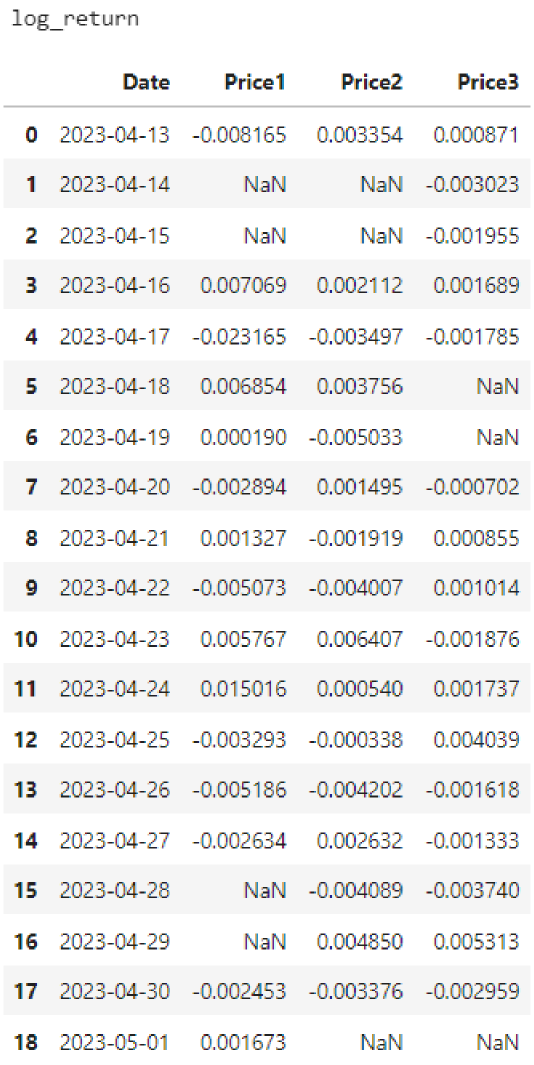
Question 1 - Using the data in “problem1.csv”

Part a - Calculate Log Returns (2pts)

I first read in data, and move the date column to the first column of the dataframe, and then, I called my function “return_cal” defined in my library “functionlib” to calculate log returns, and results are:

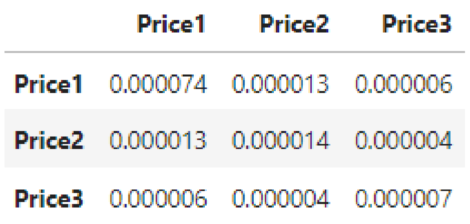


```
log_return
```

	Date	Price1	Price2	Price3
0	2023-04-13	-0.008165	0.003354	0.000871
1	2023-04-14	NaN	NaN	-0.003023
2	2023-04-15	NaN	NaN	-0.001955
3	2023-04-16	0.007069	0.002112	0.001689
4	2023-04-17	-0.023165	-0.003497	-0.001785
5	2023-04-18	0.006854	0.003756	NaN
6	2023-04-19	0.000190	-0.005033	NaN
7	2023-04-20	-0.002894	0.001495	-0.000702
8	2023-04-21	0.001327	-0.001919	0.000855
9	2023-04-22	-0.005073	-0.004007	0.001014
10	2023-04-23	0.005767	0.006407	-0.001876
11	2023-04-24	0.015016	0.000540	0.001737
12	2023-04-25	-0.003293	-0.000338	0.004039
13	2023-04-26	-0.005186	-0.004202	-0.001618
14	2023-04-27	-0.002634	0.002632	-0.001333
15	2023-04-28	NaN	-0.004089	-0.003740
16	2023-04-29	NaN	0.004850	0.005313
17	2023-04-30	-0.002453	-0.003376	-0.002959
18	2023-05-01	0.001673	NaN	NaN

Part b - Calculate Pairwise Covariance (4pt)

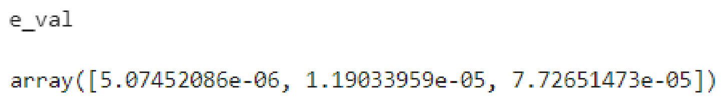
Using log returns I derived from part a, and call my defined function “missing_cov” in my library “functionlib” to calculate the pairwise covariance matrix, and the resulting matrix is:



	Price1	Price2	Price3
Price1	0.000074	0.000013	0.000006
Price2	0.000013	0.000014	0.000004
Price3	0.000006	0.000004	0.000007

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

I first ran the function called "eigh" from numpy.linalg to check eigenvalues of the resulting covariance matrix got from part b, and three eigenvalues derived are:

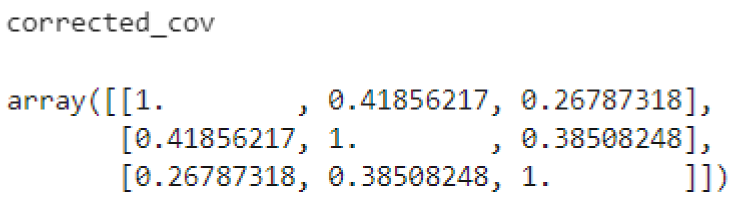


```
e_val

array([5.07452086e-06, 1.19033959e-05, 7.72651473e-05])
```

From these eigenvalues, we can determine that this covariance matrix is not PSD, since its eigenvalues are not positive or zero.

Therefore, I would use "near_psd" method to correct the matrix into a PSD one. I called a defined function called "near_PSD" in my library "functionlib" to correct my resulting matrix, and the corrected matrix is:



```
corrected_cov

array([[1.        , 0.41856217, 0.26787318],
       [0.41856217, 1.        , 0.38508248],
       [0.26787318, 0.38508248, 1.        ]])
```

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

Data like this would highly possibly show up in some financial markets, especially those of which operate 13 days in two weeks. As shown in the data, on some days, prices of certain assets exist, while prices of other assets don't – this often shows up when these assets are exchanged or traded on different markets, particularly markets across different countries.

Question 2 – “problem2.csv” contains data about a call option. Time to maturity is given in days. Assume 255 days in a year.

Part a - Calculate the call price (1pt)

I first read in data from “problem2.csv”, and I extract data from the dataframe and store them into variables respectively. Then, I call a defined function called “gbsm” in my library “functionlib” to calculate price of the call option, and the result is:

```
price

0    14.886793
dtype: float64
```

Part b - Calculate Delta (1pt)

I call a defined function called “delta_gbsm” in my library “functionlib” to calculate delta of the call option, and the result is:

```
delta

0    0.85923
dtype: float64
```

Part c - Calculate Gamma (1pt)

I call a defined function called “gamma_gbsm” in my library “functionlib” to calculate gamma of the call option, and the result is:

```
gamma

0    0.01525
dtype: float64
```

Part d - Calculate Vega (1pt)

I call a defined function called “vega_gbsm” in my library “functionlib” to calculate vega of the call option, and the result is:

```
vega

0    17.439796
dtype: float64
```

Part e - Calculate Rho (1pt)

I call a defined function called "rho_gbsm" in my library "functionlib" to calculate rho of the call option, and the result is:

```
rho

0    42.031419
dtype: float64
```

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;

Part f - Calculate VaR at 5% (2pt)

I first simulate returns of the underlying with a normal distribution for 5000 times, and use those returns to calculate simulated prices after 1 day. With those prices, I am able to calculate simulated prices of options associated with the underlying, and the table recording prices of the underlying and the call option is below:

```
sim_prices
```

	0	opt
0	84.779317	3.338524
1	109.584227	21.564220
2	90.581809	6.262134
3	111.865262	23.730510
4	115.085416	26.838705
...	...	...
4995	56.390006	0.002407
4996	130.292123	41.888241
4997	111.366829	23.254276
4998	70.701482	0.290361
4999	116.543502	28.260701

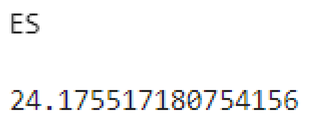
With this table, I could calculate the profit and loss of the position, and 5% VaR is calculated with the defined function called "cal_VaR" in my library "functionlib", and the result is (notice that here the VaR is a positive number, since I negate the loss before this result is returned; in other words, this positive number stands for a loss):

```
VaR

17.042036127743145
```

Part g - Calculate ES at 5% (2pt)

With the array of simulated profit and loss, I just call the defined function called "cal_ES" in my library "functionlib" to calculate ES, and the result is (notice that here the ES is a positive number, since I negate the loss before this result is returned; in other words, this positive number stands for a loss):



```
ES

24.175517180754156
```

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

This portfolio is a covered call, where traders short calls while owning the underlying. The distribution of the payoff structure is negatively skewed, representing that it would sometimes generate extraordinarily losses while its expected returns are positive.

Question 3 - Data in “problem2_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.

Part a - Calculate the Maximum Sharpe Ratio Portfolio (4pt)

I first read in data, and extract data from dataframes as needed. Then, I call a function called “optimize_Sharpe” in my library to calculate the portfolio that maximizes Sharpe Ratio, and the result is (due to time constraints, I don’t put the result into a more decent format):

```
weights

array([0.33121316, 0.32378021, 0.34500663])
```

From left to right, it stands for the weight of Asset1, Asset2, and Asset3 respectively.

Part b - Calculate the Risk Parity Portfolio (4pt)

I call a function called “risk_budget_parity” in my library to calculate the portfolio that achieves risk parity, and the result is

```
risk_par_weights
```

	Stock	w	σ
0	Asset1	0.331301	0.206985
1	Asset2	0.323724	0.211830
2	Asset3	0.344975	0.198781

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

From results above, we could notice that there are minor changes to weights of assets when we are calculating the maximum Sharpe Ratio portfolio and risk parity portfolio. There is a minor decrease in the weight of asset 2, since its standard deviation is the highest among those three assets, and this amount of decreased weight is transferred to other assets in the portfolio. But, overall, these minor differences could be ignored. The reason why these two portfolios are roughly the same is that correlations between these assets are the same, as shown below, and the Sharpe Ratio is also the same; therefore, under this circumstance, the risk parity portfolio is the maximum Sharpe Ratio portfolio.

```
functionlib.cov2cor(cov)
```

	Asset1	Asset2	Asset3
0	1.00	0.22	0.22
1	0.22	1.00	0.22
2	0.22	0.22	1.00

Question 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.

Part a - Calculate the new weights for the start of each time period (2pt)

I just first read in data, and start the iteration process by setting up some basic variables, such as the one recording the number of periods. Then, inside the iteration process, I save current weights into a matrix, and update current weights by multiplying them by the summation of 1 and returns of the period, and these updated returns are normalized, the result is:

	Asset1	Asset2	Asset3
0	0.506493	0.180245	0.313262
1	0.506878	0.160396	0.332725
2	0.530835	0.143437	0.325727
3	0.545732	0.137384	0.316884
4	0.563363	0.132167	0.304470
5	0.585406	0.137339	0.277255
6	0.612950	0.135921	0.251129
7	0.607373	0.146272	0.246356
8	0.612526	0.132423	0.255051
9	0.629634	0.114222	0.256144
10	0.601404	0.127636	0.270960
11	0.640746	0.111061	0.248193
12	0.650285	0.098119	0.251596
13	0.668849	0.091860	0.239290
14	0.656272	0.097558	0.246169
15	0.656168	0.100810	0.243021
16	0.654548	0.094491	0.250961
17	0.661554	0.090142	0.248304
18	0.654521	0.091120	0.254358
19	0.647589	0.090228	0.262183

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

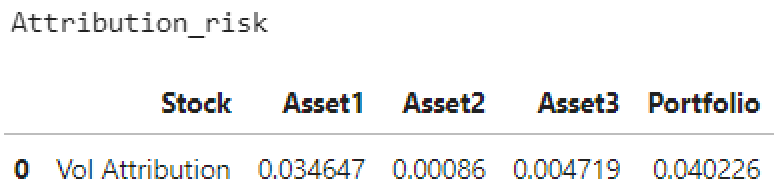
I just copied and pasted part of a defined function called “rr_attribute” in my library to conduct the ex-post return attribution, and the result is:

```
Attribution_return
```

	Stock	Asset1	Asset2	Asset3	Portfolio
0	TotalReturn	1.204677	-0.190591	0.304910	0.671324
1	Return Attribution	0.620708	-0.043908	0.094523	0.671324

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

Similar to last question, I just copied and pasted part of a defined function called "rr_attribute" in my library to conduct the ex-post return attribution, and the result is:



```
Attribution_risk
```

	Stock	Asset1	Asset2	Asset3	Portfolio
0	Vol Attribution	0.034647	0.00086	0.004719	0.040226

Question 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:

Part a - Calculate VaR (5%) for each asset (3pt)

I first applied Gaussian Copula to simulate returns of each asset, and then I compute simulate prices of each asset. Eventually, I apply the defined function cal_VaR to calculate 5% VaRs for each asset, and the result is (from left to right, it stands for the 5% VaR for each asset; note that positive numbers stand for losses):

```
VaR

array([0.03757075, 0.06623416, 0.05430831, 0.04446364])
```

Part b

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

I just add up simulated prices for each asset, and use resulting simulated portfolio values to minus present portfolio value. Then, I employ the defined function cal_VaR to calculate VaR for the portfolio, and the result is:

```
VaR_total

0.18855894931590456
```