portfolio Construction [PAI 2018]

[2] Chapter 2 A Brief Primer on Metaheuristics [PAI 2018]

[3] MATLAB Demonstration of 130-30 Portfolio Construction using DE HOF, in Mathworks Central File Exchange <https://in.mathworks.com/matlabcentral/fileexchange/64507-metaheuristic-portfolio-optimization-models>

[4] Sharpe Ratio based Portfolio Optimization <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson6_SharpeRatioOptimization/Lesson6_MainContent.ipynb>

[5] Heuristic Portfolio Selection <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson3_HeuristicPortfolioSelection/Lesson3_MainContent.ipynb>

[6] Fundamentals of Risk and Return of a Portfolio <https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson1_FundaRiskReturnPortfolio/Lesson1_MainContent.ipynb>

## References

[FEO 2006] Feoktistov V, "Differential Evolution in search of solutions", Springer, 2006.

[JAC 1999] Jacobs B I, Levy K N and D Starer, "Long-Short Portfolio Management: An Integrated Approach", The Journal of Portfolio Management, vol. 25, no. 2, pp. 23-32, 1999.

[JOI 1994] Joines J A and C R Houck, "On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs", Proceedings of the First IEEE Conference on Evolutionary Computation, pp.579-584, 1994.

[PAI, 2018] Vijayalakshmi Pai G. A., "Metaheuristics for Portfolio Optimization- An Introduction using MATLAB", Wiley-ISTE, 2018. <https://www.mathworks.com/academia/books/metaheuristics-for-portfolio-optimization-pai.html>

***Global Optimization of Portfolios - a rendezvous with VitaOptimum Plus* !**

A picture containing diagram

Description automatically generated

**1. Prologue**

**Portfolio Optimization** is a discipline that deals with the construction of portfolios from tradable assets such as stocks, bonds and securities, with the dual objectives of ***maximising the expected portfolio return*** and ***minimizing portfolio risk***. The problem models can turn complex when constraints, both linear/non-linear and representing investor preferences, market norms, investment strategies, why even religious beliefs, are incorporated into them.

Some of these optimization problem models have been satisfactorily solved, if not efficiently, using traditional methods. However, it needs to be realized that traditional methods despite their sophistication, suffer from pitfalls such as dependence on a suitable *starting point* or *initial point* in the search space from where the search for solution begins, or the need to enumerate all sets of candidate solution sets to determine the best or optimal solution, which can be time consuming, or the quality or effectiveness of the optimal solutions to the problems concerned.

Again, there are problem models which are difficult to be solved by traditional methods. Fortunately, heuristic methods inspired by nature have helped solve these complex problems to obtain acceptable or near-optimal solutions, if not global optimal solutions.

**2. Goal**

In this post, we discuss the solution of a fundamental portfolio optimization model, termed **Sharpe Ratio based portfolio optimization**, which is a non-linear single objective constrained optimization problem.

We solve the model using a traditional method viz., **Sequential Least Squares Quadratic Programming (SLSQP)** using a solver available in **SciPy**, a Python library.

We also solve the same problem using **VitaOptimum Plus**, a Python based non-traditional solver that employs cutting edge technologies of Artificial Intelligence, Evolutionary Computation, Swarm Intelligence and Advanced Statistics, to arrive at the global optimum.

The quality of the optimal portfolios obtained by the two solvers are compared and analyzed.

**3. Sharpe Ratio**

**Sharpe Ratio**, developed by Nobel Laureate William F Sharpe [SHA 66], is a measure of calculating risk adjusted return. It serves to help investors know about the returns on their investments relative to the risks they hold. The Sharpe Ratio is defined as



**..........(1)**

where $r\_x$ is the average rate of return on the investment $x$, $R\_f$ is the best available **risk free rate of return** and $\sigma$ is the standard deviation of $r\_x$, which denotes the risk on the investment.

Higher the Sharpe Ratio, more is the excess returns over that of holding a risk free investment, relative to the increased volatility that the investment is exposed to. A Sharpe Ratio of 0, needless to say, only denotes the investment to be risk-free or one that does not yield any excess return. In practice, while a Sharpe ratio of 1 marks the investment to be acceptable or good for investors, a value less than 1 grades the investment as sub-optimal, and values greater than 1 and moving towards 2 or 3, grade the investments as highly superior.

**4. Maximizing Sharpe Ratio**

Having understood the significance of the Sharpe Ratio, let us suppose an investor wishes to make an investment in assets in such a way that the Sharpe Ratio of the portfolio would be the best possible or the maximum that can be ensured for that investment.

Let P be a portfolio comprising n assets with weights represented by vector W and asset returns represented by vector $\mu$. The portfolio return $r$ is determined by a weighted summation of its individual asset returns and portfolio risk is as given by the denominator of equation (2) shown below. (See **Lesson 1 Fundamentals of Risk and Return of a Portfolio** to know about risk and return of a portfolio).

To keep the discussion simple for now, let us suppose that the investor decides to enforce only **basic constraints** on the portfolio. (See Sec. 5.2 of **Lesson 5 Mean Variance Optimization Model** to know about basic constraints).

The mathematical model for the Sharpe Ratio based Portfolio Optimization is given by,

A picture containing shape

Description automatically generated

**..........(2)**

The numerator of the objective function denotes the excess returns of the investment over that of a risk free asset $R\_f$ and the denominator the risk of the investment. The objective is to maximize the Sharpe Ratio. The basic constraints indicate that the investor wishes to have a **fully invested** portfolio, in other words, invest 100% of the capital in the portfolio.

**5. Solving the Sharpe Ratio Optimization Model using *scipy***

To solve the Sharpe Ratio maximization model represented by (2), we make use of the **minimize** library function from **scipy.optimize** package of Python, adopting the *Sequential Least Squares Quadratic Programming* (SLSQP)method. However, since the original objective function insists on maximization as opposed to minimization demanded by the **minimize** solver, the **principal of duality** borrowed from Optimization Theory is employed to undertake the transformation. According to the principle,



**..........(3)**

The Python code for the function **MaximizeSharpeRatioOptimization**, which defines the objective function and the basic constraints represented by equation (2), is shown here.

In [27]:

*#function to undertake Sharpe Ratio maximization subject to*

*#basic constraints of the portfolio*

*#Solver uses optimize library function from scipy that employs*

*#SLSQP (Sequential Leas Squares Quadratic Programming) method to solve the*

*#non-linear single objective constrained optimization problem*

*#dependencies*

**import** numpy **as** np

**from** scipy **import** optimize

**def** MaximizeSharpeRatioOptmzn(MeanReturns, CovarReturns, RiskFreeRate, PortfolioSize):

*# define maximization of Sharpe Ratio using principle of duality*

**def** f(x, MeanReturns, CovarReturns, RiskFreeRate, PortfolioSize):

funcDenomr **=** np**.**sqrt(np**.**matmul(np**.**matmul(x, CovarReturns), x**.**T) )

funcNumer **=** np**.**matmul(np**.**array(MeanReturns),x**.**T)**-**RiskFreeRate

func **=** **-**(funcNumer **/** funcDenomr)

**return** func

*#define equality constraint representing fully invested portfolio*

**def** constraintEq(x):

A**=**np**.**ones(x**.**shape)

b**=**1

constraintVal **=** np**.**matmul(A,x**.**T)**-**b

**return** constraintVal

*#define bounds and other parameters*

xinit**=**np**.**repeat(0.33, PortfolioSize)

cons **=** ({'type': 'eq', 'fun':constraintEq})

lb **=** 0.0

ub **=** 1.0

bnds **=** tuple([(lb,ub) **for** x **in** xinit])

*#invoke minimize solver*

opt **=** optimize**.**minimize (f, x0 **=** xinit, args **=** (MeanReturns, CovarReturns,\

RiskFreeRate, PortfolioSize), method **=** 'SLSQP', \

bounds **=** bnds, constraints **=** cons, tol **=** 1.e-16)

**return** opt

The Sharpe Ratio optimization requires the computation of risk and return of the portfolio, for which asset returns are required. Function **StockReturnsComputing** which computes the asset returns is illustrated below:

In [28]:

*# function computes asset returns*

**def** StockReturnsComputing(StockPrice, Rows, Columns):

StockReturn **=** np**.**zeros([Rows**-**1, Columns])

**for** j **in** range(Columns): *# j: Assets*

**for** i **in** range(Rows**-**1): *# i: Daily Prices*

StockReturn[i,j]**=**((StockPrice[i**+**1, j]**-**StockPrice[i,j])**/**StockPrice[i,j])**\*** 100

**return** StockReturn

**6. Case Study**

Let us suppose that an investor decides to invest in a portfolio comprising the following Dow stocks. While a typical portfolio is a prudent combination of assets comprising equities, bonds, currencies etc., as dictated by the investor's risk appetite, we choose to discuss an equity based portfolio only to keep the narrative simple.

The following are the Dow stocks that has interested the investor:

{Coca-Cola (KO), United Health (UNH), Walt Disney (DIS), IBM (IBM), Cisco (CSCO), JPMorgan Chase (JPM), Goldman Sachs (GS), Walgreens Boots Alliance (WBA), Apple (AAPL), Home Depot (HD), American Express (AXP), McDonald's (MCD), Merck (MRK), Boeing (BA), Caterpillar (CAT)}

The investor desires to obtain the optimal portfolio set that would yield the maximal Sharpe Ratio. The objective therefore, is to find the optimal weights that will ensure maximal Sharpe Ratio for the portfolio. In other words, the investor wishes to know what proportion of capital must be invested in each of the the stocks so that the portfolio yields maximal Sharpe ratio.

The following Python code reads the dataset DJIA\_Apr112014\_Apr112019\_kpf1.csv (Dow 30: April 11 2014 -April 11 2019) concerned, computes the stock returns using the Python function **StockReturnsComputing** and obtains the mean returns and the variance-covariance matrix of returns.

In [29]:

*#obtain mean and variance-covariance matrix of returns for the portfolio*

*#Dependencies*

**import** numpy **as** np

**import** pandas **as** pd

**if** \_\_name\_\_**==**'\_\_main\_\_':

*#input portfolio dataset comprising 15 stocks*

StockFileName **=** 'DJIA\_Apr112014\_Apr112019\_kpf1.csv'

Rows **=** 1259 *#excluding header*

Columns **=** 15 *#excluding date*

*#read stock prices*

df **=** pd**.**read\_csv(StockFileName, nrows**=** Rows)

*#extract asset labels*

assetLabels **=** df**.**columns[1:Columns**+**1]**.**tolist()

print('Asset labels of portfolio: \n', assetLabels)

*#read asset prices data*

StockData **=** df**.**iloc[0:, 1:]

*#compute asset returns*

arStockPrices **=** np**.**asarray(StockData)

[Rows, Cols]**=**arStockPrices**.**shape

arReturns **=** StockReturnsComputing(arStockPrices, Rows, Cols)

*#set precision for printing results*

np**.**set\_printoptions(precision**=**3, suppress **=** **True**)

*#compute mean returns and variance covariance matrix of returns*

meanReturns **=** np**.**mean(arReturns, axis **=** 0)

covReturns **=** np**.**cov(arReturns, rowvar**=False**)

print('\nMean Returns:\n', meanReturns)

print('\nVariance-Covariance Matrix of Returns:\n', covReturns)

Asset labels of portfolio:

['AAPL', 'AXP', 'BA', 'CAT', 'CSCO', 'DIS', 'GS', 'HD', 'IBM', 'JPM', 'KO', 'MCD', 'MRK', 'UNH', 'WBA']

Mean Returns:

[ 0.09 0.029 0.1 0.039 0.081 0.04 0.033 0.085 -0.016 0.06

0.019 0.057 0.036 0.095 -0.002]

Variance-Covariance Matrix of Returns:

[[2.375 0.672 0.962 1.042 0.999 0.68 0.954 0.726 0.709 0.825 0.306 0.458

0.534 0.774 0.697]

[0.672 1.648 0.8 0.95 0.7 0.569 1.065 0.658 0.663 1.001 0.307 0.35

0.556 0.718 0.667]

[0.962 0.8 2.288 1.31 0.89 0.716 1.066 0.747 0.777 0.977 0.381 0.472

0.578 0.745 0.679]

[1.042 0.95 1.31 2.733 1.041 0.688 1.321 0.796 0.885 1.169 0.358 0.455

0.616 0.72 0.681]

[0.999 0.7 0.89 1.041 1.789 0.713 0.927 0.724 0.817 0.909 0.362 0.477

0.647 0.656 0.707]

[0.68 0.569 0.716 0.688 0.713 1.35 0.773 0.586 0.574 0.717 0.302 0.368

0.466 0.557 0.631]

[0.954 1.065 1.066 1.321 0.927 0.773 2.114 0.795 0.803 1.554 0.303 0.467

0.705 0.82 0.819]

[0.726 0.658 0.747 0.796 0.724 0.586 0.795 1.39 0.619 0.753 0.343 0.472

0.487 0.659 0.689]

[0.709 0.663 0.777 0.885 0.817 0.574 0.803 0.619 1.632 0.767 0.372 0.391

0.576 0.564 0.534]

[0.825 1.001 0.977 1.169 0.909 0.717 1.554 0.753 0.767 1.702 0.324 0.483

0.675 0.761 0.717]

[0.306 0.307 0.381 0.358 0.362 0.302 0.303 0.343 0.372 0.324 0.806 0.36

0.384 0.31 0.355]

[0.458 0.35 0.472 0.455 0.477 0.368 0.467 0.472 0.391 0.483 0.36 1.086

0.402 0.43 0.433]

[0.534 0.556 0.578 0.616 0.647 0.466 0.705 0.487 0.576 0.675 0.384 0.402

1.504 0.615 0.64 ]

[0.774 0.718 0.745 0.72 0.656 0.557 0.82 0.659 0.564 0.761 0.31 0.43

0.615 1.722 0.78 ]

[0.697 0.667 0.679 0.681 0.707 0.631 0.819 0.689 0.534 0.717 0.355 0.433

0.64 0.78 2.554]]

The annual average risk free rate of return in USA during April 2019 was 3%. The daily risk free rate is computed as



**..........(4)**

The following Python code computes the maximal Sharpe Ratio for the portfolio.

In [30]:

*#obtain maximal Sharpe Ratio for the portfolio of Dow stocks*

*#set portfolio size*

portfolioSize **=** Columns

*#set risk free asset rate of return*

Rf**=**3 *# April 2019 average risk free rate of return in USA approx 3%*

annRiskFreeRate **=** Rf**/**100

*#compute daily risk free rate in percentage*

r0 **=** (np**.**power((1 **+** annRiskFreeRate), (1.0 **/** 360.0)) **-** 1.0) **\*** 100

print('\nRisk free rate (daily %): ', end**=**"")

print ("{0:.3f}"**.**format(r0))

*#initialization*

xOptimal **=**[]

minRiskPoint **=** []

expPortfolioReturnPoint **=**[]

maxSharpeRatio **=** 0

*#compute maximal Sharpe Ratio and optimal weights*

result **=** MaximizeSharpeRatioOptmzn(meanReturns, covReturns, r0, portfolioSize)

print(result)

xOptimal**.**append(result**.**x)

*#compute risk returns and max Sharpe Ratio of the optimal portfolio*

xOptimalArray **=** np**.**array(xOptimal)

Risk **=** np**.**matmul((np**.**matmul(xOptimalArray,covReturns)), np**.**transpose(xOptimalArray))

expReturn **=** np**.**matmul(np**.**array(meanReturns),xOptimalArray**.**T)

annRisk **=** np**.**sqrt(Risk**\***251)

annRet **=** 251**\***np**.**array(expReturn)

maxSharpeRatio **=** (annRet**-**Rf)**/**annRisk

*#set precision for printing results*

np**.**set\_printoptions(precision**=**3, suppress **=** **True**)

*#display results*

print('Maximal Sharpe Ratio: ', maxSharpeRatio, '\nAnnualized Risk (%): ', \

annRisk, '\nAnnualized Expected Portfolio Return(%): ', annRet)

print('\nOptimal weights (%):\n', xOptimalArray**.**T**\***100 )

Risk free rate (daily %): 0.008

fun: -0.08453042881818211

jac: array([-0.009, 0.033, -0.009, 0.042, -0.009, 0.015, 0.045, -0.009,

0.078, 0.011, 0.012, -0.009, 0.014, -0.009, 0.068])

message: 'Optimization terminated successfully.'

nfev: 570

nit: 32

njev: 31

status: 0

success: True

x: array([0.084, 0. , 0.175, 0. , 0.074, 0. , 0. , 0.253, 0. ,

0. , 0. , 0.123, 0. , 0.291, 0. ])

Maximal Sharpe Ratio: [[1.276]]

Annualized Risk (%): [[14.809]]

Annualized Expected Portfolio Return(%): [21.894]

Optimal weights (%):

[[ 8.406]

[ 0. ]

[17.527]

[ 0. ]

[ 7.396]

[ 0. ]

[ 0. ]

[25.282]

[ 0. ]

[ 0. ]

[ 0. ]

[12.264]

[ 0. ]

[29.125]

[ 0. ]]

The output shows that the maximal Sharpe Ratio attainable for the portfolio is 1.28 which is good, going by practical standards. The annual expected portfolio return is 21.89% against an annualized risk of 14.81%. To achieve this, the optimal capital allocations on the assets of the portfolio are as follows:

['AAPL': 8.406%] , ['BA': 17.527%], ['CSCO': 7.396%], ['HD': 25.282%], ['MCD': 12.264%], ['UNH': 29.125%]

No investments need be made in the rest of the assets of the portfolio since the optimal weights arrived at for these assets are 0.  
However, if the investor desires to invest in all the assets in the portfolio, with the weights distributed across all the assets in the portfolio, all that the investor needs to do is to redefine the bounds constraint of (2) as, $0\lt W\_i\lt 1$ and run the **scipy** solver over the optimization model.

**7. Solving the Sharpe Ratio Optimization Model using *VitaOptimum Plus***

VitaOptimum Plus [VIT, 2019] is a state-of-the-art global optimization solver encompassing the cutting edge technologies of Artificial Intelligence, Evolutionary Computation, Swarm Intelligence and Advanced Statistics, to efficiently tackle constrained linear and non-linear models. The Sharpe ratio based portfolio optimization model described in equation (2)and employing the **Ccs** (**C**ontinuous **c**onstrained **s**olver) Python class from **vitaoptimum.voplus.ccs** package is described below.

In [31]:

**import** ctypes

**import** numpy **as** np

**import** pandas **as** pd

**from** vitaoptimum.voplus.ccs **import** Ccs

*#global maximum Sharpe Ratio optimization solver wrapper function*

**def** GlobalMaxSharpeRatioOptmzn(MeanReturns, CovarReturns, RiskFreeRate, PortfolioSize):

*# define maximization of Sharpe Ratio using principle of duality*

**def** fobj(x, g, h):

h[0]**=**sum(x)**-**1.0

funcDenomr **=** np**.**sqrt(np**.**matmul(np**.**matmul(x, CovarReturns), x**.**T) )

funcNumer **=** np**.**matmul(np**.**array(MeanReturns),x**.**T)**-**RiskFreeRate

func **=** **-**(funcNumer **/** funcDenomr)

**return** func

*# dimension of the problem and constraints*

dim **=** PortfolioSize *# problem dimension*

*#configure quality measures (array qmeasures) to control the internal metrics of the solver*

qmeasures **=** np**.**array([1.0e-16, 1.0e-15, 1.0e-15, 0.0], dtype **=** ctypes**.**c\_double)

nh **=** 1 *# equality constraint dimension [ h(x) = 0 ]*

ng **=** 0 *# non equality constraint dimension [ g(x) <= 0 ]*

*# set boundary constraints to [0, 1]*

low **=** np**.**zeros(dim, dtype**=**ctypes**.**c\_double) **+** [0.0]

high **=** np**.**zeros(dim, dtype**=**ctypes**.**c\_double) **+** [1.0]

solver **=** Ccs(fobj**=**fobj, *# objective function with constraints*

dim**=**dim, ng**=**ng, nh**=**nh, *# dimensions*

low**=**low, high**=**high, qmeasures **=** qmeasures) *# boundary constraints*

results **=** solver**.**run()

**return** results

The optimal weights corresponding to the portfolio of Dow's stocks discussed in Sec. 6 Case Study, is shown below.

In [32]:

*#compute optimal weights and the globally maximal Sharpe Ratio*

globlOptimumresult **=** GlobalMaxSharpeRatioOptmzn(meanReturns, covReturns, r0, portfolioSize)

globlOptimumresult**.**print()

xOptimal**=**globlOptimumresult**.**solution

fitness **=** globlOptimumresult**.**best\_fobj

print ("optimum: ", xOptimal)

Continuous Constrained Global Optimization Method Result:

Is converged: True

Best function: -0.08453245101550022

Best solution: [0.09193035598787427 0.00000000000753354986 0.17615287005042135

0.00000000000389473115 0.06936760209176109 0.00000000000000000002

0.00000000000049062761 0.25277808199551655 0.0000000000000842856

0.00000000000006910409 0.00000111755593963079 0.12532223272489576

0.00000000000000072827 0.28544773958133657 0.00000000000001415621]

Quality Measures:

Delta: 2.7755575615628914e-17

Diameter: 1.137710708420947e-07

Deviation: 5.023254398067838e-15

Evaluation: 49016

Best constraint values: [-0.00000000000016753179]

optimum: [0.09193035598787427 0.00000000000753354986 0.17615287005042135

0.00000000000389473115 0.06936760209176109 0.00000000000000000002

0.00000000000049062761 0.25277808199551655 0.0000000000000842856

0.00000000000006910409 0.00000111755593963079 0.12532223272489576

0.00000000000000072827 0.28544773958133657 0.00000000000001415621]

The daily and annualized expected portfolio return and the daily and annualized portfolio risk, have been shown below. **maxSharpeRatio** which represents the globally maximum Sharpe ratio has been computed using the formula shown in equation (1).

In [33]:

*#compute risk returns and max Sharpe Ratio of the VitaOptimum Plus optimal portfolio*

xOptimalArray **=** np**.**array(xOptimal)

Risk **=** np**.**matmul((np**.**matmul(xOptimalArray,covReturns)), np**.**transpose(xOptimalArray))

expReturn **=** np**.**matmul(np**.**array(meanReturns),xOptimalArray**.**T)

annRisk **=** np**.**sqrt(Risk**\***251)

annRet **=** 251**\***np**.**array(expReturn)

maxSharpeRatio **=** (annRet**-**Rf)**/**annRisk

*#set precision for printing results*

np**.**set\_printoptions(precision**=**2, suppress **=** **True**)

*#display results*

print('\n Maximal Sharpe Ratio: %4.2f' **%** maxSharpeRatio)

print('\n Annualized Risk %5.2f' **%** (annRisk), '%')

print('\n Annualized Expected Portfolio Return %5.2f' **%** (annRet), '%')

print('\n Optimal weights (%):\n', xOptimalArray**.**T**\***100 )

Maximal Sharpe Ratio: 1.28

Annualized Risk 14.81 %

Annualized Expected Portfolio Return 21.90 %

Optimal weights (%):

[ 9.19 0. 17.62 0. 6.94 0. 0. 25.28 0. 0. 0. 12.53

0. 28.54 0. ]

Unlike traditional methods, which yield unique optimal solutions to the problem models concerned as decided by the initial points chosen by the solvers, VitaOptimum Plus following heuristics, exhibits the potential to obtain both global optimal or multiple near-optimal solutions by enabling the user to calibrate the quality measures ( **qmeasures**) array, during different runs.

For this demonstration, **qmeasures** was set to qmeasures = np.array([1.0e-16, 1.0e-15, 1.0e-15, 0.0], dtype = ctypes.c\_double). It can be seen that the behavior of VitaOptimum Plus solver is similar to that of the SciPy solver. However, the behavior of the VitaOptimum Plus solver can be controlled by configuring the **qmeasures** array appropriately, to deliver near-optimal or acceptable solutions as desired by the user. The behavior of the solver can also be observed by studying **qmeasures** values before and after optimization. More details on this can be found in the VitaOptimum Plus manual.

In general, this predominant characteristic of heuristic algorithms, which exhibit the potential to deliver both global optima or multiple near-optimum or acceptable solutions, by allowing the users to calibrate their respective control parameters to enhance or alter their behavior, is a boon that can be taken advantage of, while deciding on an effective or practical solution to the problem concerned, from the multifarious choices available.

**8. SciPy solver Vs VitaOptimum Plus solver**

The quality of the optimal portfolios obtained by the **minimize** solver of SciPy and **Ccs** solver of VitaOptimum Plus, were studied over the following experiments / observations.

**8.1 Experiment 1: Diversification Ratios of the optimal portfolios**

Diversification involves investments in different assets or asset classes or markets. A portfolio that comprises such a diversified set of securities can go a long way in mitigating risk since the securities would react differently to market events.

A **Diversification Index** quantifies diversification. There are several diversification indices discussed in the literature. **Diversification Ratio** proposed and patented by Yves Choueifaty in 2008 [CHO 08, CHO 13], is a diversification index of recent origin, built on the inter-dependence between assets of a portfolio. Diversification Ratio is the *ratio of the weighted sum of individual asset volatilities to the portfolio's volatility*.

Let N be the number of assets in the portfolio spanning different asset classes or belonging to a specific class. Let $(\bar{w}=(w\_1,w\_2,...w\_N) )$ be the weights or the proportion of capital to be invested in individual assets in the portfolio and $\bar{w}'$ its transpose. Let $(\bar{\sigma}=(\sigma\_1,\sigma\_2,...\sigma\_N))$ be the standard deviations of returns on the assets and *V*, the variance-covariance matrix of returns on the assets. The Diversification Ratio of a portfolio is given as follows:

Shape

Description automatically generated with medium confidence

**..........(5)**

**A portfolio that is *most diversified* would yield the *maximal Diversification Ratio*.**

The following Python code computes the diversification ratios of the optimal portfolios whose weights, variance-covariance matrix and standard deviations of individual asset returns are known.

(Executing the code as it appears in the following cell, obtains the Diversification Ratio of the portfolio optimized by the VitaOptimum Plus Ccs solver. The same code can be used to obtain the Diversification Ratio of the scipy solver obtained optimal portfolio too.)

In [38]:

*# compute diversification ratio of the optimal portfolio*

AssetWeights **=** xOptimalArray

PortfolioRisk **=** np**.**sqrt(np**.**matmul((np**.**matmul(AssetWeights,\

covReturns)), np**.**transpose(AssetWeights)))

AssetRisk **=** np**.**sqrt(np**.**diagonal(covReturns))

PortfolioDivRatio **=** sum(np**.**multiply(AssetRisk, AssetWeights))**/**PortfolioRisk

print("\n Diversification Ratio %4.2f" **%** PortfolioDivRatio)

Diversification Ratio 1.39

Table 1 illustrates the maximal Sharpe Ratio, Annualized Risk(%), Expected Portfolio Return (%) and Diversification Ratio of the optimal portfolio obtained by SciPy and VitaOptimum Plus solvers.

While SciPy solver delivers a unique optimal solution, VitaOptimum Plus delivers multiple solutions during different runs. For this demonstration, VitaOptimum Plus was experimented with the quality measures set as its default value, viz., qmeasures = np.zeros (4, dtype = ctypes.c\_double) to obtain near optimal solutions that should be of interest to specific investor preferences.

A choice of optimal portfolios delivered by VitaOptimum Plus solver during multiple runs and selected by different investors based on their risk appetites, in the background of maximal Sharpe Ratios of the portfolios, have been shown in the table. It can be seen that some investors have opted for minimal risk or maximal diversification ratio portfolios, while others have opted for maximal return portfolios. The fact that VitaOptimum Plus accommodates such choices due to its potential to deliver multiple near-optimal solutions, contributes to its broad based practical appeal and successful application.

To recall once again, users can configure the behavior of the solver by appropriately setting the values of the array variable **qmeasures**, which describes the four analytical measures of convergence of the **Ccs** solver, in function **GlobalMaxSharpeRatioOptmzn**.

Table

Description automatically generated

**8.2 Experiment 2: Percentage Contributions to Portfolio Risk**

In this experiment, the percentage contributions to portfolio risk by individual assets in the optimal portfolios, obtained by SciPy's *minimize* solver and VitaOptimum Plus's Ccs solver were observed.

The **Marginal Contribution to Risk** $\bar{m}$ is given by,

m¯=(m1,m2,...mN)′=−V.w′¯w′¯.V.w¯

**..........(6)**

and $\sigma\_P$, the **portfolio risk** is given by,

σP=w′¯.V.w¯

**..........(7)**

The **Absolute Contributions to Total Risk** is given bywi.mi

**..........(8)**

and **Percentage contributions to Total Risk** is given by

wi.mi∑wi.mi

**..........(9)**

The python function to compute the percentage contributions to total risk is given below. Here variable **mcr** represents the marginal contribution to risk computed for all the assets in the portfolio. The function needs to be executed over the respective inputs viz., variance-covariance matrix of asset returns, optimal weights and portfolio risk, of the respective portfolios obtained by the two different solvers.

In [37]:

**def** totalcontribrisk(covReturns, AssetWeights, PortfolioRisk):

mcr **=** np**.**matmul(covReturns, np**.**transpose(AssetWeights)) **/** PortfolioRisk

tcr **=** np**.**around(((np**.**multiply(mcr, AssetWeights)**/**PortfolioRisk)),decimals **=**3)

percenttcr **=** np**.**asfarray(tcr, float)**\***100

**return** (percenttcr)

Fig. 1 illustrates the Percentage contributions to total risk computed for the optimal portfolio obtained by the SciPy solver. Fig. 2 illustrates the same for a near-optimal portfolio with minimal risk selected by an investor, using VitaOptimum Plus solver. The characteristics of the optimal/near-optimal portfolios have been listed below the figures.

Chart, bar chart

Description automatically generated

**Fig.1 SciPy minimize solver portfolio : Percentage Contributions to portfolio risk**

Chart, bar chart

Description automatically generated

**Fig. 2 VitaOptimum Plus Ccs solver near-optimal portfolio with minimal risk : Percentage Contributions to portfolio risk**

**8.3 Observation: Optimal Asset Allocation**

The optimal asset allocation represented by the optimal weights and arrived at by SciPy's minimize solver has been shown in Fig. 3. Fig. 4 illustrates the same for a near-optimal minimal risk portfolio opted for by the investor and delivered by VitaOptimum Plus's Ccs solver. The characteristics of the optimal/near-optimal portfolios have been listed below the figures.

It can be seen that for a Maximal Sharpe Ratio of 1.28 and Diversification Ratio of 1.39, the SciPy solver yields an optimal portfolio where nine of the 15 stocks selected by the investor have been excluded from investing (zero weights), which may not be acceptable to the investor. The practicality of the optimal solution given the investor's preference, leaves much to be desired.

On the other hand, VitaOptimum Plus's Ccs solver is able to deliver a minimal risk portfolio, near-optimal though, but with a larger Diversification Ratio of 1.41, as preferred by the investor. It can be seen that the allocation of weights are distributed across assets lending to the practicality of the investment. As mentioned earlier, configuring array variable **qmeasures** can help control or enhance the behavior of the VitaOptimum Plus solver.

Chart, bar chart

Description automatically generated

**Fig.3 Optimal Asset Allocation by SciPy's minimize solver**

Chart

Description automatically generated

**Fig.4 Near-optimal Asset Allocation by VitaOptimum Plus's Ccs solver for a minimal risk portfolio**

**9. Conclusion**

SciPy solver indeed yields a unique and distinct optimal portfolio that reports maximal Sharpe Ratio. However, VitaOptimum Plus driven by heuristics, not just exhibits the potential to deliver a global optimum solution but also multiple near-optimal or acceptable solutions that can cater to the risk appetites or preferences of the investors, resulting in portfolios that are less risky or better diversified or well distributed.

The fact that the behavior of the VitaOptimum Plus solver can be controlled or enhanced by configuring the solver's control parameters is an added advantage that could be made use of by a discerning user.

**Companion Reading**

**Lesson 1 Fundamentals of Risk and Return of a portfolio**

<https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson1_FundaRiskReturnPortfolio/Lesson1_MainContent.ipynb>

**Lesson 3 Heuristic Portfolio Selection**

<https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson3_HeuristicPortfolioSelection/Lesson3_MainContent.ipynb>

**Lesson 5 Mean Variance Optimization Model**

<https://github.com/PaiViji/PythonFinance-PortfolioOptimization/blob/master/Lesson5_MeanVarianceOptimization/Lesson5_MainContent.ipynb>

**References**

[CHO 08] Choueifaty Yves and Y Coignard, Toward Maximum Diversification, *The Journal of Portfolio Management*, pp. 40-51, 2008.

[CHO 13] Choueifaty Yves, T Froidure and J Reynier, Properties of the Most Diversified Portfolio, *Journal of Investment Strategies*, 2(2), pp. 49-70, 2013.

[SHA 66] Sharpe, William F. Mutual Fund Performance, *Journal of Business*, January 1966, pp. 119-138.

[PAI 18] Vijayalakshmi Pai G. A., Metaheuristics for Portfolio Optimization- An Introduction using MATLAB, Wiley-ISTE, 2018. <https://www.mathworks.com/academia/books/metaheuristics-for-portfolio-optimization-pai.html>

[VIT, 2019] <http://skyworkflows.com/wp-content/uploads/2019/07/VitaOptimum-by-SkyWorkflows-small.pdf>.

Mean-Variance Optimization of Portfolios



## 5.1 Introduction

**An investor can take recourse to traditional portfolio construction methods to determine the asset allocation weights.  Lesson 4 Traditional Methods for Portfolio Construction discussed two such methods, viz., Equal weighted portfolio contruction and Inverse volatility weighted portfolio construction.**

**However, is there a possibility where the optimal weights, that obtains the maximum return of a portfolio for a minimal risk, can be determined? In other words, is it possible for an investor to determine the optimal or the best apportionment of capital to assets in the portfolio, so that the investor reaps maximum return for a minimum risk? Tough question, indeed! But thankfully, the answer is a firm Yes!**

**The Markowitz model [MAR 52] built on the Mean-Variance framework of asset returns is an elegant solution to this question. Also known as Mean-Variance Optimization (MVO), the model aims to solve a multi-objective optimization problem subject to basic constraints imposed on the portfolio.**

## 5.2 Mean-Variance Optimization Model

**We briefly recall the return and risk of a portfolio that was discussed in Lesson 1 Fundamentals of risk and return of a portfolio.**

**Let P be a portfolio comprising assets $A\_1, A\_2,...A\_N$ with weights $W\_1, W\_2,...W\_N$ and $\mu\_1, \mu\_2, ...\mu\_N$ as the asset returns. The portfolio return rdetermined by a weighted summation of its individual asset returns is given by,**

**Shape

Description automatically generated with medium confidence**

##### ..........(5.1)

**Portfolio risk is the standard deviation of its returns and is given by,**

**Shape

Description automatically generated with medium confidence**

##### ..........(5.2)

**where $\sigma\_{i,j}$ is the variance-covariance matrix of returns.**

**The Mean-Variance Optimization model is given by:**

**Shape

Description automatically generated with medium confidence**

##### ..........(5.3)

**The model works on two objective functions,  
(i)  to maximize expected portfolio return and  
(ii) to minimize risk of the portfolio  
It therefore aims to solve a non-linear bi-objective optimization problem.**

**The MVO model incorporates the basic constraints of  
(i) the sum of the weights equals 1.  
This means that the investor's capital is fully invested in the portfolio and in such a case, the portfolio is termed as a fully invested portfolio.  
(ii) the weights $W\_i$ lie between 0 and 1.  
This means that either no capital may be allotted for investment in an asset ($W\_i = 0$) or the entire capital may be invested in an asset ($W\_i = 1$) or the capital may be apportioned between assets in the porfolio for weights that range between the interval (0,1).**

## 5.3 Solving the MVO Model

**While there are many ways to solve a bi-objective optimization problem, we choose to solve the problem by decomposing it into two three sub-problem models, viz.,  
(i) obtaining the maximal expected return $R^{MaxRetrn}$ of the portfolio, subject to basic constraints,  
(ii) obtaining the optimal expected return $R^{MinRisk}$ corresponding to the minimum risk portfolio, subject to basic constraints, and finally,  
(iii) obtaining the optimal weights of the portfolio sets that minimize risk and whose returns R lie between $R^{MinRisk}$ and $R^{MaxRetrn}$, (i.e.) $R^{MinRisk} \le R \le R^{MaxRetrn}$, subject to basic constraints.**

**The optimal portfolio set is referred to as the efficient set.**

### 5.3.1 Obtaining the Maximal Expected Return of the Portfolio

**The mathematical model for this sub-problem is given by,**

**Shape

Description automatically generated with medium confidence**

##### ..........(5.4)

**The optimal weights $W\_{i}^{Optimal}$ obtained by solving (5.4) is used to compute $R^{MaxRetrn} = \sum{\left(W\_{i}^{Optimal}.\mu\_i\right)}$**

**The Python function MaximizeReturns employs scipy and NumPy to obtain the optimal weights. Since the problem model is linear, linprog function from the package scipy.optimize is invoked to execute linear programming to solve the problem model.**

### In [33]:

***#function obtains maximal return portfolio using linear programming***

**def MaximizeReturns(MeanReturns, PortfolioSize):**

***#dependencies***

**from scipy.optimize import linprog**

**import numpy as np**

**c = (np.multiply(-1, MeanReturns))**

**A = np.ones([PortfolioSize,1]).T**

**b=[1]**

**res = linprog(c, A\_ub = A, b\_ub = b, bounds = (0,1), method = 'simplex')**

**return res**

### 5.3.2 Obtaining the Optimal Expected Return of a Minimum Risk Portfolio

**The mathematical model for this sub-problem is given by**

**Shape

Description automatically generated with low confidence**

##### ..........(5.5)

**The optimal weights $W\_{i}^{Optimal}$ obtained by solving (5.5) is used to compute $R^{MinRisk} = \sum{\left(W\_{i}^{Optimal}.\mu\_i\right)}$**

**The Python function MinimizeRisk employs scipy.optimize and NumPy to obtain the optimal weights.**

**Functions  f and constraintEq describe the non-linear objective function and the fully invested constraint described in (5.5), respectively. optimize.minimizefunction executes minimization of scalar functions with one or more variables.**

### In [34]:

***#function obtains minimal risk portfolio***

***#dependencies***

**import numpy as np**

**from scipy import optimize**

**def MinimizeRisk(CovarReturns, PortfolioSize):**

**def f(x, CovarReturns):**

**func = np.matmul(np.matmul(x, CovarReturns), x.T)**

**return func**

**def constraintEq(x):**

**A=np.ones(x.shape)**

**b=1**

**constraintVal = np.matmul(A,x.T)-b**

**return constraintVal**

**xinit=np.repeat(0.1, PortfolioSize)**

**cons = ({'type': 'eq', 'fun':constraintEq})**

**lb = 0**

**ub = 1**

**bnds = tuple([(lb,ub) for x in xinit])**

**opt = optimize.minimize (f, x0 = xinit, args = (CovarReturns), bounds = bnds, \**

**constraints = cons, tol = 10\*\*-3)**

**return opt**

### 5.3.3 Obtaining the Optimal Weights for Minimum Risk and Maximum Return Portfolios

**The mathematical model for this sub-problem is defined as follows, where for each R, $R^{MinRisk} \le R \le R^{MaxRetrn}$, the problem model is repeatedly solved to arrive at the optimal weight sets, each of which determines a portfolio that minimizes risk and maximizes return.**

**A black rectangle with a black background

Description automatically generated with low confidence**

##### ..........(5.6)

**The Python function MinimizeRiskConstr employs scipy and NumPy to obtain the optimal weight sets. Functions f, constraintEq and constraintIneq describe the objective function, equality constraint and inequality constraint of (5.6). Function optimize.minimize uses the Trust-Region Constrained algorithm (method = 'trust-constr') to solve the constrained optimization problem with both equality and inequality constraints.**

### In [35]:

***#function obtains Minimal risk and Maximum return portfolios***

***#dependencies***

**import numpy as np**

**from scipy import optimize**

**def MinimizeRiskConstr(MeanReturns, CovarReturns, PortfolioSize, R):**

**def f(x,CovarReturns):**

**func = np.matmul(np.matmul(x,CovarReturns ), x.T)**

**return func**

**def constraintEq(x):**

**AEq=np.ones(x.shape)**

**bEq=1**

**EqconstraintVal = np.matmul(AEq,x.T)-bEq**

**return EqconstraintVal**

**def constraintIneq(x, MeanReturns, R):**

**AIneq = np.array(MeanReturns)**

**bIneq = R**

**IneqconstraintVal = np.matmul(AIneq,x.T) - bIneq**

**return IneqconstraintVal**

**xinit=np.repeat(0.1, PortfolioSize)**

**cons = ({'type': 'eq', 'fun':constraintEq},**

**{'type':'ineq', 'fun':constraintIneq, 'args':(MeanReturns,R) })**

**lb = 0**

**ub = 1**

**bnds = tuple([(lb,ub) for x in xinit])**

**opt = optimize.minimize (f, args = (CovarReturns), method ='trust-constr', \**

**x0 = xinit, bounds = bnds, constraints = cons, tol = 10\*\*-3)**

**return opt**

## 5.3 Case Study

**Let us suppose that an investor decides to invest in a $k$-portfolio ($k$-portfolio 1) comprising the following Dow stocks. (Selection of $k$ portfolio 1, is detailed in Lesson 3 Heuristic Portfolio Selection)**

**𝑘-portfolio 1:**

**{Coca-Cola (KO), United Health (UNH), Walt Disney (DIS), IBM (IBM), Cisco (CSCO), JPMorgan Chase (JPM), Goldman Sachs (GS), Walgreens Boots Alliance (WBA), Apple (AAPL), Home Depot (HD), American Express (AXP), McDonald's (MCD), Merck (MRK), Boeing (BA), Caterpillar (CAT)}**

**The investor desires to explore the optimal portfolio sets that yield maximum expected portfolio return and minimum risk. The objective is to know the optimal weights given an expected return or a desired risk.**

**To apply the MVO model, the historical data set for the $k$-portfolio ( DJIA index: April 11 2014 to April 11, 2019) is cleaned and kept fit for use by the three stage process of optimization discussed in Sec. 5.2. (Lesson 2 Some glimpses of Financial Data Wrangling discusses aspects of data cleaning or data wrangling)**

**The following Python code reads the dataset concerned, computes the stock returns using the Python function StockReturnsComputing and obtains the mean returns and the variance-covariance matrix of returns. (Refer Lesson 1 Fundamentals of Risk and Return of a Portfolio to know about risk and return of a portfolio).**

## In [36]:

***# function computes asset returns***

**def StockReturnsComputing(StockPrice, Rows, Columns):**

**import numpy as np**

**StockReturn = np.zeros([Rows-1, Columns])**

**for j in range(Columns): *# j: Assets***

**for i in range(Rows-1): *# i: Daily Prices***

**StockReturn[i,j]=((StockPrice[i+1, j]-StockPrice[i,j])/StockPrice[i,j])\* 100**

**return StockReturn**

## In [37]:

***# Obtain optimal portfolio sets that maximize return and minimize risk***

***#Dependencies***

**import numpy as np**

**import pandas as pd**

***#input k-portfolio 1 dataset comprising 15 stocks***

**StockFileName = 'DJIA\_Apr112014\_Apr112019\_kpf1.csv'**

**Rows = 1259 *#excluding header***

**Columns = 15 *#excluding date***

**portfolioSize = Columns *#set portfolio size***

***#read stock prices in a dataframe***

**df = pd.read\_csv(StockFileName, nrows= Rows)**

***#extract asset labels***

**assetLabels = df.columns[1:Columns+1].tolist()**

**print(assetLabels)**

***#extract asset prices***

**StockData = df.iloc[0:, 1:]**

***#compute asset returns***

**arStockPrices = np.asarray(StockData)**

**[Rows, Cols]=arStockPrices.shape**

**arReturns = StockReturnsComputing(arStockPrices, Rows, Cols)**

***#compute mean returns and variance covariance matrix of returns***

**meanReturns = np.mean(arReturns, axis = 0)**

**covReturns = np.cov(arReturns, rowvar=False)**

***#set precision for printing results***

**np.set\_printoptions(precision=3, suppress = True)**

***#display mean returns and variance-covariance matrix of returns***

**print('Mean returns of assets in k-portfolio 1\n', meanReturns)**

**print('Variance-Covariance matrix of returns\n', covReturns)**

**['AAPL', 'AXP', 'BA', 'CAT', 'CSCO', 'DIS', 'GS', 'HD', 'IBM', 'JPM', 'KO', 'MCD', 'MRK', 'UNH', 'WBA']**

**Mean returns of assets in k-portfolio 1**

**[ 0.09 0.029 0.1 0.039 0.081 0.04 0.033 0.085 -0.016 0.06**

**0.019 0.057 0.036 0.095 -0.002]**

**Variance-Covariance matrix of returns**

**[[2.375 0.672 0.962 1.042 0.999 0.68 0.954 0.726 0.709 0.825 0.306 0.458**

**0.534 0.774 0.697]**

**[0.672 1.648 0.8 0.95 0.7 0.569 1.065 0.658 0.663 1.001 0.307 0.35**

**0.556 0.718 0.667]**

**[0.962 0.8 2.288 1.31 0.89 0.716 1.066 0.747 0.777 0.977 0.381 0.472**

**0.578 0.745 0.679]**

**[1.042 0.95 1.31 2.733 1.041 0.688 1.321 0.796 0.885 1.169 0.358 0.455**

**0.616 0.72 0.681]**

**[0.999 0.7 0.89 1.041 1.789 0.713 0.927 0.724 0.817 0.909 0.362 0.477**

**0.647 0.656 0.707]**

**[0.68 0.569 0.716 0.688 0.713 1.35 0.773 0.586 0.574 0.717 0.302 0.368**

**0.466 0.557 0.631]**

**[0.954 1.065 1.066 1.321 0.927 0.773 2.114 0.795 0.803 1.554 0.303 0.467**

**0.705 0.82 0.819]**

**[0.726 0.658 0.747 0.796 0.724 0.586 0.795 1.39 0.619 0.753 0.343 0.472**

**0.487 0.659 0.689]**

**[0.709 0.663 0.777 0.885 0.817 0.574 0.803 0.619 1.632 0.767 0.372 0.391**

**0.576 0.564 0.534]**

**[0.825 1.001 0.977 1.169 0.909 0.717 1.554 0.753 0.767 1.702 0.324 0.483**

**0.675 0.761 0.717]**

**[0.306 0.307 0.381 0.358 0.362 0.302 0.303 0.343 0.372 0.324 0.806 0.36**

**0.384 0.31 0.355]**

**[0.458 0.35 0.472 0.455 0.477 0.368 0.467 0.472 0.391 0.483 0.36 1.086**

**0.402 0.43 0.433]**

**[0.534 0.556 0.578 0.616 0.647 0.466 0.705 0.487 0.576 0.675 0.384 0.402**

**1.504 0.615 0.64 ]**

**[0.774 0.718 0.745 0.72 0.656 0.557 0.82 0.659 0.564 0.761 0.31 0.43**

**0.615 1.722 0.78 ]**

**[0.697 0.667 0.679 0.681 0.707 0.631 0.819 0.689 0.534 0.717 0.355 0.433**

**0.64 0.78 2.554]]**

**As the first stage to executing the MVO, the mathematical model discussed in Sec. 5.3.1 is applied to the dataset concerned and the maximum expected portfolio return $R^{MaxRetrn}$ is obtained. The following Python code shows the invocation of function MaximizeReturns to arrive at the maximum return.**

## In [38]:

***#Maximal expected portfolio return computation for the k-portfolio***

**result1 = MaximizeReturns(meanReturns, portfolioSize)**

**maxReturnWeights = result1.x**

**maxExpPortfolioReturn = np.matmul(meanReturns.T, maxReturnWeights)**

**print("Maximal Expected Portfolio Return: %7.4f" % maxExpPortfolioReturn )**

**Maximal Expected Portfolio Return: 0.0997**

**In the second stage, the mathematical model discussed in Sec. 5.3.2 is applied to the dataset and $R^{MinRisk}$, the expected portfolio return corresponding to the minimum risk is determined. The following Python code shows the invocation of function MinimizeRisk to arrive at the return corresponding to minimum risk.**

## In [39]:

***#expected portfolio return computation for the minimum risk k-portfolio***

**result2 = MinimizeRisk(covReturns, portfolioSize)**

**minRiskWeights = result2.x**

**minRiskExpPortfolioReturn = np.matmul(meanReturns.T, minRiskWeights)**

**print("Expected Return of Minimum Risk Portfolio: %7.4f" % minRiskExpPortfolioReturn)**

**Expected Return of Minimum Risk Portfolio: 0.0361**

**In the third stage, the mathematical model discussed in Sec. 5.3.3 is applied, employing function MinimizeRiskConstr. The following Python code shows the repeated invocation of the function controlled by a while loop that varies from low to high in steps of increment = 0.001. Here low and high denote $R^{MinRisk}$ and $R^{MaxReturn}$ respectively. The loop therefore executes (5.6) repeatedly varying R as defined by $R^{MinRisk} \le R \le R^{MaxRetrn}$.**

## In [42]:

***#compute efficient set for the maximum return and minimum risk portfolios***

**increment = 0.001**

**low = minRiskExpPortfolioReturn**

**high = maxExpPortfolioReturn**

***#initialize optimal weight set and risk-return point set***

**xOptimal =[]**

**minRiskPoint = []**

**expPortfolioReturnPoint =[]**

***#repeated execution of function MinimizeRiskConstr to determine the efficient set***

**while (low < high):**

**result3 = MinimizeRiskConstr(meanReturns, covReturns, portfolioSize, low)**

**xOptimal.append(result3.x)**

**expPortfolioReturnPoint.append(low)**

**low = low+increment**

***#gather optimal weight set***

**xOptimalArray = np.array(xOptimal)**

***#obtain annualized risk for the efficient set portfolios***

***#for trading days = 251***

**minRiskPoint = np.diagonal(np.matmul((np.matmul(xOptimalArray,covReturns)),\**

**np.transpose(xOptimalArray)))**

**riskPoint = np.sqrt(minRiskPoint\*251)**

***#obtain expected portfolio annualized return for the***

***#efficient set portfolios, for trading days = 251***

**retPoint = 251\*np.array(expPortfolioReturnPoint)**

***#display efficient set portfolio parameters***

**print("Size of the efficient set:", xOptimalArray.shape )**

**print("Optimal weights of the efficient set portfolios: \n", xOptimalArray)**

**print("Annualized Risk and Return of the efficient set portfolios: \n", \**

**np.c\_[riskPoint, retPoint])**

**Size of the efficient set: (64, 15)**

**Optimal weights of the efficient set portfolios:**

**[[ 0.028 0.039 0.023 0.017 0.026 0.103 0.017 0.05 0.032 0.021**

**0.319 0.192 0.062 0.047 0.025]**

**[ 0.028 0.039 0.023 0.017 0.026 0.108 0.017 0.051 0.032 0.022**

**0.311 0.192 0.063 0.047 0.025]**

**[ 0.029 0.039 0.024 0.017 0.027 0.108 0.017 0.052 0.031 0.022**

**0.309 0.192 0.062 0.048 0.024]**

**[ 0.03 0.039 0.025 0.017 0.027 0.107 0.017 0.053 0.03 0.022**

**0.306 0.192 0.062 0.049 0.024]**

**[ 0.031 0.038 0.024 0.016 0.028 0.086 0.016 0.059 0.028 0.022**

**0.316 0.197 0.06 0.056 0.023]**

**[ 0.033 0.039 0.027 0.017 0.029 0.103 0.017 0.06 0.028 0.023**

**0.294 0.185 0.064 0.056 0.023]**

**[ 0.034 0.039 0.028 0.017 0.03 0.102 0.017 0.063 0.027 0.023**

**0.29 0.184 0.064 0.059 0.023]**

**[ 0.035 0.038 0.03 0.017 0.031 0.101 0.017 0.066 0.026 0.024**

**0.285 0.183 0.064 0.062 0.022]**

**[ 0.036 0.038 0.031 0.018 0.033 0.099 0.017 0.069 0.025 0.024**

**0.277 0.184 0.062 0.065 0.022]**

**[ 0.039 0.034 0.034 0.017 0.036 0.058 0.018 0.069 0.023 0.026**

**0.298 0.205 0.052 0.071 0.021]**

**[ 0.04 0.033 0.036 0.017 0.037 0.056 0.017 0.072 0.022 0.026**

**0.295 0.203 0.051 0.075 0.021]**

**[ 0.041 0.032 0.037 0.017 0.038 0.055 0.017 0.075 0.02 0.026**

**0.291 0.202 0.05 0.079 0.02 ]**

**[ 0.05 0.039 0.047 0.026 0.048 0.053 0.026 0.07 0.027 0.036**

**0.242 0.179 0.054 0.073 0.028]**

**[ 0.052 0.038 0.048 0.026 0.049 0.051 0.026 0.072 0.026 0.037**

**0.24 0.179 0.053 0.076 0.028]**

**[ 0.052 0.038 0.049 0.026 0.049 0.05 0.026 0.073 0.025 0.037**

**0.24 0.179 0.053 0.077 0.027]**

**[ 0.049 0.027 0.047 0.016 0.046 0.044 0.016 0.082 0.015 0.028**

**0.276 0.202 0.044 0.092 0.016]**

**[ 0.053 0.025 0.051 0.016 0.049 0.04 0.016 0.083 0.014 0.029**

**0.273 0.202 0.041 0.095 0.015]**

**[ 0.056 0.024 0.055 0.016 0.052 0.036 0.015 0.084 0.012 0.03**

**0.268 0.202 0.039 0.097 0.014]**

**[ 0.059 0.022 0.059 0.016 0.055 0.032 0.015 0.084 0.011 0.031**

**0.264 0.201 0.036 0.1 0.013]**

**[ 0.063 0.021 0.064 0.015 0.059 0.028 0.015 0.085 0.01 0.032**

**0.259 0.201 0.034 0.102 0.012]**

**[ 0.066 0.019 0.068 0.015 0.062 0.025 0.015 0.086 0.008 0.033**

**0.254 0.2 0.032 0.105 0.011]**

**[ 0.069 0.018 0.071 0.015 0.064 0.022 0.015 0.087 0.007 0.033**

**0.25 0.2 0.03 0.107 0.011]**

**[ 0.069 0.017 0.072 0.015 0.065 0.021 0.015 0.088 0.007 0.033**

**0.251 0.201 0.029 0.108 0.01 ]**

**[ 0.068 0.016 0.076 0.01 0.064 0.025 0.01 0.103 0.007 0.024**

**0.237 0.201 0.028 0.121 0.009]**

**[ 0.071 0.015 0.08 0.009 0.068 0.023 0.009 0.103 0.007 0.023**

**0.232 0.201 0.025 0.126 0.008]**

**[ 0.066 0.015 0.083 0.01 0.072 0.021 0.01 0.111 0.007 0.025**

**0.219 0.198 0.024 0.132 0.009]**

**[ 0.071 0.013 0.093 0.009 0.078 0.019 0.009 0.108 0.007 0.024**

**0.214 0.197 0.02 0.129 0.008]**

**[ 0.067 0.012 0.079 0.009 0.069 0.016 0.009 0.125 0.006 0.025**

**0.207 0.197 0.022 0.153 0.007]**

**[ 0.069 0.01 0.082 0.008 0.072 0.013 0.008 0.127 0.005 0.025**

**0.202 0.197 0.019 0.157 0.006]**

**[ 0.069 0.01 0.083 0.007 0.073 0.012 0.007 0.128 0.004 0.024**

**0.202 0.198 0.018 0.159 0.005]**

**[ 0.07 0.009 0.084 0.007 0.073 0.011 0.007 0.129 0.004 0.024**

**0.202 0.199 0.017 0.16 0.005]**

**[ 0.066 0.013 0.086 0.008 0.072 0.029 0.007 0.148 0.006 0.017**

**0.144 0.191 0.029 0.177 0.006]**

**[ 0.068 0.013 0.09 0.007 0.074 0.027 0.007 0.151 0.006 0.017**

**0.135 0.19 0.027 0.183 0.006]**

**[ 0.065 0.012 0.09 0.007 0.073 0.025 0.007 0.161 0.006 0.017**

**0.123 0.194 0.025 0.189 0.006]**

**[ 0.067 0.012 0.094 0.007 0.076 0.024 0.007 0.163 0.005 0.016**

**0.114 0.194 0.023 0.193 0.006]**

**[ 0.068 0.011 0.098 0.007 0.078 0.022 0.007 0.166 0.005 0.016**

**0.105 0.193 0.021 0.199 0.006]**

**[ 0.071 0.011 0.106 0.007 0.086 0.021 0.007 0.162 0.005 0.016**

**0.099 0.184 0.018 0.202 0.005]**

**[ 0.077 0.008 0.11 0.005 0.088 0.011 0.005 0.164 0.004 0.011**

**0.107 0.191 0.01 0.204 0.005]**

**[ 0.073 0.009 0.117 0.007 0.083 0.015 0.007 0.179 0.005 0.016**

**0.079 0.184 0.014 0.205 0.005]**

**[ 0.074 0.009 0.121 0.007 0.085 0.015 0.007 0.183 0.005 0.017**

**0.067 0.181 0.014 0.209 0.005]**

**[ 0.075 0.009 0.125 0.007 0.086 0.015 0.007 0.188 0.005 0.017**

**0.055 0.179 0.014 0.214 0.005]**

**[ 0.075 0.009 0.129 0.007 0.088 0.014 0.007 0.191 0.005 0.017**

**0.046 0.177 0.014 0.217 0.005]**

**[ 0.075 0.008 0.13 0.006 0.088 0.014 0.006 0.193 0.004 0.016**

**0.045 0.177 0.013 0.218 0.005]**

**[ 0.078 0.008 0.138 0.006 0.084 0.013 0.006 0.201 0.005 0.016**

**0.021 0.178 0.013 0.227 0.005]**

**[ 0.08 0.008 0.144 0.006 0.085 0.012 0.006 0.204 0.004 0.015**

**0.011 0.175 0.012 0.231 0.005]**

**[ 0.082 0.008 0.15 0.006 0.085 0.012 0.006 0.207 0.004 0.015**

**0.001 0.173 0.011 0.235 0.005]**

**[ 0.081 0.004 0.156 0.004 0.083 0.008 0.004 0.218 0.002 0.009**

**0.015 0.164 0.008 0.242 0.002]**

**[ 0.086 0.003 0.159 0.004 0.079 0.006 0.003 0.223 0.002 0.008**

**0.014 0.15 0.007 0.253 0.002]**

**[ 0.085 0.002 0.164 0.003 0.076 0.003 0.003 0.221 0.001 0.006**

**0.003 0.171 0.002 0.259 0.001]**

**[ 0.086 0.002 0.165 0.002 0.076 0.002 0.002 0.222 0.001 0.005**

**0.003 0.172 0.002 0.26 0.001]**

**[ 0.088 0.002 0.177 0.003 0.072 0.003 0.003 0.233 0.001 0.006**

**0.003 0.117 0.003 0.288 0.001]**

**[ 0.091 0.002 0.186 0.003 0.07 0.003 0.003 0.235 0.001 0.006**

**0.003 0.093 0.003 0.301 0.001]**

**[ 0.097 0.003 0.195 0.003 0.064 0.004 0.003 0.248 0.001 0.006**

**0.003 0.058 0.004 0.309 0.002]**

**[ 0.109 0.003 0.207 0.003 0.059 0.004 0.003 0.242 0.001 0.006**

**0.003 0.038 0.004 0.317 0.002]**

**[ 0.119 0.002 0.222 0.003 0.038 0.003 0.002 0.241 0.001 0.006**

**0.003 0.024 0.003 0.331 0.001]**

**[ 0.112 0.002 0.24 0.002 0.038 0.003 0.002 0.236 0.001 0.005**

**0.003 0.007 0.003 0.344 0.001]**

**[ 0.091 0.001 0.247 0.001 0.022 0.001 0.001 0.264 0.001 0.002**

**0.001 0.004 0.001 0.361 0.001]**

**[ 0.096 0.001 0.301 0.001 0.018 0.002 0.001 0.174 0.001 0.003**

**0.001 0.004 0.002 0.394 0.001]**

**[ 0.098 0. 0.315 0. 0.006 0.001 0. 0.168 0. 0.001**

**0. 0.001 0.001 0.407 0. ]**

**[ 0.088 0.001 0.382 0.001 0.012 0.001 0.001 0.067 0. 0.001**

**0.001 0.003 0.001 0.44 0.001]**

**[ 0.062 0.001 0.433 0.001 0.006 0.001 0.001 0.031 0. 0.001**

**0.001 0.002 0.001 0.46 0. ]**

**[ 0.048 0. 0.461 0. 0.002 0. 0. 0.027 -0. -0.**

**0. -0. 0. 0.461 0. ]**

**[ 0.045 -0. 0.467 -0. 0.001 -0. -0. 0.026 -0.001 -0.**

**-0. -0.001 -0. 0.464 -0. ]**

**[ 0.006 -0. 0.642 0. 0. 0. -0. -0. -0. -0.**

**-0. -0.001 -0. 0.355 -0. ]]**

**Annualized Risk and Return of the efficient set portfolios:**

**[[11.571 9.049]**

**[11.585 9.3 ]**

**[11.592 9.551]**

**[11.599 9.802]**

**[11.591 10.053]**

**[11.641 10.304]**

**[11.656 10.555]**

**[11.675 10.806]**

**[11.701 11.057]**

**[11.706 11.308]**

**[11.725 11.559]**

**[11.743 11.81 ]**

**[11.991 12.061]**

**[12.013 12.312]**

**[12.023 12.563]**

**[11.86 12.814]**

**[11.905 13.065]**

**[11.953 13.316]**

**[12.004 13.567]**

**[12.056 13.818]**

**[12.11 14.069]**

**[12.161 14.32 ]**

**[12.174 14.571]**

**[12.201 14.822]**

**[12.258 15.073]**

**[12.339 15.324]**

**[12.413 15.575]**

**[12.427 15.826]**

**[12.487 16.077]**

**[12.509 16.328]**

**[12.522 16.579]**

**[12.75 16.83 ]**

**[12.837 17.081]**

**[12.913 17.332]**

**[13.003 17.583]**

**[13.096 17.834]**

**[13.206 18.085]**

**[13.239 18.336]**

**[13.41 18.587]**

**[13.525 18.838]**

**[13.643 19.089]**

**[13.741 19.34 ]**

**[13.765 19.591]**

**[13.979 19.842]**

**[14.1 20.093]**

**[14.223 20.344]**

**[14.229 20.595]**

**[14.354 20.846]**

**[14.412 21.097]**

**[14.439 21.348]**

**[14.775 21.599]**

**[14.969 21.85 ]**

**[15.212 22.101]**

**[15.426 22.352]**

**[15.647 22.603]**

**[15.866 22.854]**

**[15.982 23.105]**

**[16.396 23.356]**

**[16.593 23.607]**

**[17.243 23.858]**

**[17.767 24.109]**

**[18.016 24.36 ]**

**[18.097 24.611]**

**[19.449 24.862]]**

**The efficient set comprises 64 different optimal portfolios. Each of the optimal weight sets can be seen to satisfy all the constraints imposed on them. The optimal weight sets and the annualized risk(%) and return (%) of each of the optimal portfolios, are now available to the investor for making prudent decisions on the portfolios.**

**To get back to the investor's objective, if a risk seeking investor aspires to get an annual return of y% mindless of the risk involved, then the efficient set allows selecting that portfolio whose risk-return point is [x%, y%] for some x% risk. The corresponding optimal weight set indicates how the investor should apportion the capital on the various assets comprising the $k$-portfolio, to reap the return desired.**

**In contrast, a risk-averse investor who is too concerned about the risk and hence fixes a risk of x%, can also get to know about the corresponding return (y%) that the portfolio will fetch for that risk, besides knowing about the optimal weights that are required for making such an investment.**

**The efficient set therefore provides a gamut of optimal investment decisions to suit the investor's risk appetite.**

**Example**

**To keep it simple, let us consider a risk averse investor who desires to invest in the aforementioned $k$-portfolio of Dow stocks, with the lowest risk possible. The efficient set lists the pair [11.571 9.049] which is the first pair in the efficient set output, offering the lowest annualized risk of 11.571%. If the investor agrees to this risk, then the investor can expect an annual return of 9.049%. The optimal weights that will effect this are as follows: [0.028 0.039 0.023 0.017 0.026 0.103 0.017 0.05 0.032 0.021 0.319 0.192 0.062 0.047 0.025].**

**To elaborate further, if the investor owns a capital of USD10000, to gain an annual expected return of 9.049% holding an annualized risk of 11.571%, the investor will have to allot the capital to the assets in the $k$-portfolio, in the following manner:**

**['AAPL', 2.8%], ['AXP', 3.9%], ['BA', 2.3%], ['CAT', 1.7%], ['CSCO', 2.6%], ['DIS', 10.3%], ['GS', 1.7%], ['HD', 5%], ['IBM', 3.2%], ['JPM', 2.1%], ['KO', 31.9%], ['MCD', 19.2%], ['MRK', 6.2%], ['UNH', 4.7%], ['WBA', 2.5%]**

## 5.4 Efficient Frontier

**The efficient set obtained by the Mean-Variance Optimization model can be graphically represented by what is called an efficient frontier. An efficient frontier is a risk-return tradeoff graph, which describes a set of optimal portfolios that yield the highest expected portfolio return for a defined level of risk or the lowest possible risk for a defined level of expected portfolio return. It graphs the optimal structure of the portfolio which yields the maximum expected return for a given level of risk or vice-versa.**

**Portfolio optimization strives to build portfolios which are on the efficient frontier and not below it, for these are sub-optimal. The set of all optimal portfolios that lie on the efficient frontier, in other words, those portfolios that generate the largest return for a given level of risk or vice-versa, are also known as the Markowitz efficient set.**

**Given the efficient set, generated for the $k$-portfolio of Dow stocks discussed in Sec. 5.3, the following Python code traces the efficient frontier using matplotlib library.**

## In [41]:

***#Graph Efficient Frontier***

**import matplotlib.pyplot as plt**

**NoPoints = riskPoint.size**

**colours = "blue"**

**area = np.pi\*3**

**plt.title('Efficient Frontier for k-portfolio 1 of Dow stocks')**

**plt.xlabel('Annualized Risk(%)')**

**plt.ylabel('Annualized Expected Portfolio Return(%)' )**

**plt.scatter(riskPoint, retPoint, s=area, c=colours, alpha =0.5)**

**plt.show()**

## A picture containing scatter chart Description automatically generated

**The efficient frontier for a portfolio serves to provide a road-map for the choices that investors can make, based on their risk appetites.**

**For a specific annualized risk (x%) or a range of values desired by the investor, the corresponding annualized expected portfolio return (y%) or a range of return values or vice-versa, can be easily obtained by a visual inspection of the graph. The point(s) of intersection on the efficient frontier which represent(s) the optimal portfolios, can be easily decoded to arrive at the optimal weights which determine the optimal allocations of capital to the assets in the portfolio, corresponding to the choice of risk or return.**

**Fig. 5.1 illustrates how an efficient frontier can direct a risk-seeking investor who desires an annualized expected return of 20%, to invest in a portfolio that holds a 14% annualized risk, while enlightening the investor on the optimal weights which will ensure the aspired risk/return.**

**Chart

Description automatically generated**

#### Fig. 5.1 Usefulness of efficient frontiers to investors - an illustration

## 5.5 Efficient Frontiers: $k$-portfolios vs "Ideal" portfolio

**Lesson 3 Heuristic Portfolio Selection discussed three different selections of $k$-portfolios comprising Dow stocks. Labelled as $k$-portfolio 1, 2 and 3, the behaviour of these portfolios were compared with that of an "ideal" portfolio where the investor decides to invest in all the stocks of the "mini-universe". The observations discussed in Sec. 4.4 of Lesson 4 Traditional Methods for Portfolio Construction showed that $k$-portfolios were endowed with merits that rendered them advantageous to the investors.**

**In this lesson, we investigate the risk-return trade-off behaviour of the $k$-portfolios, by tracing their efficient frontiers and comparing the same with that of the "ideal" portfolio. The Mean-Variance Optimization model was applied over all the portfolios. Fig. 5.2 illustrates the efficient frontiers traced for $k$-portfolio 1, $k$-portfolio 2, $k$-portfolio 3 and the "ideal" portfolio, over the DJIA Index data set (April 2014 - April 2019).**

**The CSV files DJIA\_Apr112014\_Apr112019\_kpf1.csv, DJIA\_Apr112014\_Apr112019\_kpf2.csv, DJIA\_Apr112014\_Apr112019\_kpf3.csv hold the datasets for the respective $k$-portfolios 1,2 and 3, and DJIA\_Apr112014\_Apr112019.csv, the dataset for the Dow "mini-universe".**

**The proximity of the $k$-portfolio efficient frontiers to that of the "ideal" portfolio reveals the similarity of risk-return trade-off behaviour of the $k$-portfolios, to that of the "ideal" portfolio. The proximity of the $k$-portfolio efficient frontiers to one another, also reveals the similarity of their portfolio behaviour despite holding different sets of assets that were randomly selected, one from each of the clusters, during their construction. (Refer Sec 3.4 of  Lesson 3 Heuristic Portfolio Selection for the construction of $k$-portfolios of Dow stocks)**

**Chart, line chart, scatter chart

Description automatically generated**

#### Fig. 5.2 Performance Comparison of efficient frontiers of $k$-portfolios with the "ideal" efficient frontier, for DJIA Index (April 2014-Aprl 2019)

**The efficient frontiers traced by the Mean-Variance Optimization model or known as the Markowitz model, are termed as “exact” or “ideal” in the literature. The Markowitz model merely deals with basic constraints imposed over a bi-criterion objective function. The model can be easily solved using a variety of traditional methods including the one discussed in this lesson.**

**In reality, portfolio optimization problem models can turn too complex for direct solving by traditional methods. Thus when constraints reflective of investor preferences or investment strategies or market norms or religious laws etc., are included, the problem models can turn complex, warranting the need to look for non-traditional, nature-inspired methods, referred to as metaheuristics in recent literature, to arrive at acceptable if not accurate solutions. In the face of these models, a Markowitz model is often dubbed as an “Unconstrained Optimization” problem for it can be easily solved with the simplest of the traditional techniques.**

### Companion Reading

**This blog is an abridged adaptation of concepts discussed in Chapter 1 and Chapter 3 of [PAI 18] to Dow Jones dataset (DJIA index: April, 2014- April, 2019) and implemented in Python. Readers (read "worker bees"), seeking more information may refer to the corresponding chapter in the book.**

### References

**[MAR 52] Markowitz H., Portfolio Selection, The Journal of Finance, vol. 7, no. 1, pp. 77-91, Mar., 1952.**

**[PAI 18] Vijayalakshmi Pai G. A., Metaheuristics for Portfolio Optimization- An Introduction using MATLAB, Wiley-ISTE, 2018.**[**https://www.mathworks.com/academia/books/metaheuristics-for-portfolio-optimization-pai.html**](https://www.mathworks.com/academia/books/metaheuristics-for-portfolio-optimization-pai.html)