MIE 1622H: Assignment 2 – Risk-Based and Robust Portfolio Selection Strategies

Dr. Oleksandr Romanko

February 15, 2021

Due: Monday, March 8, 2021, not later than 6:00 p.m.

Use Python for all MIE 1622H assignments.

You should hand in:

- Your report.
- Compress all of your Python code files and output files into a file **StudentID.zip** (which can be uncompressed by 7-zip (`http://www.7-zip.org`) software under Windows), and submit it via Quercus portal no later than 6:00 p.m. on March 8.

Where to hand in: Online via Quercus portal, both your code and report.

Introduction

The purpose of this assignment is to compare computational investment strategies based on selecting portfolio with equal risk contributions and using robust mean-variance optimization. In Assignment 1, you have already implemented and compared computational investment strategies based on minimizing portfolio variance, maximizing Sharpe ratio as well as "equally weighted" and "buy and hold" portfolio strategies.

You are proposed to test seven strategies (use your implementation of strategies 1-4 from Assignment 1):

1. "Buy and hold" strategy;
2. "Equally weighted" (also known as "$1/n$") portfolio strategy;
3. "Minimum variance" portfolio strategy;
4. "Maximum Sharpe ratio" portfolio strategy;
5. "Equal risk contributions" portfolio strategy;
6. "Leveraged equal risk contributions" portfolio strategy;
7. "Robust mean-variance optimization" portfolio strategy.

If a portfolio strategy relies on some targets, e.g., target return estimation error, you need to select those targets consistently for each holding period. It is up to you to decide how to select those targets.

Feel free to use Python prototypes provided in class.

Questions

1. **(60 %) Implement investment strategies in Python:**

   You need to test three portfolio re-balancing strategies:

   1. "Equal risk contributions" portfolio strategy: compute a portfolio that has equal risk contributions to standard deviation for each period and re-balance accordingly. You can use IPOPT example from the lecture notes, but you need to compute the gradient of the objective function yourself (to validate your gradient computations you may use finite differences method). The strategy should be implemented in the function `strat_equal_risk_contr`.
   2. "Leveraged equal risk contributions" portfolio strategy: take long 200% position in equal risk contributions portfolio and short risk-free asset for each period and re-balance accordingly, implement it in the function `strat_lever_equal_risk_contr`. You do not have to include risk-free asset as an additional asset when calculating optimal positions. Just make sure that you subtract amount borrowed when calculating portfolio value in a similar way as you do for transaction cost at each time period.
   3. "Robust mean-variance optimization" portfolio strategy: compute a robust mean-variance portfolio for each period and re-balance accordingly. You can use CPLEX example from the lecture notes, but you need to select target risk estimation error and target return according to your preferences as an investor (provide justification of your choices). The strategy should be implemented in the function `strat_robust_optim`.

   Design and implement a rounding procedure, so that you always trade (buy or sell) an integer number of shares.

   Design and implement a validation procedure in your code to test that each of your strategies is feasible (you have enough budget to re-balance portfolio, you correctly compute transaction costs, funds in your cash account are non-negative).

   There is a file **portf_optim2.py** on the course web-page. You are required to complete the code in the file.

   Your Python code should use only CPLEX and IPOPT optimization solvers.

2. **(15 %) Analyze your results:**

   - Produce the following output for the 12 periods (years 2019 and 2020):

```
Period 1: start date 01/02/2019, end date 02/28/2019
  Strategy "Buy and Hold", value begin = $ 1000070.06, value end = $ 1121179.83
  Strategy "Equally Weighted Portfolio", value begin = ... , value end = ...
  Strategy "Minimum Variance Portfolio", value begin = ... , value end = ...
  Strategy "Maximum Sharpe Ratio Portfolio", value begin = ... , value end = ...
  Strategy "Equal Risk Contributions Portfolio", value begin = ... , value end = ...
  Strategy "Leveraged Equal Risk Contributions Portfolio", value begin = ... , value end = ...
  Strategy "Robust Optimization Portfolio", value begin = ... , value end = ...

  ...

Period 12: start date 11/02/2020, end date 12/31/2020
  Strategy "Buy and Hold", value begin = $ 811070.20, value end = $ 972162.37
  Strategy "Equally Weighted Portfolio", value begin = ... , value end = ...
```

```
Strategy "Minimum Variance Portfolio", value begin = ..., value end = ...
Strategy "Maximum Sharpe Ratio Portfolio", value begin = ..., value end = ...
Strategy "Equal Risk Contributions Portfolio", value begin = ..., value end = ...
Strategy "Leveraged Equal Risk Contributions Portfolio", value begin = ... , value end = ...
Strategy "Robust Optimization Portfolio", value begin = ..., value end = ...
```

- Plot one chart in Python that illustrates the daily value of your portfolio (for each of the seven trading strategies) over the years 2019 and 2020 using daily prices provided. Include the chart in your report.
- Plot one chart in Python that illustrates maximum drawdown of your portfolio (for each of the seven trading strategies) for each of the 12 periods (years 2019 and 2020) using daily prices provided. Include the chart in your report.
- Plot one chart in Python to show dynamic changes in portfolio allocations under strategy 7. In each chart, x-axis represents the rolling up time horizon, y-axis denotes portfolio weights between 0 and 1, and distinct lines display the position of selected assets over time periods. Does your robust portfolio selection strategy reduce trading as compared with strategies 3 and 4 that you have implemented in Assignment 1?
- Compare your "equal risk contributions", "leveraged equal risk contributions" and "robust mean-variance optimization" trading strategies between each other and to four strategies implemented in Assignment 1 and discuss their performance relative to each other. Which strategy would you select for managing your own portfolio and why?

3. **(25 %) Test your trading strategies for years 2008 and 2009:**

- Download daily closing prices for the same twenty stocks for years 2008 and 2009 (file **MIE1622H_Assign2_data_2008-2009.zip**). If necessary, transform data into a format that your code can read.
- Test seven strategies that you have implemented for the 12 periods (years 2008 and 2009). Use the same initial portfolio that you are given in Assignment 1. Produce the same output for your strategies as in Question 2.
- Plot one chart in Python that illustrates the daily value of your portfolio (for each of the seven trading strategies) over the years 2008 and 2009 using daily prices that you have downloaded. Include the chart in your report.
- Plot one chart in Python that illustrates maximum drawdown of your portfolio (for each of the seven trading strategies) for each of the 12 periods (years 2008 and 2009) using daily prices provided. Include the chart in your report.
- Plot three charts in Python for strategies 3, 4 and 7 to show dynamic changes in portfolio allocations using the new data set. Does your robust portfolio selection strategy reduce trading as compared with the strategies 3 and 4?
- Compare and discuss relative performance of your seven trading strategies during 2019-2020 and 2008-2009 time periods. Which strategy would you select for managing your own portfolio during 2008-2009 time period and why?

Python File to be Completed

```
# Import libraries
import pandas as pd
import numpy as np
import math

# Complete the following functions
def strat_buy_and_hold(x_init, cash_init, mu, Q, cur_prices):
   x_optimal = x_init
   cash_optimal = cash_init
   return x_optimal, cash_optimal

def strat_equally_weighted(x_init, cash_init, mu, Q, cur_prices):

    return x_optimal, cash_optimal

def strat_min_variance(x_init, cash_init, mu, Q, cur_prices):

   return x_optimal, cash_optimal

def strat_max_Sharpe(x_init, cash_init, mu, Q, cur_prices):

   return x_optimal, cash_optimal

def strat_equal_risk_contr(x_init, cash_init, mu, Q, cur_prices):

   return x_optimal, cash_optimal

def strat_lever_equal_risk_contr(x_init, cash_init, mu, Q, cur_prices):

   return x_optimal, cash_optimal

def strat_robust_optim(x_init, cash_init, mu, Q, cur_prices):

   return x_optimal, cash_optimal

# Input file
input_file_prices = 'Daily_closing_prices.csv'

# Read data into a dataframe
df = pd.read_csv(input_file_prices)

# Convert dates into array [year month day]
def convert_date_to_array(datestr):
    temp = [int(x) for x in datestr.split('/')]
    return [temp[-1], temp[0], temp[1]]

dates_array = np.array(list(df['Date'].apply(convert_date_to_array)))
data_prices = df.iloc[:, 1:].to_numpy()
dates = np.array(df['Date'])
# Find the number of trading days in Nov-Dec 2018 and
# compute expected return and covariance matrix for period 1
day_ind_start0 = 0
day_ind_end0 = len(np.where(dates_array[:,0]==2018)[0])
cur_returns0 = data_prices[day_ind_start0+1:day_ind_end0,:] / data_prices[day_ind_start0:day_ind_end0-1,:] - 1
mu = np.mean(cur_returns0, axis = 0)
Q = np.cov(cur_returns0.T)

# Remove datapoints for year 2018
data_prices = data_prices[day_ind_end0:,:]
dates_array = dates_array[day_ind_end0:,:]
dates = dates[day_ind_end0:]

# Initial positions in the portfolio
init_positions = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 980, 0, 0, 0, 0, 0, 0, 0, 0, 0, 20000])
```

```
# Initial value of the portfolio
init_value = np.dot(data_prices[0,:], init_positions)
print('\nInitial portfolio value = $ {}\n'.format(round(init_value, 2)))

# Initial portfolio weights
w_init = (data_prices[0,:] * init_positions) / init_value

# Number of periods, assets, trading days
N_periods = 6*len(np.unique(dates_array[:,0])) # 6 periods per year
N = len(df.columns)-1
N_days = len(dates)

# Annual risk-free rate for years 2019-2020 is 2.5%
r_rf = 0.025
# Annual risk-free rate for years 2008-2009 is 4.5%
r_rf2008_2009 = 0.045

# Number of strategies
strategy_functions = ['strat_buy_and_hold', 'strat_equally_weighted', 'strat_min_variance', 'strat_max_Sharpe',
                      'strat_equal_risk_contr', 'strat_lever_equal_risk_contr', 'strat_robust_optim']
strategy_names     = ['Buy and Hold', 'Equally Weighted Portfolio', 'Mininum Variance Portfolio',
                      'Maximum Sharpe Ratio Portfolio', 'Equal Risk Contributions Portfolio',
                      'Leveraged Equal Risk Contributions Portfolio', 'Robust Optimization Portfolio']
N_strat = 1  # comment this in your code
#N_strat = len(strategy_functions)  # uncomment this in your code
fh_array = [strat_buy_and_hold, strat_equally_weighted, strat_min_variance, strat_max_Sharpe,
            strat_equal_risk_contr, strat_lever_equal_risk_contr, strat_robust_optim]

portf_value = [0] * N_strat
x = np.zeros((N_strat, N_periods),  dtype=np.ndarray)
cash = np.zeros((N_strat, N_periods),  dtype=np.ndarray)
for period in range(1, N_periods+1):
   # Compute current year and month, first and last day of the period
   if dates_array[0, 0] == 19:
       cur_year  = 19 + math.floor(period/7)
   else:
       cur_year  = 2019 + math.floor(period/7)

   cur_month = 2*((period-1)%6) + 1
   day_ind_start =
       min([i for i, val in enumerate((dates_array[:,0] == cur_year) & (dates_array[:,1] == cur_month)) if val])
   day_ind_end =
       max([i for i, val in enumerate((dates_array[:,0] == cur_year) & (dates_array[:,1] == cur_month+1)) if val])
   print('\nPeriod {0}: start date {1}, end date {2}'.format(period, dates[day_ind_start], dates[day_ind_end]))

   # Prices for the current day
   cur_prices = data_prices[day_ind_start,:]

   # Execute portfolio selection strategies
   for strategy  in range(N_strat):

      # Get current portfolio positions
      if period == 1:
         curr_positions = init_positions
         curr_cash = 0
         portf_value[strategy] = np.zeros((N_days, 1))
      else:
         curr_positions = x[strategy, period-2]
         curr_cash = cash[strategy, period-2]

      # Compute strategy
      x[strategy, period-1], cash[strategy, period-1] = fh_array[strategy](curr_positions, curr_cash, mu, Q, cur_prices)

      # Verify that strategy is feasible (you have enough budget to re-balance portfolio)
      # Check that cash account is >= 0
      # Check that we can buy new portfolio subject to transaction costs

      ###################### Insert your code here ############################
```

```
        # Compute portfolio value
        p_values = np.dot(data_prices[day_ind_start:day_ind_end+1,:], x[strategy, period-1]) + cash[strategy, period-1]
        portf_value[strategy][day_ind_start:day_ind_end+1] = np.reshape(p_values, (p_values.size,1))
        print('  Strategy "{0}", value begin = $ {1:.2f}, value end = $ {2:.2f}'.format( strategy_names[strategy],
               portf_value[strategy][day_ind_start][0], portf_value[strategy][day_ind_end][0]))


        # Compute expected returns and covariances for the next period
        cur_returns = data_prices[day_ind_start+1:day_ind_end+1,:] / data_prices[day_ind_start:day_ind_end,:] - 1
        mu = np.mean(cur_returns, axis = 0)
        Q = np.cov(cur_returns.T)

# Plot results
###################### Insert your code here ############################
```

Sample Python Function for Trading Strategy

```
def strat_buy_and_hold(x_init, cash_init, mu, Q, cur_prices):

    x_optimal = x_init
    cash_optimal = cash_init

    return x_optimal, cash_optimal
```