HW5

October 3, 2023

```
[ ]: import pandas as pd
     import numpy as np
```

1) Use the data to estimate the mean return and covariance matrix.

```
[ ]: data = pd.read_excel('DataforHomework5.xlsx',index_col='Year')
     data.head()
```

```
[ ]:       Stock  Treasury Bond  Money Market     NASDAQ
     Year
     1961  26.81           2.20          2.33  31.664780
     1962  -8.78           5.72          2.93 -15.024354
     1963  22.69           1.79          3.38  20.445586
     1964  16.36           3.71          3.85  23.118500
     1965  12.36           0.93          4.32  17.152602
```

```
[ ]: expected_returns = data.mean()
     expected_returns
```

```
[ ]: Stock           12.044186
     Treasury Bond     7.792326
     Money Market      6.323023
     NASDAQ           12.899098
     dtype: float64
```

```
[ ]: cov_matrix = data.cov()
     cov_matrix
```

```
[ ]:                    Stock  Treasury Bond  Money Market      NASDAQ
     Stock          283.919768      38.850792      2.092916  357.149248
     Treasury Bond   38.850792     114.793828     -2.448836   -6.498260
     Money Market     2.092916      -2.448836     11.814812   -4.392481
     NASDAQ         357.149248      -6.498260     -4.392481  649.448769
```

2) Let risk -free return be 3% solve a nonlinear optimization model to construct a portfolioof these four assets to maximize the Sharpe Ratio.

[ ]:
```python
from scipy.optimize import minimize
# using trick

# minimize y_t @ cov_matrix @ y
def objective(params):
    k = params[0]
    y = params[1:]
    return y @ cov_matrix @ y

# Constraint 1: Sum of y equals k
def constraint1(params):
    k = params[0]
    y = params[1:]
    return np.sum(y) - k

# Constraint 2: Weighted sum of (expected_returns - rf) @ y equals 1
def constraint2(params):
    k = params[0]
    y = params[1:]
    return np.sum((expected_returns - 0.03) * y) - 1

cons = (
    {'type': 'eq', 'fun': constraint1},
    {'type': 'eq', 'fun': constraint2}
)

# Initial guess
initial_y = np.array([0.25, 0.25, 0.25, 0.25])
initial_guess = [1] + list(initial_y)  # 1 for initial k and initial_y values␣
 ↪for y

# Bounds for k and y. k
k_bounds = (0, None)
y_bounds = [(0, None) for _ in initial_y]
all_bounds = [k_bounds] + y_bounds

result = minimize(objective, initial_guess, constraints=cons, bounds=all_bounds)

k_opt = result.x[0]
y_opt = result.x[1:]
opt_x = y_opt/sum(y_opt)

print("Optimal k:", k_opt)
print("Optimal y:", y_opt)
print("Optimal x:", opt_x)
print(f"Optimal Sharpe ratio is {1/np.sqrt(y_opt @ cov_matrix @ y_opt)}")
```

```
Optimal k: 0.14894533186969777
Optimal y: [5.11731844e-17 1.81559434e-02 1.25313935e-01 5.47545351e-03]
Optimal x: [3.43570247e-16 1.21896693e-01 8.41341809e-01 3.67614979e-02]
Optimal Sharpe ratio is 2.1110806469204175
```

3) Construct the portfolio of these four assets to minimize the MAD unter the condition that mean reaturn of the porfolio i sat least 9%

[ ]:
```
returns = data
```

[ ]:
```
# Mad objective functions
def mean_absolute_deviation(x,mean_returns,returns):
    portfolio_return = returns @ x
    expected_return = x @ mean_returns
    return np.mean(np.abs(portfolio_return - expected_return))

def constraint1(x):
    return np.sum(x) - 1

def constraint2(x):
    return x @ expected_returns - 0.09

cons = (
    {'type': 'eq', 'fun': constraint1},
    {'type': 'ineq', 'fun': constraint2}
)

initial_x = np.array([0.25, 0.25, 0.25, 0.25])

# Bounds for x. Each element of x should be between 0 and 1.
x_bounds = [(0, 1) for _ in initial_x]
result = minimize(lambda x: mean_absolute_deviation(x, expected_returns,␣
 ↪returns), initial_x, constraints=cons, bounds=x_bounds)
optimal_x = result.x

print("Optimal x:", optimal_x)
```

```
Optimal x: [0.01966769 0.04385879 0.91676811 0.0197054 ]
```