Mathematical Modeling Portfolio

Using nonlinear programming to find an optimal solution for stock portfolios to maximize returns on investments at an acceptable risk level.

David Jackson

To use nonlinear programming to find optimal solutions for stock portfolios, we first need to collect data on the stocks in which wo will invested in for project. We will use the top 10 stocks in the S&P 500. This could be done with any number of stocks, depending on what you want to invest in. The purpose of using nonlinear programming for portfolio optimization is not to determine what to invest in but to determine how much stock you should buy in each of the companies that you want to invest in while accounting for an acceptable risk level.

There are many applications of nonlinear programming, with the optimization of stock portfolios being one of them. Nonlinear programming helps in training machine learning models, optimizing model parameters to achieve the desired performance. Nonlinear programming is used in economic models for resource allocation, production planning, and cost minimization. Nonlinear programming is used to optimize design parameters for various structures and systems. Nonlinear programming helps design and optimize control systems for various applications, such as robotics and process control. Nonlinear programming is used in optimizing chemical processes, such as reaction optimization, distillation, and energy recovery. Nonlinear programming can be used to optimize water resource allocation, pollution control, and infrastructure design. Nonlinear programming can be used to optimize transportation networks, such as logistics and supply chain management, considering factors like economies of scale, capacity constraints, and transportation costs. As we can see, the applications of nonlinear programming are numerous and a very useful tool in many different branches of science and sectors of the economy.

After finding the stocks that you want to use, you will need to import the data into a software of your choice (I used RStudio). The 10 different stocks we used were Walmart, Amazon, Apple, UnitedHealth Group, Berkshire Hathaway, CVS Health, Exxon Mobil, Alphabet, McKesson, and Cencora. Using data from the last 10 years (2015-2025) using the open stock price that is the price that the stock starts to sell at the beginning of the trading day, we were able to find the average daily simple return, which is given by.

$$\text{simple return} = \frac{(x_n - x_{n-1})}{x_{n-1}}$$

Where x_n is the open stock price at day n so x_{n-1} is the stock price of the previous day

The average log return is given by

$$\text{log return} = log(x_n) - log(x_{n-1})$$

Also note that log is the natural logarithm.

The average annual simple return is given by

$$\text{average annual simple return} = \text{average daily simple return} \cdot 252$$

And the average annual log return of the stocks is given by

$$\text{average annual log return} = \text{average daily log return} \cdot 252$$

Note that 252 is the number of trading days in a year.

Once you have all the data from your stocks, you can also find the variance-covariance matrix, which you will need for the optimization along with the opening price and the returns. Depending on where you get your data from, you may also need to filter the data so that it will work with your software. All of the data we used had dollar signs in it, which will have to be removed for the software to calculate the needed values. After you have all of your data collected and calculated all of the values you need and have your variance-covariance Matrix, you will want to import these into Excel. It may be easier to have an individual Excel file for each of these. *(I put all of the stock data, variances, average daily simple returns, average daily log returns, average annual simple returns, and average annual log returns into one Excel document and the variance-covariance Matrix into another document. It is easier to import into Python or another coding language that way).*

The objective function for our nonlinear program is given below

Minimize

$$\begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} \sigma_{11}^2 & c_{12} & c_{13} & \cdots & c_{1m} \\ c_{21} & \sigma_{22}^2 & c_{23} & \cdots & c_{2m} \\ c_{31} & c_{32} & \sigma_{33}^2 & \cdots & c_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & c_{n3} & \cdots & \sigma_{nm}^2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Subject to the constraints

$$p_1x_1 + p_2x_2 + \cdots + p_nx_n \leq B$$

$$r_1x_1 + r_2x_2 + \cdots + r_nx_n \geq G$$

$$x_n \geq 0$$

Or this

Minimize

$$Z = x_1^2\sigma_{11}^2 + x_2^2\sigma_{22}^2 + x_3^2\sigma_{33}^2 + \cdots + x_n^2\sigma_{nn}^2 + 2x_1x_2c_{12} + 2x_1x_3c_{13} + \cdots + 2x_{n-1}x_nc_{n-1,n}$$

Subject to the constraints

$$p_1x_1 + p_2x_2 + \cdots + p_nx_n \leq B$$

$$r_1x_1 + r_2x_2 + \cdots + r_nx_n \geq G$$

$$x_n \geq 0$$

Where σ_{nm}^2 is the variance of the nth or mth stock, c_{nm} is the covariance of the nth stock with the mth stock, and x_n denotes the nth stock. p_n represents the price of the nth stock, and B is the budget you have to purchase the stocks. r_n represents the return of the nth stock, and G represents the minimal acceptable gains of the stocks. Your covariance matrix should be a square matrix; if it is not, then you did something wrong. This is why the variance of the nth stock or the mth stock does not matter, because along the diagonal $n = m$, so it doesn't matter. In addition, the covariance of the stocks should be the same, mirrored across the diagonal axis of the matrix. That is, c_{nm} and c_{mn} are the same.

For the stock portfolio, I chose the minimal acceptable return to be 5% with a budget of $15,000, and the nonlinear program gave us our optimal results below.

Optimal stock allocations: Simple returns

Walmart: 14.7074 shares

Amazon: 30.6852 shares

Apple: 20.7927 shares

UnitedHealth Group: 0.0000 shares

Berkshire Hathaway: 0.0000 shares

CVS Health: 0.0000 shares

Exxon Mobil: 2.5987 shares

Alphabet: 21.7625 shares

McKesson: 0.0000 shares

Cencora: 0.0000 shares

Expected portfolio return: 0.0821

Total cost: 15000.00

It is easy to replace the minimal acceptable return and your budget in the program to see how the results will change.

Redoing the optimization with daily mean log returns, we get the following results

Optimal stock allocations: Log returns

Walmart: 15.4808 shares

Amazon: 30.8874 shares

Apple: 21.3957 shares

UnitedHealth Group: 0.0000 shares

Berkshire Hathaway: 0.0000 shares

CVS Health: 0.0000 shares

Exxon Mobil: 0.0000 shares

Alphabet: 22.1755 shares

McKesson: 0.0000 shares

Cencora: 0.0000 shares

Expected portfolio return: 0.0671

Total cost: 15000.00

Log returns are used in portfolio optimization because they are easier to work with mathematically. They tend to be closer to a normal distribution for statistical analysis, they better represent the compounding effect, and prevent negative asset prices, all of which contribute to more accurate and reliable portfolio analysis and modeling.
For stock portfolio optimization, it's generally recommended to use daily returns rather than annual returns. While annual returns are a common metric for evaluating overall portfolio performance, daily returns are better and provide a more precise evaluation of how individual assets and the portfolio are performing over time.
Although we did calculate the annual simple return and the annual log return, we will not be using them in the optimization, they were used to verify existing data and to understand our stocks better.
We can see that the results from the simple daily returns and the log daily returns are slightly different. The simple returns gave us a higher expected portfolio return, than the log results, but the log results may be more accurate for the reasons listed before.

We also made a nonlinear program to find an optimal solution with a maximum acceptable risk and minimal acceptable return, with a budget, and got the following results.

Optimal stock allocations (to maximize return under risk and budget constraints): Simple returns

Walmart: 9.9339 shares

Amazon: 5.9675 shares

Apple: 2.0824 shares

UnitedHealth Group: 1.8647 shares

Berkshire Hathaway: 0.0128 shares

CVS Health: 0.0000 shares

Exxon Mobil: 0.0000 shares

Alphabet: 3.6413 shares

McKesson: 1.1413 shares

Cencora: 0.9178 shares

Expected portfolio return: 0.0200

Total cost: $15000.00

Portfolio variance (risk): 0.050000

Optimal stock allocations (to maximize return under risk and budget constraints): Log returns

Walmart: 13.3512 shares

Amazon: 7.8792 shares

Apple: 2.7004 shares

UnitedHealth Group: 1.1567 shares

Berkshire Hathaway: 0.0127 shares

CVS Health: 0.0000 shares

Exxon Mobil: 0.0406 shares

Alphabet: 4.6862 shares

McKesson: 0.4181 shares

Cencora: 0.9912 shares

Expected portfolio return: 0.0200

```
Total cost: $15000.00
Portfolio variance (risk): 0.080000
```

Looking at the outputs from the nonlinear program for maximum acceptable risk and minimal acceptable return, with a budget. We can see that the portfolio risk for the daily simple log returns is 0.08 instead of the 0.05 which we have used for all the previous models. We had to increase the maximum acceptable risk because the program was unable to satisfy the constraint at that lower level. We could also decrease the minimum acceptable return or increase your budget or some combination of them. Until your program can find an optimal solution or can satisfy the constraints.

You should also note that the variance-covariance matrix is based on the simple daily returns of each of your stocks, or on the simple daily log returns. If using optimal stock allocations using simple return, the simple returns variance-covariance matrix should be used. Likewise, if log returns are used for the optimal stock allocations, then the log returns variance-covariance matrix should be used.

The code for R Studio and the Python code for optimization can be found at the end of this paper. Page 9 displays tables that show the data calculated in R, the variance-covariance matrix, and the covariance matrix. The covariance matrix plot can be found on pages 10-11.

In conclusion, we were able to find an optimal solution to our nonlinear program, maximizing our returns with an acceptable risk using both simple returns and log returns. We also made a nonlinear program using a maximal acceptable risk and a minimal acceptable return with a budget, using both the simple return and log returns, and identified that our log return needed to have one of the constraints changed to find an optimal solution. Using non-linear programming to optimize this portfolio of stocks was very interesting. Another interesting application to look into with nonlinear programming would be transportation problems to find optimal solutions to transport networks when accounting for transport costs, supply chains, and capacity constraints. The applications of nonlinear programming are vast, and there are many interesting applications to explore using nonlinear programming in the future.

Table 1

Rank	Company	Open stock price at date x	Variance of stock	Mean daily simple return of stock	Mean daily log return of stock	Mean annual simple return of stock	Mean annual log return of stock
1	**Walmart**	85.43	281.1504	0.000545962	0.000454575	0.137582525	0.114552976
2	**Amazon**	200.89	3072.678	0.001179325	0.000946677	0.2971899	0.238562554
3	**Apple**	193.89	4571.464	0.000901748	0.000727155	0.227240597	0.183242934
4	**UnitedHealth Group**	540.64	22956.84	0.000742058	0.000611682	0.186998566	0.154143788
5	**Berkshire Hathaway**	776929.5	20986468046	0.000578921	0.000509804	0.145888142	0.128470633
6	**CVS Health**	65.62	241.5582	-1.95E-05	-0.000175704	-0.004904527	-0.044277509
7	**Exxon Mobil**	109.865	447.7973	0.000264832	0.000101287	0.066737639	0.025524299
8	**Alphabet**	149.9	2142.18	0.000849589	0.00068706	0.214096302	0.173139196
9	**McKesson**	712.59	21518.43	0.00061451	0.000463489	0.15485652	0.116799278
10	**Cencora**	291.3	2730.95	0.000517473	0.000379946	0.13040312	0.095746392

Table 2

Variance-Covariance Matrix Simple Returns									
Walmart	**Amazon**	**Apple**	**UnitedHealth Group**	**Berkshire Hathaway**	**CVS Health**	**Exxon Mobil**	**Alphabet**	**McKesson**	**Cencora**
0.0001828	-5.6E-06	1.331E-06	6.12179E-06	9.40484E-06	1.441E-05	1.4409E-05	1.4E-06	3.82E-06	-3.6E-06
-5.62E-06	0.000465	6.849E-06	1.44497E-05	1.21727E-05	7.806E-06	7.806E-06	-5.6E-06	1.78E-05	8.47E-06
1.331E-06	6.85E-06	0.0003482	9.64531E-05	0.000100915	8.302E-05	8.3021E-05	0.000198	7.51E-05	7.9E-05
6.122E-06	1.44E-05	9.645E-05	0.000259667	7.64118E-05	8.075E-05	8.0754E-05	8.35E-05	0.000107	0.000103
9.405E-06	1.22E-05	0.0001009	7.64118E-05	0.000137851	0.0001108	0.00011078	9.09E-05	6.99E-05	7.08E-05
1.441E-05	7.81E-06	8.302E-05	8.07541E-05	0.000110785	0.0003261	0.00032606	7.46E-05	8.36E-05	7.9E-05
1.441E-05	7.81E-06	8.302E-05	8.07541E-05	0.000110785	0.0003261	0.00032606	7.46E-05	8.36E-05	7.9E-05
1.399E-06	-5.6E-06	0.0001975	8.35019E-05	9.08533E-05	7.457E-05	7.4572E-05	0.000326	6.13E-05	6.36E-05
3.819E-06	1.78E-05	7.512E-05	0.000107096	6.98675E-05	8.363E-05	8.3632E-05	6.13E-05	0.0003	0.000194
-3.57E-06	8.47E-06	7.904E-05	0.000102673	7.0804E-05	7.903E-05	7.9033E-05	6.36E-05	0.000194	0.000274

Table 3

Variance-Covariance Matrix Log Returns									
Walmart	**Amazon**	**Apple**	**UnitedHealth Group**	**Berkshire Hathaway**	**CVS Health**	**Exxon Mobil**	**Alphabet**	**McKesson**	**Cencora**
0.000183	-5.6E-06	1.33E-06	6.12179E-06	9.40484E-06	1.4409E-05	1.4409E-05	1.4E-06	3.82E-06	-3.6E-06
-5.6E-06	0.000465	6.85E-06	1.44497E-05	1.21727E-05	7.806E-06	7.806E-06	-5.6E-06	1.78E-05	8.47E-06
1.33E-06	6.85E-06	0.000348	9.64531E-05	0.000100915	8.3021E-05	8.3021E-05	0.000198	7.51E-05	7.9E-05
6.12E-06	1.44E-05	9.65E-05	0.000259667	7.64118E-05	8.0754E-05	8.0754E-05	8.35E-05	0.000107	0.000103
9.4E-06	1.22E-05	0.000101	7.64118E-05	0.000137851	0.00011078	0.00011078	9.09E-05	6.99E-05	7.08E-05
1.44E-05	7.81E-06	8.3E-05	8.07541E-05	0.000110785	0.00032606	0.00032606	7.46E-05	8.36E-05	7.9E-05
1.44E-05	7.81E-06	8.3E-05	8.07541E-05	0.000110785	0.00032606	0.00032606	7.46E-05	8.36E-05	7.9E-05
1.4E-06	-5.6E-06	0.000198	8.35019E-05	9.08533E-05	7.4572E-05	7.4572E-05	0.000326	6.13E-05	6.36E-05
3.82E-06	1.78E-05	7.51E-05	0.000107096	6.98675E-05	8.3632E-05	8.3632E-05	6.13E-05	0.0003	0.000194
-3.6E-06	8.47E-06	7.9E-05	0.000102673	7.0804E-05	7.9033E-05	7.9033E-05	6.36E-05	0.000194	0.000274

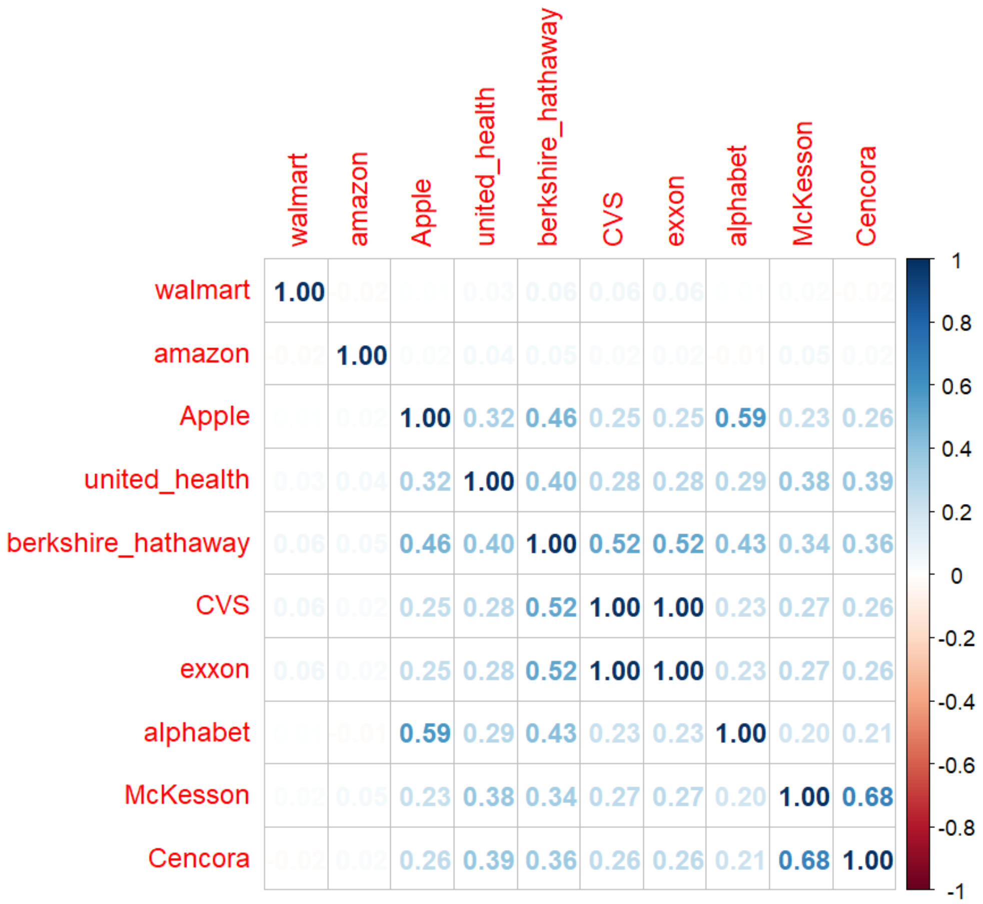

Figure 0.0.1. Covariance Matrix Plot Simple Return

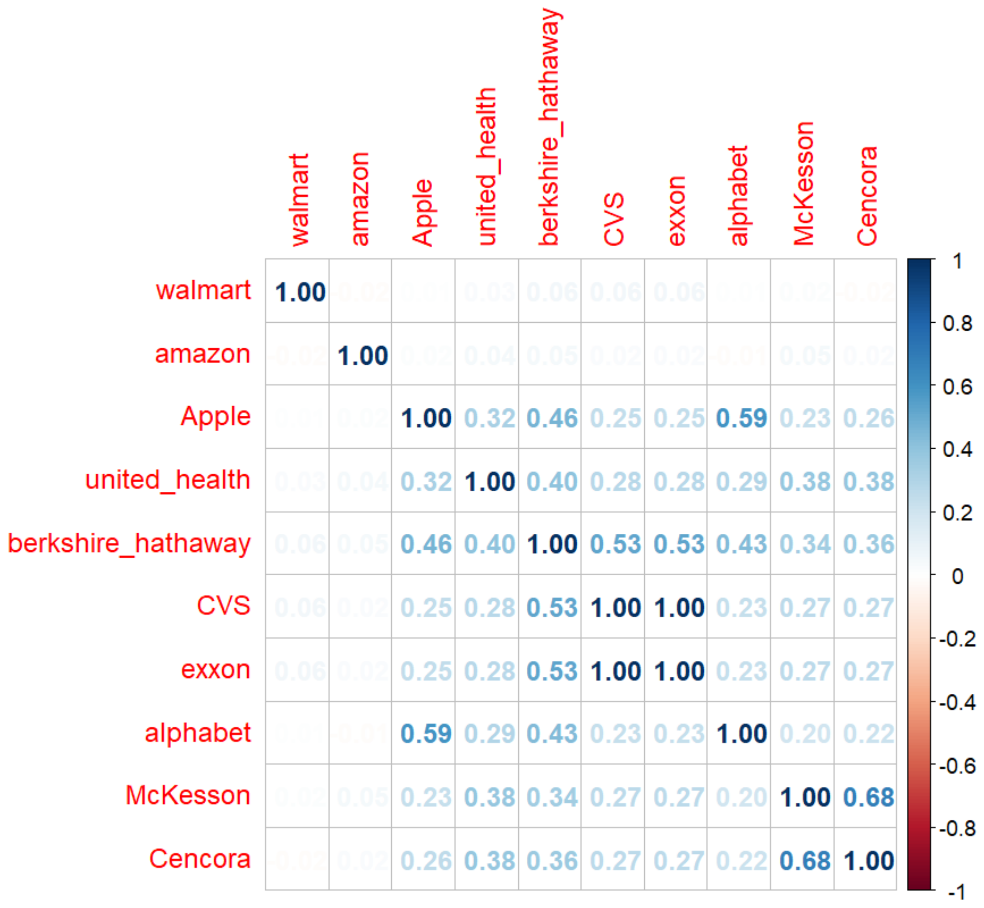

FIGURE 0.0.2. Covariance Matrix Plot Log Return

Here is the code for the data analysis in R

```
install_github("rbrown53/math3150package")

library(math3150package)

install.packages("GGally")

library(GGally)

library(dplyr)

install.packages("corrplot")

library(corrplot)

install.packages("writexl")  # Skip this line if already installed

library(writexl)

#--------------------------------------------------------------

#Walmart Inc. Common Stock

walmart <-data.frame(read.csv

("C:/Users/david/Downloads/HistoricalData_1743387433478.csv"))

head(walmart)


walmart <- walmart |>

  mutate(

    Open = as.numeric(gsub("\\$", "", Open)),  # Ensure numeric

    simple_return = (Open - lead(Open)) / lead(Open),

    log_return = log(Open) - log(lead(Open))

  )


open_data1 <- select(walmart, c('Open', 'simple_return', 'log_return'))
```

```
head(open_data1)


#mean(walmart$Open)

var(walmart$Open)

mean(walmart$simple_return, na.rm = TRUE)

mean(walmart$log_return, na.rm = TRUE)




#---------------------------------------------------------------

#Amazon.com, Inc. Common Stock


amazon <- data.frame(read.csv

("C:/Users/david/Downloads/HistoricalData_1743207033348.csv"))

head(amazon)



amazon <- amazon |>

  mutate(

    Open = as.numeric(gsub("\\$", "", Open)),  # Ensure numeric

    simple_return = (Open - lead(Open)) / lead(Open),

    log_return = log(Open) - log(lead(Open))

  )


open_data2 <- select(amazon, c('Open', 'simple_return', 'log_return'))
```

```
head(open_data2)


#mean(amazon$Open)

var(amazon$Open)

mean(amazon$simple_return, na.rm = TRUE)

mean(amazon$log_return, na.rm = TRUE)


#---------------------------------------------------------------

# Apple Inc. Common Stock


Apple <- data.frame(read.csv

("C:/Users/david/Downloads/HistoricalData_1743895524032.csv"))

head(Apple)



Apple <- Apple |>

  mutate(

    Open = as.numeric(gsub("\\$", "", Open)),  # Ensure numeric

    simple_return = (Open - lead(Open)) / lead(Open),

    log_return = log(Open) - log(lead(Open))

  )


open_data3 <- select(Apple, c('Open', 'simple_return', 'log_return'))
```

```
head(open_data3)


#mean(Apple$Open)

var(Apple$Open)

mean(Apple$simple_return, na.rm = TRUE)

mean(Apple$log_return, na.rm = TRUE)


#---------------------------------------------------------------

# UnitedHealth Group Incorporated Common Stock


united_health <- data.frame(read.csv
    ("C:/Users/david/Downloads/HistoricalData_1743896046270.csv"))

head(united_health)



united_health <- united_health |>
  mutate(
    Open = as.numeric(gsub("\\$", "", Open)),  # Ensure numeric
    simple_return = (Open - lead(Open)) / lead(Open),
    log_return = log(Open) - log(lead(Open))
  )


open_data4 <- select(united_health,
    c('Open', 'simple_return', 'log_return'))
```

```
head(open_data4)


#mean(united_health$Open)

var(united_health$Open)

mean(united_health$simple_return, na.rm = TRUE)

mean(united_health$log_return, na.rm = TRUE)


#---------------------------------------------------------------

# Berkshire Hathaway Inc.


berkshire_hathaway <- data.frame(read.csv

    ("C:/Users/david/Downloads/HistoricalData_1743896392128.csv"))

head(berkshire_hathaway)



berkshire_hathaway <- berkshire_hathaway |>

  mutate(

    Open = as.numeric(gsub("\\$", "", Open)),  # Ensure numeric

    simple_return = (Open - lead(Open)) / lead(Open),

    log_return = log(Open) - log(lead(Open))

  )


open_data5 <- select(berkshire_hathaway,

    c('Open', 'simple_return', 'log_return'))
```

```
head(open_data5)


#mean(berkshire_hathaway$Open)

var(berkshire_hathaway$Open)

mean(berkshire_hathaway$simple_return, na.rm = TRUE)

mean(berkshire_hathaway$log_return, na.rm = TRUE)


#---------------------------------------------------------------

# CVS Health Corporation Common Stock


CVS <- data.frame(read.csv

    ("C:/Users/david/Downloads/HistoricalData_1743897103318.csv"))

head(CVS)



CVS <- CVS |>

  mutate(

    Open = as.numeric(gsub("\\$", "", Open)),  # Ensure numeric

    simple_return = (Open - lead(Open)) / lead(Open),

    log_return = log(Open) - log(lead(Open))

  )


open_data6 <- select(CVS, c('Open', 'simple_return', 'log_return'))


head(open_data6)
```

```
#mean(CVS$Open)

var(CVS$Open)

mean(CVS$simple_return, na.rm = TRUE)

mean(CVS$log_return, na.rm = TRUE)


#---------------------------------------------------------------

# Exxon Mobil Corporation Common Stock


exxon <- data.frame(read.csv

    ("C:/Users/david/Downloads/HistoricalData_1743897103318.csv"))

head(exxon)



exxon <- exxon |>

  mutate(

    Open = as.numeric(gsub("\\$", "", Open)),  # Ensure numeric

    simple_return = (Open - lead(Open)) / lead(Open),

    log_return = log(Open) - log(lead(Open))

  )


open_data7 <- select(exxon, c('Open', 'simple_return', 'log_return'))


head(open_data7)
```

```
#mean(exxon$Open)

var(exxon$Open)

mean(exxon$simple_return, na.rm = TRUE)

mean(exxon$log_return, na.rm = TRUE)


#---------------------------------------------------------------

# Alphabet Inc. Class C Capital Stock


alphabet <- data.frame(read.csv
    ("C:/Users/david/Downloads/HistoricalData_1743897553507.csv"))

head(alphabet)



alphabet <- alphabet |>
  mutate(
    Open = as.numeric(gsub("\\$", "", Open)),  # Ensure numeric
    simple_return = (Open - lead(Open)) / lead(Open),
    log_return = log(Open) - log(lead(Open))
  )


open_data8 <- select(alphabet, c('Open', 'simple_return', 'log_return'))


head(open_data8)


#mean(alphabet$Open)
```

```
var(alphabet$Open)

mean(alphabet$simple_return, na.rm = TRUE)

mean(alphabet$log_return, na.rm = TRUE)


#---------------------------------------------------------------

# McKesson Corporation Common Stock


McKesson <- data.frame(read.csv

    ("C:/Users/david/Downloads/HistoricalData_1743897883281.csv"))

head(McKesson)




McKesson <- McKesson |>

  mutate(

    Open = as.numeric(gsub("\\$", "", Open)),  # Ensure numeric

    simple_return = (Open - lead(Open)) / lead(Open),

    log_return = log(Open) - log(lead(Open))

  )


open_data9 <- select(McKesson, c('Open', 'simple_return', 'log_return'))


head(open_data9)


#mean(McKesson$Open)

var(McKesson$Open)
```

```
mean(McKesson$simple_return, na.rm = TRUE)

mean(McKesson$log_return, na.rm = TRUE)


#-----------------------------------------------------------

# Cencora, Inc. Common Stock


Cencora <- data.frame(read.csv

    ("C:/Users/david/Downloads/HistoricalData_1743898158562.csv"))

head(Cencora)




Cencora <- Cencora |>

  mutate(

    Open = as.numeric(gsub("\\$", "", Open)),  # Ensure numeric

    simple_return = (Open - lead(Open)) / lead(Open),

    log_return = log(Open) - log(lead(Open))

  )


open_data10 <- select(Cencora, c('Open', 'simple_return', 'log_return'))


head(open_data10)


#mean(Cencora$Open)

var(Cencora$Open)

mean(Cencora$simple_return, na.rm = TRUE)
mean(Cencora$log_return, na.rm = TRUE)


#---------------------------------------------------------------

min_rows <- min(
  nrow(walmart), nrow(amazon), nrow(Apple), nrow(united_health),
  nrow(berkshire_hathaway), nrow(CVS), nrow(exxon),
  nrow(alphabet), nrow(McKesson), nrow(Cencora)
)


stock_data <- data.frame(
  walmart = walmart$simple_return[1:min_rows],
  amazon = amazon$simple_return[1:min_rows],
  Apple = Apple$simple_return[1:min_rows],
  united_health = united_health$simple_return[1:min_rows],
  berkshire_hathaway = berkshire_hathaway$simple_return[1:min_rows],
  CVS = CVS$simple_return[1:min_rows],
  exxon = exxon$simple_return[1:min_rows],
  alphabet = alphabet$simple_return[1:min_rows],
  McKesson = McKesson$simple_return[1:min_rows],
  Cencora = Cencora$simple_return[1:min_rows]
)
stock_data_clean <- na.omit(stock_data)
head(stock_data_clean)
nrow(stock_data_clean)
```

```
var_matrix <- var(stock_data_clean)

var_data <- data.frame(var_matrix)

write_xlsx(list(CorrelationMatrix = var_data),

           path = "C:/Users/david/Downloads/varmatrix.xlsx")


C = cor(stock_data_clean)

corrplot(C, method = 'number') # colorful number


cov_matrix <- cor(stock_data_clean)

cov_data <- data.frame(cov_matrix)


write_xlsx(list(CorrelationMatrix = cov_data),

           path = "C:/Users/david/Downloads/covmatrix.xlsx")





log_stock_data <- data.frame(

  walmart = walmart$log_return[1:min_rows],

  amazon = amazon$log_return[1:min_rows],

  Apple = Apple$log_return[1:min_rows],

  united_health = united_health$log_return[1:min_rows],

  berkshire_hathaway = berkshire_hathaway$log_return[1:min_rows],

  CVS = CVS$log_return[1:min_rows],

  exxon = exxon$log_return[1:min_rows],
  alphabet = alphabet$log_return[1:min_rows],

  McKesson = McKesson$log_return[1:min_rows],

  Cencora = Cencora$log_return[1:min_rows]

)

stock_data_clean_log <- na.omit(log_stock_data)

head(stock_data_clean_log)

nrow(stock_data_clean_log)



var_matrix2 <- var(stock_data_clean_log)

var_data2 <- data.frame(var_matrix2)

write_xlsx(list(CorrelationMatrix = var_data2),

           path = "C:/Users/david/Downloads/varmatrix_log2.xlsx")



C = cor(stock_data_clean_log)

corrplot(C, method = 'number') # colorful number


cov_matrix2 <- cor(stock_data_clean_log)

cov_data2 <- data.frame(cov_matrix2)


write_xlsx(list(CorrelationMatrix = cov_data2),

           path = "C:/Users/david/Downloads/covmatrix_log2.xlsx")
```

This is Python code for the nonlinear programming to find an optimal solution

```
import pandas as pd

import numpy as np

from scipy.optimize import minimize


# === Load your data (assumes you saved it to CSV from Excel or

read directly with openpyxl/xlrd) ===

# Example: stock_data.csv has columns: ['Stock', 'Price',

'Return', 'Variance']

# Covariance matrix is in another file, or computed from

historical data

# Replace this with your actual file

df = pd.read_excel("C:/Users/david/Downloads/Math Mod data.xlsx")

print(df.columns.tolist())

# Extract necessary arrays

prices = df['Open stock price at date x'].values

returns = df['Mean daily simple return of stock'].values

# Assume you computed the covariance matrix in R and saved it to

a .csv or .xlsx

cov_df = pd.read_excel("C:/Users/david/Downloads/varmatrix.xlsx")

cov_matrix = cov_df.values


n = len(prices)


# === Portfolio objective: Minimize risk (variance), or maximize
```

```
return ===

# In this case, maximize return under risk and budget constraint

def objective(x):

    return -np.dot(returns, x)

    # Maximize return (minimize negativereturn)


# === Constraints ===

B = 15000  # Budget in dollars (set your value)

S = 0.05   # Minimum acceptable return (e.g., 5%)


constraints = [

    {'type': 'ineq', 'fun': lambda x: B - np.dot(prices, x)},

    # Total cost  B

    {'type': 'ineq', 'fun': lambda x: np.dot(returns, x) - S},

    # Return  S

]

# === Bounds: no short selling ===

bounds = [(0, None) for _ in range(n)]  # x_i  0


# === Initial guess ===

x0 = np.zeros(n)


# === Solve ===

result = minimize(objective, x0, method='SLSQP',

    bounds=bounds, constraints=constraints)
```

```
# === Output ===
if result.success:
    print("Optimal stock allocations:")
    for i, stock in enumerate(df['Company']):
        print(f"{stock}: {result.x[i]:.4f} shares")
    print(f"Expected portfolio return: {np.dot(returns, result.x):.4f}")
    print(f"Total cost: {np.dot(prices, result.x):.2f}")
else:
    print("Optimization failed:", result.message)


df = pd.read_excel("C:/Users/david/Downloads/Math Mod data.xlsx")
print(df.columns.tolist())
# Extract necessary arrays
prices = df['Open stock price at date x'].values
returns_1 = df['Mean daily log return of stock'].values
# Assume you computed the covariance matrix in R and saved it to
a .csv or .xlsx
cov_df =
    pd.read_excel("C:/Users/david/Downloads/varmatrix_log2.xlsx")
cov_matrix = cov_df.values

n = len(prices)

# Portfolio objective: Minimize risk (variance), or maximizereturn
# In this case, maximize return under risk and budget constraint
```

```
def objective(x):

    return -np.dot(returns_1, x)

    # Maximize return (minimize negative return)


# === Constraints ===

B = 15000  # Budget in dollars (set your value)

S = 0.05   # Minimum acceptable return (e.g., 5%)


constraints = [

    {'type': 'ineq', 'fun': lambda x: B - np.dot(prices, x)},

    # Total cost  B

    {'type': 'ineq', 'fun': lambda x: np.dot(returns_1, x) - S},

    # Return  S

]


# === Bounds: no short selling ===

bounds = [(0, None) for _ in range(n)]  # x_i  0


# === Initial guess ===

x0 = np.zeros(n)


# === Solve ===

result = minimize(objective, x0, method='SLSQP',

    bounds=bounds, constraints=constraints)
```

```
# === Output ===
if result.success:
    print("Optimal stock allocations:")
    for i, stock in enumerate(df['Company']):
        print(f"{stock}: {result.x[i]:.4f} shares")
    print(f"Expected portfolio return: {np.dot(returns_1, result.x):.4f}")
    print(f"Total cost: {np.dot(prices, result.x):.2f}")
else:
    print("Optimization failed:", result.message)


# === Parameters ===
budget = 15000                    # Max total cost in dollars
max_risk = 0.05                    # Max allowed portfolio variance
min_return = 0.02                 # Minimum acceptable return


prices = df['Open stock price at date x'].values
returns = df['Mean daily simple return of stock'].values
cov_df = pd.read_excel("C:/Users/david/Downloads/varmatrix.xlsx")
cov_matrix = cov_df.values


n = len(prices)


# === Objective: maximize return (negated for minimize) ===
def objective(x):
    return -np.dot(returns, x)
```

```
# === Constraints (budget added back in) ===
constraints = [
    {'type': 'ineq', 'fun': lambda x: budget - np.dot(prices, x)},
    # Total cost  budget
    {'type': 'ineq', 'fun': lambda x: max_risk - np.dot(x,
        np.dot(cov_matrix, x))},
    # Risk  max_risk
    {'type': 'ineq', 'fun': lambda x: np.dot(returns, x) - min_return},
    # Return  min_return
]

# === Bounds and smarter initial guess ===
bounds = [(0, None) for _ in range(n)]  # No short selling
x0 = np.array([(budget / n) / prices[i] for i in range(n)])
# Equal budget allocation guess

# === Solve ===
result = minimize(objective, x0, method='SLSQP',
    bounds=bounds, constraints=constraints)

# === Output ===
if result.success:
    print("Optimal stock allocations
    (to maximize return under risk and budget constraints):")
    for i, stock in enumerate(df['Company']):
        print(f"{stock}: {result.x[i]:.4f} shares")
    print(f"Expected portfolio return: {np.dot(returns, result.x):.4f}")
    print(f"Total cost: ${np.dot(prices, result.x):.2f}")
    print(f"Portfolio variance (risk): {np.dot(result.x,
        np.dot(cov_matrix, result.x)):.6f}")
else:
    print("Optimization failed:", result.message)

# === Parameters ===
budget = 15000                    # Max total cost in dollars
max_risk = 0.08                    # Max allowed portfolio variance
min_return = 0.02                 # Minimum acceptable return

prices = df['Open stock price at date x'].values
returns = df['Mean daily log return of stock'].values
cov_df = pd.read_excel("C:/Users/david/Downloads/varmatrix_log2.xlsx")
cov_matrix = cov_df.values

n = len(prices)

# === Objective: maximize return (negated for minimize) ===
def objective(x):
    return -np.dot(returns, x)
```

```
# === Constraints (budget added back in) ===

constraints = [

    {'type': 'ineq', 'fun': lambda x: budget - np.dot(prices, x)},

    # Total cost  budget

    {'type': 'ineq', 'fun': lambda x: max_risk - np.dot(x,

    np.dot(cov_matrix, x))},

    # Risk  max_risk

    {'type': 'ineq', 'fun': lambda x: np.dot(returns, x) - min_return},

    # Return  min_return

]


# === Bounds and smarter initial guess ===

bounds = [(0, None) for _ in range(n)]  # No short selling

x0 = np.array([(budget / n) / prices[i] for i in range(n)])

# Equal budget allocation guess


# === Solve ===

result = minimize(objective, x0, method='SLSQP',

    bounds=bounds, constraints=constraints)


# === Output ===

if result.success:

    print("Optimal stock allocations

    (to maximize return under risk and budget constraints):")

    for i, stock in enumerate(df['Company']):
        print(f"{stock}: {result.x[i]:.4f} shares")
    print(f"Expected portfolio return: {np.dot(returns, result.x):.4f}")
    print(f"Total cost: ${np.dot(prices, result.x):.2f}")
    print(f"Portfolio variance (risk):
        {np.dot(result.x, np.dot(cov_matrix, result.x)):.6f}")
else:
    print("Optimization failed:", result.message)
```

References

Alphabet Inc. Class C Capital Stock (GOOG) Historical Data. (n.d.). Nasdaq. Retrieved April 22, 2025, from
https://www.nasdaq.com/market-activity/stocks/goog/historical

Amazon.com, Inc. Common Stock (AMZN) Historical Data. (n.d.). Nasdaq. Retrieved April 22, 2025, from
https://www.nasdaq.com/market-activity/stocks/amzn/historical

Apple (AAPL) Historical Data. (n.d.). Nasdaq. Retrieved April 22, 2025, from
https://www.nasdaq.com/market-activity/stocks/aapl/historical

Berkshire Hathaway Inc. (BRK/A) Historical Data. (n.d.). Nasdaq. Retrieved April 22, 2025, from
https://www.nasdaq.com/market-activity/stocks/brk-a/historical

Cencora, Inc. Common Stock (COR) Historical Quotes. (2025). Nasdaq. Retrieved 4 22, 2025, from
https://www.nasdaq.com/market-activity/stocks/cor/historical

CVS Health Corporation Common Stock (CVS) Historical Data. (n.d.). Nasdaq. Retrieved April 22, 2025, from
https://www.nasdaq.com/market-activity/stocks/cvs/historical

Exxon Mobil Corporation Common Stock (XOM) Historical Data. (n.d.). Nasdaq. Retrieved April 22, 2025, from https://www.nasdaq.com/market-activity/stocks/xom/historical

Hillier, F. S., Lieberman, G. J., & Veatch, M. H. (2024). Introduction to Operations Research. McGraw Hill LLC.

McKesson Corporation Common Stock (MCK) Historical Quotes. (2025). Nasdaq. Retrieved 4 22, 2025, from https://www.nasdaq.com/market-activity/stocks/mck/historical

UnitedHealth Group Incorporated Common Stock (DE) (UNH) Historical Data. (n.d.). Nasdaq. Retrieved April 22, 2025, from https://www.nasdaq.com/market-activity/stocks/unh/historical

Walmart Inc. Common Stock (WMT) Historical Data. (n.d.). Nasdaq. Retrieved April 22, 2025, from https://www.nasdaq.com/market-activity/stocks/wmt/historical