Midterm : Programming for Financial Data Science ([redacted])

Ross Kearney

19th February 2022

1 Exercise 1 Part 5:

Model A takes historical adjusted close stock prices from 27th January 2020 to 23rd March 2020, right as the COVID-19 pandemic caused the markets to have a severe downturn.

Model B takes historical adjusted close stock prices from 1st January 2015 to 31st December 2019, a significant period in the stock market, ending before COVID-19 had any impact on the markets.

Therefore, using Model A and extrapolating the expected annual return, along with the optimal portfolio based on this period, produces less than optimal results as it is calculated as though the indefinite future of the stock market will have the same (bear market) performance as it did during this 8-week period. The optimal portfolio calculated using Model A is a defensive portfolio, containing 80% gold (GLD) in an attempt to reduce the negative impact felt by the crashing market.

Model B uses a much greater period in the stock market, which is a good representative of the historical and (potential) future performance in the stock market. The optimal portfolio calculated using Model B is more balanced between assets, with 30% allocated to gold, meaning that the portfolio is better exposed to the general long-term increase in the market.

It is safe to assume that Model B is the correct model as it captures a more representative period in time for the general performance of the market, while Model A captures a small time period, which was one of the sharpest short term drops in market history.

2 Exercise 2 Part 3:

The shrinkage method benefits optimal portfolio estimated returns when there are a small number of observations to work off of. Shrinkage can effectively improve the accuracy of the covariance matrix estimation particularly in cases where the number of training samples is smaller than the dimensionality, while this is not the case here, the number of observations is relatively small against the dimensionality of the portfolio covariance matrix. The Shrinkage method works by pulling the most extreme coefficients towards more central values, hence systematically reducing estimation error where it has the most severe impact.

The expected annual return for Model A using the optimal portfolio weighting is - 53.403%. Given Model A's small number of observations, extrapolating the 8-week data estimates a severe portfolio loss. However, when the annual return of Model A's optimal portfolio is calculated using the new annual covariance matrix with the estimated shrinkage, the annual return is calculated as - 48.387%, a $\approx$ 10% improvement in returns.

3 Exercise 2 Part 4

Although the best expected annual return comes from the optimal portfolio of Model B (the 5-year data set), the most accurate portfolio is arguably the Shrinkage adjusted Model B. The Shrinkage adjusted Model B has a conservative expected annual return of 4.639%. In the last iteration of the run programme, the portfolio allocations for the Shrinkage adjusted Model B were:

DJI: 0.6129 % GSPC: 1.5282 % FTSE: 0.2435 % EZU: 17.2322 % GLD: 80.3832 %

A portfolio primarily weighted in gold with a smaller percentage weighted in EZU, a Eurozone ETF. While the DJI, GSPC, and FTSE all have sub 2% weightings.

4 Python Code

```
"""
Created on Mon Jan 31 16:02:24 2022

@author: rosskearney
"""

# Optimal portfolio runs 5,000 iterations
# To increase run time, reduce iterations

#Import the relevant python libraries
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.covariance import LedoitWolf

#Get the stock tickers in the portfolio
assets = ['^DJI','^GSPC','^FTSE','EZU','GLD']

#Assign equal weights to the stocks.
equalWeights = np.array([0.2,0.2,0.2,0.2,0.2])


# ========================EXERCISE 1================================
# ============================PART 1================================
# ================MODEL A Download weekly prices==================

# Start and end dates part A (8 week data)
StartDateA = '2020-01-27'
EndDateA = '2020-03-23'

# Create a dataframe to store the adjusted close price of the stocks
dataA = pd.DataFrame()

# Store the adjusted close price of the sock into the data frame
for stock in assets:
  dataA[stock] = yf.download(stock, start=StartDateA, end=EndDateA,
                             interval='1wk')['Adj Close']


# ========================EXERCISE 1================================
# ============================PART 2================================
# ================MODEL B Download weekly prices==================

# Star and end dates part B
StartDateB = '2015-01-01'
EndDateB = '2019-12-31'

# Create a dataframe to store the adjusted close price of the stocks
dataB = pd.DataFrame()

# Store the adjusted close price of the sock into the df
for stock in assets:
  dataB[stock] = yf.download(stock, start=StartDateB, end=EndDateB,
                             interval='1wk')['Adj Close']


# ========================EXERCISE 1================================
# ============================PART 3================================
# expected annual returns and the expected annual covariance matrix
# ===================== Model A (8 week) ============================

# Calculate the weekly log returns
returnsA = np.log(dataA / dataA.shift(1))
```

```
# Weekly return converted to annual (More accurate)
(((returnsA.mean()+1)**52)-1)

# Mean annual return of equal weight portfolio
np.sum((((returnsA.mean()+1)**52)-1)*equalWeights)

# Expected annual covariance matrix
cov_matrix_annualA = ((returnsA.cov()+1)**52)-1

# Expected return if indices made an equally weighted portfolio
portfolio_annual_returnA = np.sum((((returnsA.mean()+1)**52)-1)*equalWeights)

print('\n\n****************** Exercise 1 Part 3 **********************')

# Print portfolio percentages & covariance matrix
portfolio_ret_percentA = portfolio_annual_returnA*100
print("\nExpected annual return for Model A is:", portfolio_ret_percentA, "%")

print("\nExpected annual covariance matrix for Model A is:\n\n",
      cov_matrix_annualA)


# ======================EXERCISE 1================================
# =========================PART 3================================
# expected annual returns and the expected annual covariance matrix
# ================== Model B (5 year) ============================

# Weekly log returns
returnsB = np.log(dataB / dataB.shift(1))

# Weekly return converted to annual (More accurate)
(((returnsB.mean()+1)**52)-1)

# Mean annual return of equal weight portfolio
np.sum((((returnsB.mean()+1)**52)-1)*equalWeights)

# Expected annual covariance matrix
cov_matrix_annualB = ((returnsB.cov()+1)**52)-1

# Expected return if indices made an equally weighted portfolio
portfolio_annual_returnB = np.sum((((returnsB.mean()+1)**52)-1)*equalWeights)

# Print portfolio percentages & covariance matrix
portfolio_ret_percentB = portfolio_annual_returnB*100
print("\nExpected annual return for Model B is:", portfolio_ret_percentB, "%")

print("\nExpected annual covariance matrix for Model B is:\n")
print(cov_matrix_annualB)


# ======================EXERCISE 1================================
# =========================PART 4================================
# ======= Calculate the optimal portfolio in the two cases =========
# ==================== Model A (8 week) ============================

# Run 5000 iterations of portfolio weights
num_portsA = 5000
all_weightsA = np.zeros((num_portsA,len(dataA.columns)))
ret_arrA = np.zeros(num_portsA)
vol_arrA = np.zeros(num_portsA)
sharpe_arrA = np.zeros(num_portsA)

# For each stock in number of iterations
for ind in range(num_portsA):

    # First portfolio is random weighting, ensure total adds up to 100%
    weightsA = np.array(np.random.random(5))
```

```
    weightsA = weightsA / np.sum(weightsA)

    #save weights
    all_weightsA[ind,:] = weightsA

    # exptected return for portfolio weighting
    ret_arrA[ind] = np.sum((((returnsA.mean()+1)**52)-1)*weightsA)

    # expected volatility for portfolio weighting
    weightsTA = np.array(weightsA).T
    vol_arrA[ind] = np.sqrt(np.dot(weightsTA, np.dot(returnsA.cov() * 52, weightsA)))

    # sharpe ratio
    sharpe_arrA[ind] = ret_arrA[ind] / vol_arrA[ind]

# Locates the max sharpe ratio and its index location
sharpe_arrA.max()
sharpe_arrA.argmax()
all_weightsA[sharpe_arrA.argmax(),:]

# Optimum portfolio percentage weights rounded to 4 significant figs.
DJIa = round(all_weightsA[sharpe_arrA.argmax(),:][0]*100,4)
GSPCa = round(all_weightsA[sharpe_arrA.argmax(),:][1]*100,4)
FTSEa = round(all_weightsA[sharpe_arrA.argmax(),:][2]*100,4)
EZUa = round(all_weightsA[sharpe_arrA.argmax(),:][3]*100,4)
GLDa = round(all_weightsA[sharpe_arrA.argmax(),:][4]*100,4)

print('\n\n****************** Exercise 1 Part 4 **********************')

# Print optimum portfolio percentage weights
print('\nThe optimum portfolio allocation percentages for Model A are:\n\nDJI:',
      DJIa, '%\nGSPC:',GSPCa, '%\nFTSE:',FTSEa,'%\nEZU:',EZUa,'%\nGLD:',GLDa,'%')

newWeightA = [DJIa,GSPCa,FTSEa,EZUa,GLDa]

newReturnsA = np.sum((((returnsA.mean()+1)**52)-1)*newWeightA)

print('\n\nThe expected annual return for Model A using the optimal portfolio is:',
      newReturnsA, '%')

# Plot the efficiency frontier
max_sharpe_returnA = ret_arrA[sharpe_arrA.argmax()]
max_sharpe_volatilityA = vol_arrA[sharpe_arrA.argmax()]

plt.figure(figsize=(10,8))
plt.scatter(vol_arrA, ret_arrA, c = sharpe_arrA,cmap = 'plasma')
plt.colorbar(label = 'Sharpe Ratio')
plt.xlabel('Volatility')
plt.ylabel('Return')
plt.title('Model A efficiency frontier with max Sharpe ratio highlighted')
# Plots the max sharpe ratio on the efficient frontier
plt.scatter(max_sharpe_volatilityA,max_sharpe_returnA, c = 'red', s = 50,
            edgecolors = 'black')


# ========================EXERCISE 1================================
# ============================PART 4================================
# ======= Calculate the optimal portfolio in the two cases =========
# ===================== Model B (5 year) ============================

# Run 5000 iterations
num_portsB = 5000
all_weightsB = np.zeros((num_portsB,len(dataB.columns)))
ret_arrB = np.zeros(num_portsB)
vol_arrB = np.zeros(num_portsB)
sharpe_arrB = np.zeros(num_portsB)

# For each iteration
```

```
for ind in range(num_portsB):

    # Initial random allocation of weights
    weightsB = np.array(np.random.random(5))
    weightsB = weightsB / np.sum(weightsB)

    #save weights
    all_weightsB[ind,:] = weightsB

    # exptected return
    # ret_arr[ind] = np.sum((returns.mean() * weights) * 52)
    ret_arrB[ind] = np.sum((((returnsB.mean()+1)**52)-1)*weightsB)

    # expected volatility
    weightsTB = np.array(weightsB).T
    vol_arrB[ind] = np.sqrt(np.dot(weightsTB, np.dot(returnsB.cov() * 52, weightsB)))

    # sharpe ratio
    sharpe_arrB[ind] = ret_arrB[ind] / vol_arrB[ind]

# Locates the max sharpe ratio and its index location
sharpe_arrB.max()
sharpe_arrB.argmax()
all_weightsB[sharpe_arrB.argmax(),:]

# Optimum portfolio percentage weights
DJIb = round(all_weightsB[sharpe_arrB.argmax(),:][0]*100,4)
GSPCb = round(all_weightsB[sharpe_arrB.argmax(),:][1]*100,4)
FTSEb = round(all_weightsB[sharpe_arrB.argmax(),:][2]*100,4)
EZUb = round(all_weightsB[sharpe_arrB.argmax(),:][3]*100,4)
GLDb = round(all_weightsB[sharpe_arrB.argmax(),:][4]*100,4)

# Print optimum portfolio stock percentages
print('\n\nThe optimum portfolio allocation percentages for Model B are:\n\nDJI:',
      DJIb, '%\nGSPC:',GSPCb, '%\nFTSE:',FTSEb,'%\nEZU:',EZUb,'%\nGLD:',GLDb,'%')

newWeightB = [DJIb,GSPCb,FTSEb,EZUb,GLDb]

newReturnsB = np.sum((((returnsB.mean()+1)**52)-1)*newWeightB)

print('\n\nThe expected annual return for Model A using the optimal portfolio is:',
      newReturnsB, '%')

max_sharpe_returnB = ret_arrB[sharpe_arrB.argmax()]
max_sharpe_volatilityB = vol_arrB[sharpe_arrB.argmax()]

plt.figure(figsize=(10,8))
plt.scatter(vol_arrB, ret_arrB, c = sharpe_arrB,cmap = 'plasma')
plt.colorbar(label = 'Sharpe Ratio')
plt.xlabel('Volatility')
plt.ylabel('Return')
plt.title('Model B efficiency frontier with max Sharpe ratio highlighted')
# Plots the max sharpe ratio on the efficient frontier
plt.scatter(max_sharpe_volatilityB,max_sharpe_returnB, c = 'red', s = 50,
            edgecolors = 'black')


# ========================EXERCISE 1================================
# ============================PART 5================================
# ========== Which of the two models is correct? Why?  =============


# ************** In accompanying word document. ********************


# ========================EXERCISE 2================================
# ============================PART 1================================
# Calculate new annual covariance matrix with the estimated shrinkage
```

```
# ==================== Model A (8 week) ==========================

print('\n\n****************** Exercise 2 Part 1 **********************')

# Assign original covariance matrix
real_covA = cov_matrix_annualA

# Assign mean return of stocks
meanRetA = (((returnsA.mean()+1)**52)-1)

# Create random seed to first iteration is the same anytime script is run
np.random.seed(0)

# Calculate covariance matrix with shrinkage
XA = np.random.multivariate_normal(mean=meanRetA, cov=real_covA, size=50)
covA = LedoitWolf().fit(XA)

# New covariance matrix
CovNewA = covA.covariance_

# Ensure covariance shape is 5x5
CovReshapeA = np.mat(CovNewA).reshape(5, int(np.prod(np.mat(CovNewA).shape) / 5)).T

# Put covariance matrix into a dataframe
CovTableA = pd.DataFrame(CovReshapeA)

# Label columns and rows
CovTableA.columns = ['^DJI', '^GSPC', '^FTSE', 'EZU', 'GLD']
CovTableA.index = ['^DJI', '^GSPC', '^FTSE', 'EZU', 'GLD']

# Print Model A covariance matrix with shrinkage
print("\nThe new annual covariance matrix with the estimated shrinkage for Model A is:\n\n",
      CovTableA)

# ========================EXERCISE 2================================
# ============================PART 1================================
# Calculate new annual covariance matrix with the estimated shrinkage
# ==================== Model B (5 year) ==========================

# Assign original covariance matrix
real_covB = cov_matrix_annualB

# Assign mean return of stocks
meanRetB = (((returnsB.mean()+1)**52)-1)

# Create random seed to first iteration is the same anytime script is run
np.random.seed(0)

# Calculate covariance matrix with shrinkage
XB = np.random.multivariate_normal(mean=meanRetB, cov=real_covB, size=50)
covB = LedoitWolf().fit(XB)

# New covariance matrix
CovNewB = covB.covariance_

# Ensure covariance shape is 5x5
CovReshapeB = np.mat(CovNewB).reshape(5, int(np.prod(np.mat(CovNewB).shape) / 5)).T

# Put covariance matrix into a dataframe
CovTableB = pd.DataFrame(CovReshapeB)

# Label columns and rows
CovTableB.columns = ['^DJI', '^GSPC', '^FTSE', 'EZU', 'GLD']
CovTableB.index = ['^DJI', '^GSPC', '^FTSE', 'EZU', 'GLD']

# Print Model B covariance matrix with shrinkage
print("\n\nThe new annual covariance matrix with the estimated shrinkage for Model B is:\n\n",
      CovTableB)
```

```
print('\n\n****************** Exercise 2 Part 2 **********************')

# ========================EXERCISE 2================================
# ============================PART 2================================
# ======= Calculate the optimal portfolio in the two cases =========
# ===================== Model A (8 week) ===========================

# Run 5000 iterations of portfolio weights
num_portsAS = 5000
all_weightsAS = np.zeros((num_portsAS,len(dataA.columns)))
ret_arrAS = np.zeros(num_portsAS)
vol_arrAS = np.zeros(num_portsAS)
sharpe_arrAS = np.zeros(num_portsAS)

# For each stock in number of iterations
for ind in range(num_portsAS):

    # First portfolio is random weighting, ensure total adds up to 100%
    weightsAS = np.array(np.random.random(5))
    weightsAS = weightsAS / np.sum(weightsAS)

    #save weights
    all_weightsAS[ind,:] = weightsAS

    # exptected return for portfolio weighting
    ret_arrAS[ind] = np.sum((((returnsA.mean()+1)**52)-1)*weightsAS)

    # expected volatility for portfolio weighting
    weightsTAS = np.array(weightsAS).T
    vol_arrAS[ind] = np.sqrt(np.dot(weightsTAS, np.dot(CovTableA, weightsAS)))

    # sharpe ratio
    sharpe_arrAS[ind] = ret_arrAS[ind] / vol_arrAS[ind]

# Locates the max sharpe ratio and its index location
sharpe_arrAS.max()
sharpe_arrAS.argmax()
all_weightsAS[sharpe_arrAS.argmax(),:]

# Optimum portfolio percentage weights rounded to 4 significant figs.
DJIas = round(all_weightsAS[sharpe_arrAS.argmax(),:][0]*100,4)
GSPCas = round(all_weightsAS[sharpe_arrAS.argmax(),:][1]*100,4)
FTSEas = round(all_weightsAS[sharpe_arrAS.argmax(),:][2]*100,4)
EZUas = round(all_weightsAS[sharpe_arrAS.argmax(),:][3]*100,4)
GLDas = round(all_weightsAS[sharpe_arrAS.argmax(),:][4]*100,4)

# Print optimal portfolio weights after shrinkage
print('\nThe optimum portfolio allocation percentages for Model A with covariance matrix shrinkage are:\n\nDJI:'
      , DJIas, '%\nGSPC:',GSPCas, '%\nFTSE:',FTSEas,'%\nEZU:',EZUas,'%\nGLD:',GLDas,'%')


newWeightA_S = [DJIas,GSPCas,FTSEas,EZUas,GLDas]

newReturnsA_S = np.sum((((returnsA.mean()+1)**52)-1)*newWeightA_S)

print('\n\nThe expected annual return for Model A using the optimal portfolio after shrinkage is:',
      newReturnsA_S, '%')


# ========================EXERCISE 2================================
# ============================PART 2================================
# ======= Calculate the optimal portfolio in the two cases =========
# ===================== Model B (5 year) ===========================

# Run 5000 iterations of portfolio weights
num_portsBS = 5000
all_weightsBS = np.zeros((num_portsBS,len(dataA.columns)))
```

```
ret_arrBS = np.zeros(num_portsBS)
vol_arrBS = np.zeros(num_portsBS)
sharpe_arrBS = np.zeros(num_portsBS)

# For each stock in number of iterations
for ind in range(num_portsBS):

    # First portfolio is random weighting, ensure total adds up to 100%
    weightsBS = np.array(np.random.random(5))
    weightsBS = weightsBS / np.sum(weightsBS)

    #save weights
    all_weightsBS[ind,:] = weightsBS

    # exptected return for portfolio weighting
    ret_arrBS[ind] = np.sum((((returnsA.mean()+1)**52)-1)*weightsBS)

    # expected volatility for portfolio weighting
    weightsTBS = np.array(weightsBS).T
    vol_arrBS[ind] = np.sqrt(np.dot(weightsTBS, np.dot(CovTableB, weightsBS)))

    # sharpe ratio
    sharpe_arrBS[ind] = ret_arrBS[ind] / vol_arrBS[ind]

# Locates the max sharpe ratio and its index location
sharpe_arrBS.max()
sharpe_arrBS.argmax()
all_weightsBS[sharpe_arrBS.argmax(),:]

# Optimum portfolio percentage weights rounded to 4 significant figs.
DJIbs = round(all_weightsBS[sharpe_arrBS.argmax(),:][0]*100,4)
GSPCbs = round(all_weightsBS[sharpe_arrBS.argmax(),:][1]*100,4)
FTSEbs = round(all_weightsBS[sharpe_arrBS.argmax(),:][2]*100,4)
EZUbs = round(all_weightsBS[sharpe_arrBS.argmax(),:][3]*100,4)
GLDbs = round(all_weightsBS[sharpe_arrBS.argmax(),:][4]*100,4)

# Print optimum allocation weights after shrinkage
print('\n\nThe optimum portfolio allocation percentages for Model B with covariance matrix shrinkage are:\n\nDJI:'
      , DJIbs, '%\nGSPC:',GSPCbs, '%\nFTSE:',FTSEbs,'%\nEZU:',EZUbs,'%\nGLD:',GLDbs,'%')

newWeightB_S = [DJIbs,GSPCbs,FTSEbs,EZUbs,GLDbs]

newReturnsB_S = np.sum((((returnsB.mean()+1)**52)-1)*newWeightB_S)

print('\n\nThe expected annual return for Model A using the optimal portfolio after shrinkage is:',
      newReturnsB_S, '%')
```
