Final 2023 Spring

Created by Qihang Ma -- 2023.05.01

Name I used to generate data

- python3.11 datageneration.py "Qihang Ma" "..."
- Writing Random Values for Qihang Ma into ...
- Hashed Name: 7b8449b9dcbacd4fe5537e2a5211ca1569bf9f77

If you want to run my code, you may need to install my library with the name "RiskLib".

In [ ]:
```
import warnings
warnings.filterwarnings("ignore")
from RiskLib import calculation, cov_matrix, linear_regression, optimal_port
import pandas as pd
import numpy as np
import datetime as dt
from scipy.optimize import fsolve, minimize
import statsmodels.api as sm
from scipy.stats import t, norm, kurtosis, skew
import matplotlib.pyplot as plt
```

Problem 1

Using the data in “problem1.csv”

- a. Calculate Log Returns (2pts)
- b. Calculate Pairwise Covariance (4pt)
- c. Is this Matrix PSD? If not, fix it with the “near_psd” method (2pt)
- d. Discuss when you might see data like this in the real world. (2pt)

In [ ]:
```
missing_data = pd.read_csv('problem1.csv')
missing_data
```

Out[ ]:

	Price1	Price2	Price3	Date
0	102.826412	94.650195	98.743159	2023-04-12
1	NaN	94.790948	100.022901	2023-04-13
2	102.785907	NaN	NaN	2023-04-14
3	102.847258	96.056428	98.541876	2023-04-15
4	102.818215	94.861366	97.983723	2023-04-16
5	NaN	NaN	98.458978	2023-04-17
6	102.829005	94.108024	98.650071	2023-04-18
7	102.920044	94.206004	NaN	2023-04-19
8	102.848208	94.611114	99.883936	2023-04-20
9	102.642653	96.234730	99.904116	2023-04-21
10	102.754235	95.428819	97.964658	2023-04-22
11	102.868349	93.233250	98.728656	2023-04-23
12	102.917764	94.484647	98.832556	2023-04-24
13	102.893253	92.947829	100.991001	2023-04-25
14	102.836209	95.488845	98.757883	2023-04-26
15	102.784084	94.710784	100.125275	2023-04-27
16	102.900801	95.225558	99.508122	2023-04-28
17	102.876522	96.947049	100.611910	2023-04-29
18	102.824049	92.089138	98.087277	2023-04-30
19	103.022571	95.964771	100.139857	2023-05-01

1.1 Calculate the log return for the price

- If the data is missing, then the return for that day is NaN, and for the next day, it will also be NaN.

In [ ]:

```
missing_returns = calculation.return_calculate(missing_data, method = 'LOG')
missing_returns
```

Out[ ]:

	Price1	Price2	Price3
0	NaN	0.001486	0.012877
1	NaN	NaN	NaN
2	0.000597	NaN	NaN
3	-0.000282	-0.012519	-0.005680
4	NaN	NaN	0.004839
5	NaN	NaN	0.001939
6	0.000885	0.001041	NaN
7	-0.000698	0.004291	NaN
8	-0.002001	0.017015	0.000202
9	0.001087	-0.008410	-0.019604
10	0.001110	-0.023276	0.007768
11	0.000480	0.013333	0.001052
12	-0.000238	-0.016399	0.021604
13	-0.000555	0.026971	-0.022360
14	-0.000507	-0.008182	0.013751
15	0.001135	0.005421	-0.006183
16	-0.000236	0.017917	0.011031
17	-0.000510	-0.051408	-0.025413
18	0.001929	0.041224	0.020710

1.2 Calculate the pairwise Covariance Matrix

- Since there are some missing values in the returns, calculate the covariance matrix with the pairwise method.

In [ ]:

```
pairwise_cov = cov_matrix.missing_cov(missing_returns.values, skipMiss=False
pd.DataFrame(pairwise_cov)
```

Out[ ]:

	0	1	2
0	9.983249e-07	0.000003	0.000003
1	2.985551e-06	0.000501	0.000112
2	2.997105e-06	0.000112	0.000217

1.3 Check if the matrix psd or not

In [ ]:
```
cov_matrix.is_psd(pairwise_cov)
```

Out[ ]: `True`

So, it is a psd matrix.

1.4 When we will see the missing data

Not all markets are open at the same time on the same days. A holiday in one market is not necessarily a holiday in another, even in the same country. Or in different countries, there will be different opening time. So, we may see the missing datas like this.

Problem 2

“problem2.csv” contains data about a call option. Time to maturity is given in days. Assume 255 days in a year.

- a. Calculate the call price (1pt)
- b. Calculate Delta (1pt)
- c. Calculate Gamma (1pt)
- d. Calculate Vega (1pt)
- e. Calculate Rho (1pt)

Assume you are long 1 share of underlying and are short 1 call option. Using Monte Carlo assuming a Normal distribution of arithmetic returns where the implied volatility is the annual volatility and 0 mean

- f. Calculate VaR at 5% (2pt)
- g. Calculate ES at 5% (2pt)
- h. This portfolio’s payoff structure most closely resembles what? (1pt)

In [ ]:
```
call_option = pd.read_csv('problem2.csv')
call_option
```

Out[ ]:

	Underlying	Strike	IV	TTM	RF	DivRate
0	85.084564	74.575976	0.22	148	0.045	0.04477

2.1 Calculate the value and greeks about this option with Black-scholes model

In [ ]:

```
S0 = call_option['Underlying'].values[0]
K = call_option['Strike'].values[0]
iv = call_option['IV'].values[0]
rf = call_option['RF'].values[0]
q = call_option['DivRate'].values[0]
ttm = call_option['TTM'].values[0]/255

call = Option.black_scholes_matrix(S0,K,ttm,rf,q,iv,'call')
pd.DataFrame(call.greeks(), index=['call'])
```

Out[ ]:

	Value	Delta	Gamma	Vega	Theta	Rho	Carry Rho
call	11.844251	0.787432	0.018651	17.240393	-2.749939	32.010991	38.885301

2.2 Simulate the returns and calculate the VaR and ES

In [ ]:
```
ttm_1 = (call_option['TTM'].values[0]-1)/255
VaRs = []
ESs = []
dfs = []

for i in range(1000):

    np.random.seed(i)
    sim_r = np.random.normal(size=10000, loc=0, scale=iv/np.sqrt(255))
    sim_price = pd.DataFrame((1+sim_r) * S0, columns=['Price'])

    sim_price['PnL'] = sim_price.apply(lambda x:call.price() - Option.black_


    VaRs.append(VaR.calculate_var(sim_price['PnL']))
    ESs.append(VaR.calculate_ES(sim_price['PnL']))
    dfs.append(simulation.Fitting_t_MLE(sim_price['PnL'])[0])

VaRs = np.array(VaRs)
ESs = np.array(ESs)
dfs = np.array(dfs)

print("VaR Mean: {:.4f} -- 5% range [{:.4f}, {:.4f}].".format(VaRs.mean(), r
print("ES Mean: {:.4f} -- 5% range [{:.4f}, {:.4f}].\n".format(ESs.mean(), r

print("If fitting with t distribution, the degree of freedom:")
print("df Mean: {:.4f} -- 5% range [{:.4f}, {:.4f}].".format(dfs.mean(), np.
```

```
VaR Mean: 0.4337 -- 5% range [0.4216, 0.4461].
ES Mean: 0.5607 -- 5% range [0.5455, 0.5762].

If fitting with t distribution, the degree of freedom:
df Mean: 52.0227 -- 5% range [25.3523, 121.9174].
```

Since the degree of freedom is relatively high, so this portfolio's payoff structure most closely resembles normal distribution.

Problem 3

Data in “problem3_cov.csv” is the covariance for 3 assets. “problem3_ER.csv” is the expected return for each asset as well as the risk free rate.

- a. Calculate the Maximum Sharpe Ratio Portfolio (4pt)
- b. Calculate the Risk Parity Portfolio (4pt)
- c. Compare the differences between the portfolio and explain why. (2pt)

In [ ]:
```
covar_matrix = pd.read_csv('problem3_cov.csv')
origin_exp_return = pd.read_csv('problem3_ER.csv')

assets = covar_matrix.columns.to_list()

exp_return = origin_exp_return.values[0][1:]
rf = origin_exp_return['RF'].values[0]
```

3.1 Calculate the Maximum Sharpe Ratio Portfolio

- With the constrain of positive weights

In [ ]:
```
weights_sr, _ = optimal_portfolio.Optweight_sr(assets, exp_return, covar_mat
weights_sr
```

Out[ ]:

	Stock	Weight	cEr
0	Asset1	0.398464	0.056812
1	Asset2	0.289620	0.051914
2	Asset3	0.311915	0.052918

3.2 Calculate the Risk Parity Portfolio

In [ ]:
```
weights_rp = risk_parity.vol_risk_parity(exp_return, covar_matrix)
weights_rp
```

Out[ ]:

	Weight	cEr	CSD
0	0.398465	0.056812	0.064306
1	0.289625	0.051915	0.064306
2	0.311910	0.052918	0.064306

3.3 Comparation of two portfolios

Correlations and Sharpe ratios are equal -> risk parity is the maximum sharpe ratio portfolio

Problem 4

Data in “problem4_returns.csv” is a series of returns for 3 assets. “problem4_startWeight.csv” is the starting weights of a portfolio of these assets as of the first day in the return series.

- a. Calculate the new weights for the start of each time period (2pt)
- b. Calculate the ex-post return attribution of the portfolio on each asset (4pt)
- c. Calculate the ex-post risk attribution of the portfolio on each asset (2pt)

In [ ]:
```
returns = pd.read_csv('problem4_returns.csv').drop('Date', axis=1)
start_weights = pd.read_csv('problem4_startWeight.csv').values[0]
```

4.1 Calculate the new weights from the start to each time period

In [ ]:
```
stocks = list(returns.columns)
n = returns.shape[0]

weights = np.empty((n+1, len(start_weights)))
lastW = np.copy(start_weights)
matReturns = returns[stocks].values

for i in range(n):
    # Save Current Weights in Matrix
    weights[i,:] = lastW

    # Update Weights by return
    lastW = lastW * (1.0 + matReturns[i,:])

    # Portfolio return is the sum of the updated weights
    pR = np.sum(lastW)
    # Normalize the wieghts back so sum = 1
    lastW = lastW / pR

weights[n,:] = lastW

pd.DataFrame(weights, columns=stocks)
```

Out[ ]:

	Asset1	Asset2	Asset3
0	0.429088	0.284605	0.286308
1	0.442359	0.271454	0.286187
2	0.457152	0.273731	0.269117
3	0.432297	0.288847	0.278856
4	0.451087	0.264639	0.284274
5	0.471658	0.233571	0.294771
6	0.486315	0.228444	0.285241
7	0.507897	0.239593	0.252510
8	0.534966	0.233213	0.231821
9	0.512858	0.242326	0.244816
10	0.531614	0.232880	0.235506
11	0.535166	0.248667	0.216168
12	0.530176	0.254229	0.215594
13	0.558669	0.240152	0.201179
14	0.530123	0.261294	0.208583
15	0.519007	0.267382	0.213611
16	0.520033	0.263188	0.216779
17	0.518247	0.253837	0.227916
18	0.515476	0.260974	0.223550
19	0.537459	0.246885	0.215656
20	0.586327	0.224125	0.189548

4.2 & 4.3 Calculate the ex-post return & risk attribution

In [ ]:

```
risk_attribution.expost_attribution(start_weights,returns)
```

Out[ ]:

	Value	Asset1	Asset2	Asset3	Portfolio
0	TotalReturn	0.629554	-0.060875	-0.210485	0.192545
1	Return Attribution	0.276461	-0.019145	-0.064771	0.192545
2	Vol Attribution	0.021414	0.008023	0.006717	0.036153

Problem 5

Input prices in “problem5.csv” are for a portfolio. You hold 1 share of each asset. Using arithmetic returns, fit a generalized T distribution to each asset return series. Using a Gaussian Copula:

- a. Calculate VaR (5%) for each asset (3pt)
- b. Calculate VaR (5%) for a portfolio of Asset 1 & 2 and a portfolio of Asset 3 & 4 (4pt)
- c. Calculate VaR (5%) for a portfolio of all 4 assets. (3pt)

In [ ]:

```
dataset = pd.read_csv('problem5.csv')
all_returns = calculation.return_calculate(dataset).drop('Date',axis=1)

latest_prices = dataset.drop('Date',axis=1).tail(1).values[0]
```

In [ ]:
```
def gaussian_copula(returns, fitting_model=None, n_sample=10000, seed=12345)
    stocks = returns.columns.tolist()
    n = len(stocks)

    if fitting_model is None:
        fitting_model = np.full(n, 't')


    # Fitting model for each stock
    parameters = []
    assets_returns_cdf = pd.DataFrame()
    for i, stock in enumerate(stocks):
        if fitting_model[i] == 't':
            params = t.fit(returns[stock])
            fitting = 't'
        elif fitting_model[i] == 'n':
            params = norm.fit(returns[stock])
            fitting = 'n'
        parameters.append(params)
        assets_returns_cdf[stock] = t.cdf(returns[stock],df=params[0], loc=p

    # Simulate N samples with spearman correlation matrix
    np.random.seed(seed)
    spearman_corr_matrix = assets_returns_cdf.corr(method='spearman')
    sim_sample = simulation.multivariate_normal_simulation(spearman_corr_mat
    sim_sample = pd.DataFrame(sim_sample, columns=stocks)

    # Convert simulation result with cdf of standard normal distribution
    sim_sample_cdf = pd.DataFrame()
    for stock in stocks:
        sim_sample_cdf[stock] = norm.cdf(sim_sample[stock],loc=0,scale=1)

    # Convert cdf matrix to return matrix with parameter
    sim_returns = pd.DataFrame()
    for i, stock in enumerate(stocks):
        if fitting_model[i] == 't':
            sim_returns[stock] = t.ppf(sim_sample_cdf[stock], df=parameters[
        elif fitting_model[i] == 'n':
            sim_returns[stock] = norm.ppf(sim_sample_cdf[stock],  loc=parame

    return sim_returns, pd.DataFrame(parameters,index=[stocks,fitting_model]
```

In [ ]:
```
sim_returns, params = gaussian_copula(all_returns, seed=1)
```

5.1 Calculate VaR for each asset

In [ ]:
```
for i, asset in enumerate(sim_returns.columns):
    single_VaR = VaR.calculate_var(sim_returns[asset])
    single_dollar_VaR = VaR.calculate_var(sim_returns[asset] * latest_prices

    print("For Asset {}, the VaR is {:.6f}, $VaR is {:.6f}.".format(i+1,sing
```

```
For Asset 1, the VaR is 0.000814, $VaR is 0.092485.
For Asset 2, the VaR is 0.000547, $VaR is 0.049353.
For Asset 3, the VaR is 0.000498, $VaR is 0.043831.
For Asset 4, the VaR is 0.000720, $VaR is 0.062007.
```

5.2 Calculate VaR for a portfolio of Asset 1 & 2 and a portfolio of Asset 3 & 4

In [ ]:
```
VaR_12 = VaR.calculate_var(sim_returns[['Price1','Price2']].dot(latest_price
print("For Portfolio of Asset 1 & 2, the VaR is {:.6f}, $VaR is {:.6f}.".for
```

```
For Portfolio of Asset 1 & 2, the VaR is 0.000670, $VaR is 0.136627.
```

In [ ]:
```
VaR_34 = VaR.calculate_var(sim_returns[['Price3','Price4']].dot(latest_price
print("For Portfolio of Asset 3 & 4, the VaR is {:.6f}, $VaR is {:.6f}.".for
```

```
For Portfolio of Asset 3 & 4, the VaR is 0.000592, $VaR is 0.103059.
```

5.3 Calculate VaR for the whole portfolio

In [ ]:
```
VaR_all = VaR.calculate_var(sim_returns.dot(latest_prices))
print("For the whole Portfolio, the VaR is {:.6f}, $VaR is {:.6f}.".format(V
```

```
For the whole Portfolio, the VaR is 0.000618, $VaR is 0.233630.
```

In [ ]: