Final_Qijun

May 1, 2023

1 data generation script

1.0.1 python3 datageneration.py "Qijun Yang" "./"

2 Final Exam

3 Q1

```
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)
```

3.0.1 a. Calculate Log Returns (2pts)

[ ]:
```
import pandas as pd
import numpy as np
from RiskPackage.CalculateReturn import return_calculate
# Some data is missing
df=pd.read_csv('problem1.csv',index_col='Date')
# Remove the mean & Log Return(CONTINUOUS)
rt=return_calculate(df,option="CONTINUOUS",rm_means=True)  # Remove the NAN data
rt
```

[ ]:
```
              Price1    Price2    Price3
Date
2023-04-15 -0.001612  0.001592  0.000575
2023-04-16  0.005625  0.026347  0.000830
2023-04-17 -0.007878 -0.018574 -0.000470
2023-04-18 -0.003407 -0.025790 -0.000927
2023-04-19  0.006595  0.017381  0.000220
2023-04-20  0.002060  0.014041  0.000619
2023-04-23  0.003370  0.008536 -0.000213
2023-04-24 -0.003166 -0.026338 -0.000765
2023-04-25 -0.000378  0.011499  0.000684
2023-04-26  0.000251 -0.000263 -0.000189
2023-04-27 -0.002395 -0.008243 -0.000325
```

```
2023-04-28  0.000935 -0.000187 -0.000039
```

[ ]:
```python
# If we keep the missing data
rt_with_NA = np.log(df/df.shift())
rt_with_NA
```

[ ]:
```
              Price1    Price2    Price3
Date
2023-04-12       NaN       NaN       NaN
2023-04-13  0.003281       NaN       NaN
2023-04-14  0.000404       NaN       NaN
2023-04-15 -0.001680  0.001320  0.000609
2023-04-16  0.005556  0.026074  0.000864
2023-04-17 -0.007946 -0.018846 -0.000436
2023-04-18 -0.003475 -0.026063 -0.000893
2023-04-19  0.006527  0.017108  0.000254
2023-04-20  0.001992  0.013768  0.000653
2023-04-21       NaN -0.005138 -0.000099
2023-04-22       NaN  0.005400  0.000159
2023-04-23  0.003302  0.008264 -0.000179
2023-04-24 -0.003235 -0.026610 -0.000731
2023-04-25 -0.000446  0.011227  0.000718
2023-04-26  0.000183 -0.000535 -0.000155
2023-04-27 -0.002463 -0.008515 -0.000291
2023-04-28  0.000867 -0.000459 -0.000005
2023-04-29       NaN       NaN -0.000025
2023-04-30       NaN       NaN       NaN
2023-05-01 -0.002794 -0.006280       NaN
```

3.0.2 b. Calculate Pairwise Covariance (4pt)

[ ]:
```python
# Pandas' cov method can help us do that.
# Here is the documentation:
'''
    DataFrame.cov(min_periods=None, ddof=1, numeric_only=False)[source]
        Compute pairwise covariance of columns, excluding NA/null values.

        Compute the pairwise covariance among the series of a DataFrame. The␣
↪returned data frame is the covariance matrix of the columns of the DataFrame.
'''
cov=df.cov()
cov
```

[ ]:
```
          Price1    Price2    Price3
Price1  0.064025  0.243096  0.004409
Price2  0.243096  1.199251  0.032727
Price3  0.004409  0.032727  0.001196
```

3.0.3 c. Is this Matrix PSD? If not, fix it with the "near_psd" method (2pt)

[ ]:
```python
# Test the matrix is PSD or not
from RiskPackage.NonPsdFixes import PSD
try:
    PSD.confirm(cov.to_numpy())
except:
    print("The matrix is not PSD")
```

```
The matrix is not PSD
```

[ ]:
```python
# Use Rebonato and Jackel's Method (near_psd) to fix it
from RiskPackage.NonPsdFixes import near_psd
# Weighted Matrix -- Set it to be diagonal
n=cov.to_numpy().shape[0]
weight=np.eye(n)
# Fix non-PSD matrix
cov_psd=near_psd(cov.to_numpy(),weight).psd
cov_psd
```

[ ]:
```
array([[6.40249348e-02, 2.42102261e-01, 4.41350426e-03],
       [2.42102261e-01, 1.19925078e+00, 3.25967968e-02],
       [4.41350426e-03, 3.25967968e-02, 1.19599547e-03]])
```

[ ]:
```python
# Test the Fixed matrix is PSD or not
try:
    PSD.confirm(cov_psd)
except:
    print("The matrix is not PSD")
```

```
Matrix is Sysmetric Positive Definite!
```

3.0.4 d. Discuss when you might see data like this in the real world. (2pt)

1. When I want make some transactions about several securities among different conturies. I may encounter missing data like this since not all markets are open at the same time on the same days.
2. When I deal with high frequency data to do high-frequency trading, the missing data is much more common. With time intervals measured in milliseconds, it is impossible to see updated prices on all assets.

4 Q2

```
"problem2.csv" contains data about a call option. Time to maturity is given in days. Assume 25
    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 volati
    f. Calculate VaR at 5% (2pt)
    g. Calculate ES at 5% (2pt)
    h. This portfolio's payoff structure most closely resembles what? (1pt)
```

4.0.1 a. Calculate the call price (1pt)

[ ]:
```python
# Initialization
df=pd.read_csv('problem2.csv')
t=df['TTM'][0]
tradingDayYear=255 # trading days
ttm=t/tradingDayYear # TTM
s=df['Underlying'][0] # Underlying Price
strike=df['Strike'][0] # Strike Price
rf=df['RF'][0] # Risk-Free rate
q=df['DivRate'][0] # Dividend Rate
c=rf-q # cost of carry
vol=df['IV'][0] # Implied Vol

from RiskPackage.Options import gbsm
call_price=gbsm(s,strike,ttm,vol,rf,c,call=True)
print("The Call Option Price is {}".format(call_price))
```

```
The Call Option Price is 3.733860012361646
```

4.0.2 b. Calculate Delta (1pt)

4.0.3 c. Calculate Gamma (1pt)

4.0.4 d. Calculate Vega (1pt)

4.0.5 e. Calculate Rho (1pt)

[ ]:
```python
# I have written a function to calculate all greeks at once
from RiskPackage.Options import greeks_closed_form

# Get all the Greeks
call_greeks_closed_form=greeks_closed_form(s,strike,ttm,vol,rf,c,call=True)
call_greeks_closed_form[['Detla','Gamma','Vega','Rho']]
```

[ ]:

	Detla	Gamma	Vega	Rho
Call	0.432162	0.031107	31.558321	25.364706

4.0.6 f. Calculate VaR at 5% (2pt)

[ ]:
```
from scipy.stats import norm

# Simulated Returns with 0 mean and implied Vol
size=10000
simulatedReturn=norm.rvs(size=size,loc=0,scale=vol)
# Simulated Underlying Price
simPrice=s*(1+simulatedReturn)
# Simulated Portfolio Value Change
SimPortValueChange = (simPrice - s) * 1  + ␣
 ↪(gbsm(simPrice,strike,ttm,vol,rf,c,call=True)- call_price)*(-1)

from RiskPackage.RiskMetrics import RiskMetrics
VaR=RiskMetrics.VaR_historical(SimPortValueChange,alpha=0.05)
print("Var is {}".format(VaR))
```

```
Var is 22.36460957190024
```

4.0.7 g. Calculate ES at 5% (2pt)

[ ]:
```
from RiskPackage.RiskMetrics import RiskMetrics
ES=RiskMetrics.ES_historical(SimPortValueChange,alpha=0.05)
print("ES is {}".format(ES))
```

```
ES is 28.89056862227861
```

4.0.8 h. This portfolio's payoff structure most closely resembles what? (1pt)

Due to the Put Call Parity $C + Xe^{-rt} = P + S$, we have $S - C = Xe^{-rt} - P$, which means that like we are shorting a put option.

5 Q3

```
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)
```

5.0.1 a. Calculate the Maximum Sharpe Ratio Portfolio (4pt)

[ ]:
```
# Covariance Matrix of all assets
cov=pd.read_csv('problem3_cov.csv')
cov.index=cov.columns
# Expected Returns of all assets
ER=pd.read_csv('problem3_ER.csv')
ER_assets=ER.iloc[0,1:]
```

```python
# Risk-free rate
rf=ER['RF'][0]

from RiskPackage.PortfolioOptimization import super_efficient_portfolio
from RiskPackage.PortfolioOptimization import riskBudget
# Calculate the Super Efficient Portfolio/Maximum Sharpe Ratio Portfolio
resSR=super_efficient_portfolio(ER_assets,cov,rf=rf)
SR_Portfolio=pd.DataFrame(index=cov.index)
SR_Portfolio['Weight']=resSR.x
SR_Portfolio['RiskPortion']=riskBudget(SR_Portfolio['Weight'],cov)
print("------ Maximum Sharpe Ratio Portfolio -------")
SR_Portfolio
```

```
------ Maximum Sharpe Ratio Portfolio -------
```

[ ]:

	Weight	RiskPortion
Asset1	0.335766	0.337592
Asset2	0.298656	0.331164
Asset3	0.365578	0.331244

5.0.2 b. Calculate the Risk Parity Portfolio (4pt)

[ ]:
```python
from RiskPackage.PortfolioOptimization import RiskParity
# Calculate the Risk Parity Portfolio
resRP=RiskParity(cov)
RP_Portfolio=pd.DataFrame(index=cov.index)
RP_Portfolio['Weight']=resRP.x
RP_Portfolio['RiskPortion']=riskBudget(RP_Portfolio['Weight'],cov)
print("------ Risk Parity Portfolio -------")
RP_Portfolio
```

```
------ Risk Parity Portfolio -------
```

[ ]:

	Weight	RiskPortion
Asset1	0.331671	0.333336
Asset2	0.300538	0.333334
Asset3	0.367791	0.333330

5.0.3 c. Compare the differences between the portfolio and explain why. (2pt)

As we can see from above, the risk(volatility) propotion of two portfolios are different. In Maximum Sharpe Ratio Portfolio, Asset1 has the most risk exposure. However, all the risk are equally distributed in Risk Parity Portfolio. Three assets has the same risk exposure. The reason why this happens is that Maximum Sharpe Ratio Portfolio is constructed by optimaizing the portfolio Sharpe ratio, but the Risk Parity Portfolio is built by making sure each assets contribute same volatility to the total portfolio.

6 Q4

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 fi
    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)

6.0.1 a. Calculate the new weights for the start of each time period (2pt)

[ ]:
```
# Get returns and updated weights for each day
rts=pd.read_csv('problem4_returns.csv',index_col='Date')
rts.index=pd.to_datetime(rts.index)
# initial weights
last_weight=pd.read_csv('problem4_startWeight.csv').loc[0,:]
last_weight.index=rts.columns

# Update the Weights
weights=[]
portfolio_rts=[]
for i in range(rts.shape[0]):
    # Store the weights
    weights.append(last_weight)

    # Update Weights by return
    last_weight=last_weight*(rts.iloc[i,:]+1)
    # Calculate the portforlio return
    p_rt=last_weight.sum()
    # Normalize the wieghts back so sum = 1
    last_weight=last_weight/p_rt

    # Store the portforlio return
    portfolio_rts.append(p_rt-1)
weights=pd.DataFrame(weights,index=rts.index)
weights.index=pd.to_datetime(weights.index)
print("--------------- New Weights ----------------")
weights
```

```
--------------- New Weights ----------------
```

[ ]:

Date	Asset1	Asset2	Asset3
2023-04-12	0.315777	0.262697	0.421526
2023-04-13	0.328093	0.261497	0.410410
2023-04-14	0.286589	0.289380	0.424031
2023-04-15	0.283248	0.284838	0.431914
2023-04-16	0.269069	0.297431	0.433500
2023-04-17	0.274706	0.274285	0.451009

2023-04-18	0.273578	0.265141	0.461281
2023-04-19	0.239212	0.277775	0.483013
2023-04-20	0.247415	0.265680	0.486904
2023-04-21	0.228139	0.274530	0.497332
2023-04-22	0.229906	0.284942	0.485152
2023-04-23	0.247224	0.296308	0.456468
2023-04-24	0.261224	0.327946	0.410829
2023-04-25	0.245000	0.340344	0.414656
2023-04-26	0.243514	0.330274	0.426212
2023-04-27	0.246413	0.304130	0.449456
2023-04-28	0.245670	0.289219	0.465111
2023-04-29	0.245182	0.282629	0.472189
2023-04-30	0.237708	0.287623	0.474669
2023-05-01	0.252431	0.294725	0.452844

6.0.2 b. Calculate the ex-post return attribution of the portfolio on each asset (4pt)

[ ]:

```python
# Add the Portfolio Return to the dataframe rts
rts['PortfolioReturn']=pd.DataFrame(portfolio_rts,index=rts.index)
# Calculate the total return
total_rt=(rts['PortfolioReturn']+1).prod()-1

# Calculate the Carino K
k=np.log(total_rt+1)/total_rt
# Carino k_t is the ratio scaled by 1/K
carinoK = np.log(1+rts['PortfolioReturn'])/k/rts['PortfolioReturn']
# Transform carinoK to dataframe to be multiplied by weights and rts
carinoK_df=pd.DataFrame([carinoK]*weights.shape[1],index=weights.columns).T
# Calculate the return attribution (has been adjusted by carinoK_df)
return_attribution=(weights * rts * carinoK_df).dropna(axis=1)

# Calculate the total return attribution
total_rt_attribution=return_attribution.sum()
# Combine the total_rt_attribution and total_rts together to compare with each
 ↪other
attribution_df=pd.concat([total_rt_attribution],axis=1).T
attribution_df.index=['TotalReturnAttribution']
attribution_df.loc['TotalReturnAttribution','PortfolioReturn']=attribution_df.
 ↪loc['TotalReturnAttribution'][:-1].sum()
attribution_df
```

[ ]:

	Asset1	Asset2	Asset3	PortfolioReturn
TotalReturnAttribution	-0.000948	0.110041	0.179663	0.109093

6.0.3 c. Calculate the ex-post risk attribution of the portfolio on each asset (2pt)

[ ]:
```
from scipy import linalg
# Y is stock returns scaled by their weight at each time
Y = (weights * rts).dropna(axis=1)

# Set up X with the Portfolio Return
X = pd.DataFrame(rts['PortfolioReturn'])
X['Intersect']=1

# Calculate the Beta and discard the intercept
B = (linalg.inv(X.T@X)@X.T@Y).iloc[:-1,:]

# Component SD is Beta times the standard Deviation of the portfolio
cSD = B * rts['PortfolioReturn'].std()

#Check that the sum of component SD is equal to the portfolio SD
print("Does the the sum of component SD is equal to the portfolio SD? {}".
 ↪format(np.isclose(cSD.sum(axis=1),rts['PortfolioReturn'].std())[0]))

# Add the portfolio SD to the cSD
cSD['Portfolio']=rts['PortfolioReturn'].std()

# Add the Vol attribution to attribution_df
columns=list(attribution_df.columns)
columns[-1]='Portfolio'
Vol_attribution=pd.DataFrame(columns=columns)
Vol_attribution.loc['VolatilityAttribution',:]=cSD.values
Vol_attribution
```

```
Does the the sum of component SD is equal to the portfolio SD? True
```

[ ]:

	Asset1	Asset2	Asset3	Portfolio
VolatilityAttribution	0.011066	0.004482	0.020816	0.036363

7 Q5

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 Gauss 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)

[ ]:
```
from scipy.stats import norm,spearmanr,multivariate_normal,t
df_price=pd.read_csv('problem5.csv',index_col='Date')
rts=return_calculate(df_price,option="DISCRETE")
```

```
rts.columns=['Return1','Return2','Return3','Return4']

# Set the Retruns to be Y
Y = rts

# Use the CDF to transform the data to uniform universe
U=[]
Model_T=[]
for i in range(Y.shape[1]):
    params=t.fit(Y.iloc[:,i].values)
    Model_T.append(t(df=params[0],loc=params[1],scale=params[2]))
    U.append(Model_T[i].cdf(Y.iloc[:,i]))

nSim = 10000

# Gaussian Copula
# Use the standard normal quantile function to transform the uniform to normal
Z=norm.ppf(U)
Z=pd.DataFrame(Z,index=Y.columns).T
# Spearman correlation
corr_spearman = spearmanr(Z,axis=0)[0]
# Simulate Normal & Transform to uniform
simU=norm.cdf(multivariate_normal.rvs(cov=corr_spearman, size=nSim))
simU=pd.DataFrame(simU,columns=Y.columns)
# Transform to T Distribution
simReturns=[]
for i in range(Y.shape[1]):
    simReturns.append(Model_T[i].ppf(simU.iloc[:,i]))

# convert simulated returns to dataframe
simReturns=pd.DataFrame(simReturns,index=Y.columns).T
```

7.0.1 a. Calculate VaR (5%) for each asset (3pt)

[ ]:
```
from RiskPackage.RiskMetrics import RiskMetrics
# Var1
VaR_1=RiskMetrics.VaR_historical(simReturns['Return1'].to_numpy(),alpha=0.
 ↪05)*df_price.iloc[-1,:]['Price1']
print("The fisrt VaR is ${:.5f}".format(VaR_1))

# Var2
VaR_2=RiskMetrics.VaR_historical(simReturns['Return2'].to_numpy(),alpha=0.
 ↪05)*df_price.iloc[-1,:]['Price2']
print("The second VaR is ${:.5f}".format(VaR_2))

# Var3
```

```
VaR_3=RiskMetrics.VaR_historical(simReturns['Return3'].to_numpy(),alpha=0.
 ↪05)*df_price.iloc[-1,:]['Price3']
print("The third VaR is ${:.5f}".format(VaR_3))

# Var4
VaR_4=RiskMetrics.VaR_historical(simReturns['Return4'].to_numpy(),alpha=0.
 ↪05)*df_price.iloc[-1,:]['Price4']
print("The forth VaR is ${:.5f}".format(VaR_4))
```

```
The fisrt VaR is $0.07549
The second VaR is $0.07713
The third VaR is $0.07580
The forth VaR is $0.07743
```

7.0.2 b. Calculate VaR (5%) for a portfolio of Asset 1 &2 and a portfolio of Asset 3&4 (4pt)

[ ]:
```
# portfolio of Asset 1 & 2
# Portfolio Value Variation = simulated returns1 * lastPrice1 + simulated␣
 ↪returns2 * lastPrice2
SimulatedVariation=pd.DataFrame(df_price.iloc[-1,:][['Price1','Price2']].
 ↪to_numpy()*simReturns[['Return1','Return2']].to_numpy()).sum(axis=1)
VaR_12=RiskMetrics.VaR_historical(SimulatedVariation.to_numpy(),alpha=0.05)
# portfolio of Asset 3 & 4
# Portfolio Value Variation = simulated returns3 * lastPrice3 + simulated␣
 ↪returns3 * lastPrice4
SimulatedVariation=pd.DataFrame(df_price.iloc[-1,:][['Price3','Price4']].
 ↪to_numpy()*simReturns[['Return3','Return4']].to_numpy()).sum(axis=1)
VaR_34=RiskMetrics.VaR_historical(SimulatedVariation.to_numpy(),alpha=0.05)

print("The VaR of Portfolio 1 & 2 is ${:.5f}".format(VaR_12))
print("The VaR of Portfolio 3 & 4 is ${:.5f}".format(VaR_34))
```

```
The VaR of Portfolio 1 & 2 is $0.15262
The VaR of Portfolio 3 & 4 is $0.15323
```

7.0.3 c. Calculate VaR (5%) for a portfolio of all 4 assets. (3pt)

[ ]:
```
# portfolio of 4 Assets
# Portfolio Value Variation = simulated returns1 * price1 + simulated returns2␣
 ↪* price2
SimulatedVariation=pd.DataFrame(df_price.iloc[-1,:].to_numpy()*simReturns.
 ↪to_numpy()).sum(axis=1)
VaR_All=RiskMetrics.VaR_historical(SimulatedVariation.to_numpy(),alpha=0.05)
print("The VaR of Portfolio of 4 Assets is ${:.5f}".format(VaR_All))
```

```
The VaR of Portfolio of 4 Assets is $0.30585
```