FinTech545 - hw2

Samuel Fuller

February 2024

1 Problem 1

By calculating the weights and covariance matrix for the DailyReturn.csv data with the below formulae:

$$w_{t-i} = (1-\lambda)\lambda^{i-1}$$

$$\widehat{\operatorname{cov}(\mathrm{x},\, \mathrm{y})} = \sum_{i=1}^{n} w_{t-i} \left(x_{t-i} - \overline{x}\right)\left(y_{t-i} - \overline{y}\right)$$

Varying λ between 0.1, 0.3, 0.5, 0.7, 0.9 and using PCA on the outputted covariance matrices (which can be viewed by running the Problem1.py code), we are left with the following graphs.

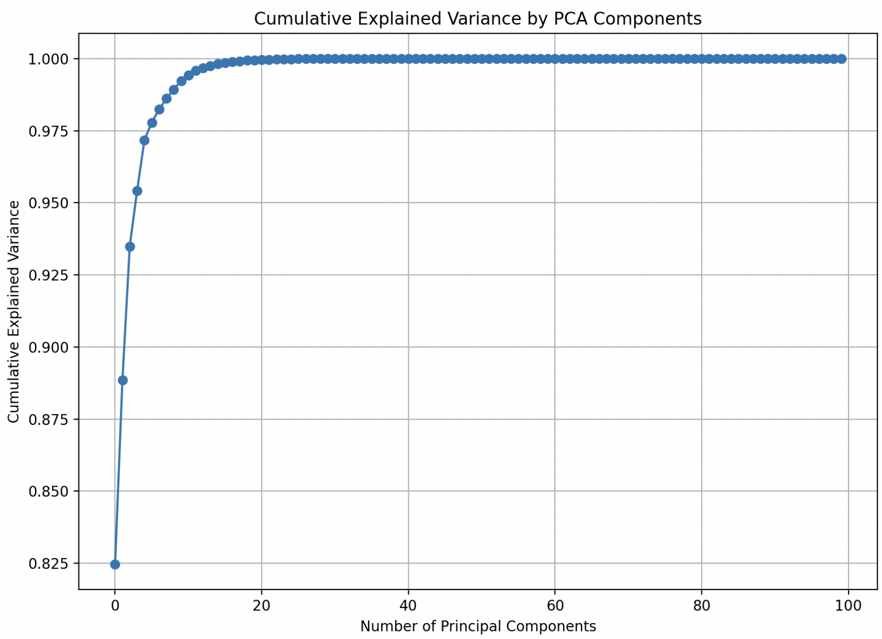

Figure 1: Cumulative variance explained by number of principal components $\lambda = 0.1$

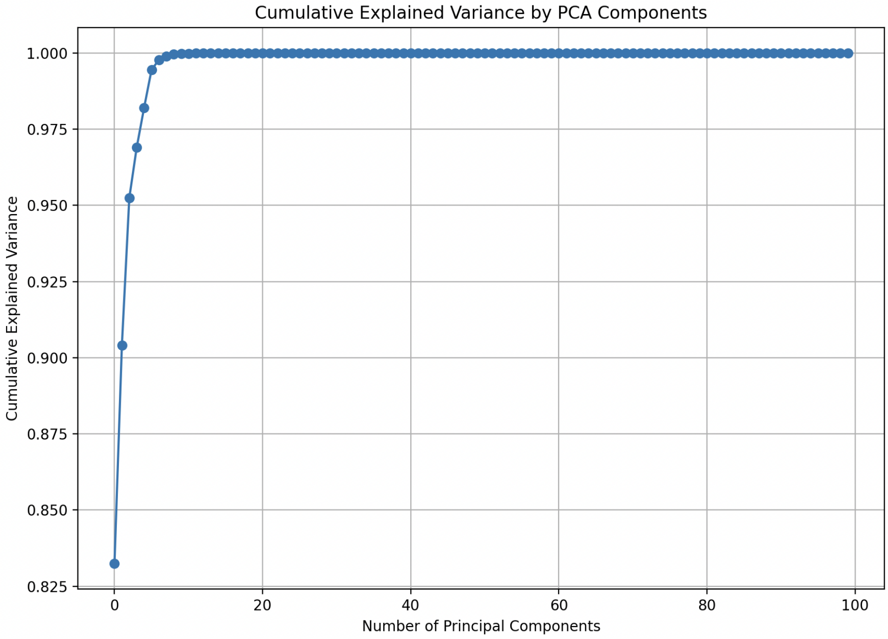

Figure 2: Cumulative variance explained by number of principal components $\lambda = 0.3$

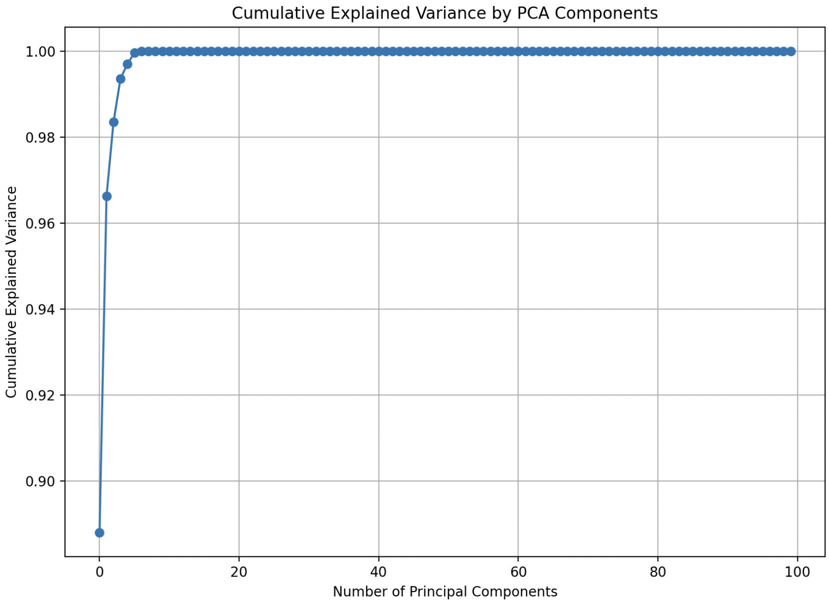

Figure 3: Cumulative variance explained by number of principal components $\lambda = 0.5$

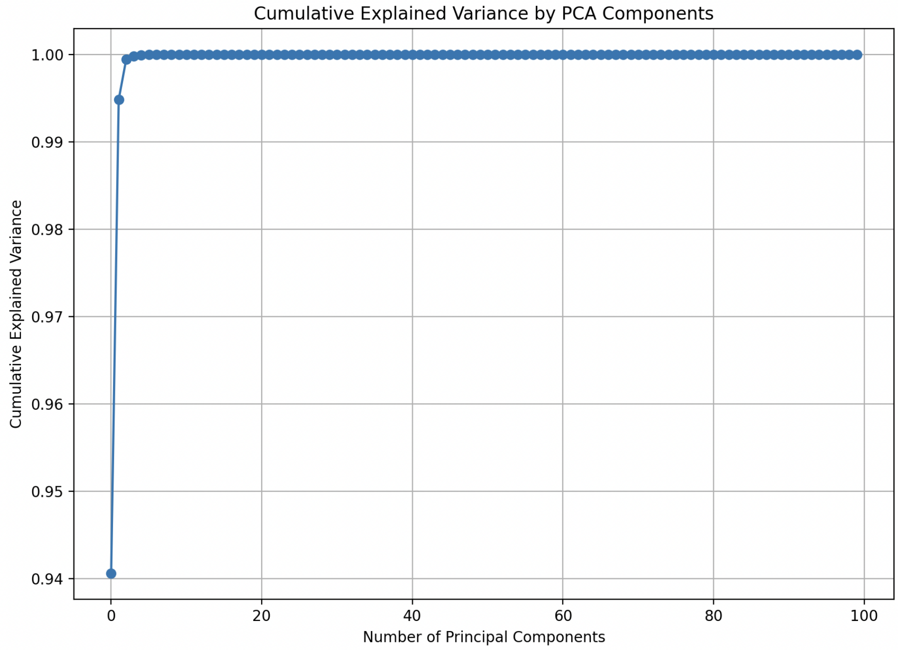

Figure 4: Cumulative variance explained by number of principal components $\lambda = 0.7$

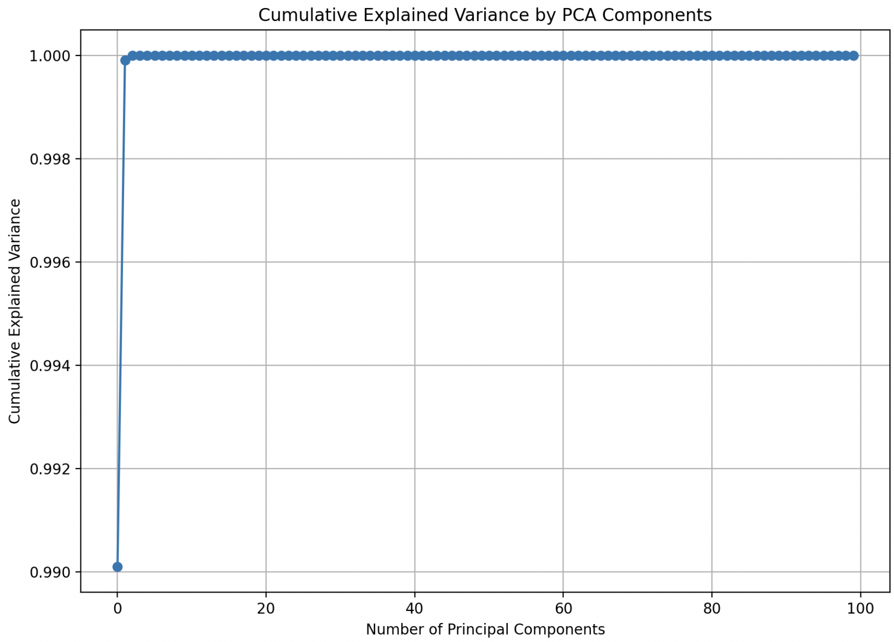

Figure 5: Cumulative variance explained by number of principal components $\lambda = 0.9$

Evidently, as λ increases, recent observations have a much higher weight, thus, the principal components will explain more of the recent variance in the data as the covariance matrix effectively represents this short-term variance as opposed to long-term variance. This will cause the cumulative variance explained by each principal component to increase faster as λ increases (directly related) as the covariance matrix captures the short-term variance closer. Lower λ will focus more on the long-term trends in the data and will take more principal components to explain the data.

2 Problem 2

After implementing the near_psd, chol_psd, and Higham functions mirroring those in the course repository and feeding in the 500x500 non-psd correlation matrix to the near_psd and Higham functions, we are left with the following Frobenius Norms and runtimes :

For an input of size n = 500:

Higham Time: 4.920338869094849
Higham Norm: 0.08964799527320237
Near Time: 4.959342956542969
Near Norm: 0.6275226557632608

For an input of size n = 1000:

Higham Time: 110.6014609336853
Higham Norm: 0.09061109584297863
Near Time: 111.67380404472351
Near Norm: 1.2614863873034337

The higham and near_psd methods run with obviously similar runtimes, however, the Higham method has a far lower norm relative to the original non-psd sigma matrix. Thus, the Higham algorithm is not only more efficient (at this scale) but is also more accurate. Please run the Problem2.py code in order to verify the correctness of the algorithm and see the nearest PSD matrices. The runtime for both algorithms is approximately $O(n^4.5)$ given the code runs 25x slower when input is doubled. Perhaps the implementation of the near_psd method is inefficient and if so, it could be worthwhile using the algorithm on larger scale matrices if it is asymptotically more efficient given the marginal difference in the norm is most likely worthwhile at this scale.

3 Problem 3

After solving for the standard Pearson correlation matrix using the np.corrcoef function as well as the np.var function for variance in addition to using the implementation to find the exponentially weighted correlations and variances using the code from question 1 in combination with the relationship between $Corr_{i,j} = \frac{Cov_{i,j}}{Cov_{i,i} * Cov_{j,j}}$ to solve for both the exponentially weighted correlation matrix and variance vector, we combine these values to formulate 4 different covariance matrices from which we will simulate 4 multivariate normal distributions using various sampling methods. Snippets of the 4 input matrices are below:

Exponential correlation + exponential variance covariance matrix:

	AAPL	MSFT	AMZN	TSLA	GOOGL	...	LMT	SYK	GM	TFC	TJX
AAPL	0.000053	0.000077	0.000119	-0.00021	0.000319	...	-0.000001	0.000105	-0.000013	0.000111	-0.000035
MSFT	0.000077	0.00013	0.000189	-0.00034	0.000486	...	-0.0	0.000167	-0.000019	0.000178	-0.000053
AMZN	0.000119	0.000189	0.000284	-0.000507	0.000739	...	-0.000001	0.000251	-0.00003	0.000266	-0.000081
TSLA	-0.00021	-0.00034	-0.000507	0.000909	-0.001311	...	0.000002	-0.000448	0.000052	-0.000477	0.000143
GOOGL	0.000319	0.000486	0.000739	-0.001311	0.001946	...	-0.000005	0.000652	-0.000079	0.000691	-0.000213
...	...	...	...	...	...	...	...	...	...	...	...
LMT	-0.000001	-0.0	-0.000001	0.000002	-0.000005	...	0.0	-0.000001	0.0	-0.000001	0.000001
SYK	0.000105	0.000167	0.000251	-0.000448	0.000652	...	-0.000001	0.000222	-0.000026	0.000235	-0.000071
GM	-0.000013	-0.000019	-0.00003	0.000052	-0.000079	...	0.0	-0.000026	0.000003	-0.000028	0.000009
TFC	0.000111	0.000178	0.000266	-0.000477	0.000691	...	-0.000001	0.000235	-0.000028	0.00025	-0.000076
TJX	-0.000035	-0.000053	-0.000081	0.000143	-0.000213	...	0.000001	-0.000071	0.000009	-0.000076	0.000023

Exponential correlation + var() covariance matrix:

	AAPL	MSFT	AMZN	TSLA	GOOGL	...	LMT	SYK	GM	TFC	TJX
AAPL	0.000501	0.000468	0.000674	-0.000909	0.00056	...	-0.000157	0.000451	-0.000619	0.000438	-0.000454
MSFT	0.000468	0.000505	0.000685	-0.000946	0.000547	...	-0.000034	0.00046	-0.000575	0.000451	-0.000443
AMZN	0.000674	0.000685	0.00096	-0.001317	0.000777	...	-0.000104	0.000643	-0.000838	0.000628	-0.000629
TSLA	-0.000909	-0.000946	-0.001317	0.001812	-0.001058	...	0.000102	-0.000882	0.001133	-0.000863	0.000857
GOOGL	0.00056	0.000547	0.000777	-0.001058	0.000636	...	-0.000127	0.00052	-0.000694	0.000507	-0.000516
...	...	...	...	...	...	...	...	...	...	...	...
LMT	-0.000157	-0.000034	-0.000104	0.000102	-0.000127	...	0.000276	-0.000069	0.000184	-0.000059	0.000103
SYK	0.000451	0.00046	0.000643	-0.000882	0.00052	...	-0.000069	0.000431	-0.00056	0.000421	-0.000421
GM	-0.000619	-0.000575	-0.000838	0.001133	-0.000694	...	0.000184	-0.00056	0.000772	-0.000544	0.000563
TFC	0.000438	0.000451	0.000628	-0.000863	0.000507	...	-0.000059	0.000421	-0.000544	0.000412	-0.00041
TJX	-0.000454	-0.000443	-0.000629	0.000857	-0.000516	...	0.000103	-0.000421	0.000563	-0.00041	0.000418

Pearson correlation + var() covariance matrix:

	AAPL	MSFT	AMZN	TSLA	GOOGL	...	LMT	SYK	GM	TFC	TJX
AAPL	0.000501	0.000407	0.000483	0.000612	0.000442	...	0.000086	0.000269	0.000404	0.00026	0.000212
MSFT	0.000407	0.000505	0.000529	0.000521	0.000475	...	0.000071	0.000282	0.00038	0.000253	0.000216
AMZN	0.000483	0.000529	0.00096	0.000743	0.000593	...	0.000058	0.000361	0.00052	0.000328	0.000296
TSLA	0.000612	0.000521	0.000743	0.001812	0.000571	...	0.000091	0.000315	0.00069	0.000305	0.000337
GOOGL	0.000442	0.000475	0.000593	0.000571	0.000636	...	0.000055	0.000303	0.000407	0.000276	0.000226
...	...	...	...	...	...	...	...	...	...	...	...
LMT	0.000086	0.000071	0.000058	0.000091	0.000055	...	0.000276	0.000055	0.000039	0.000022	0.00002
SYK	0.000269	0.000282	0.000361	0.000315	0.000303	...	0.000055	0.000431	0.000303	0.000242	0.000204
GM	0.000404	0.00038	0.00052	0.00069	0.000407	...	0.000039	0.000303	0.000772	0.000335	0.0003
TFC	0.00026	0.000253	0.000328	0.000305	0.000276	...	0.000022	0.000242	0.000335	0.000412	0.000224
TJX	0.000212	0.000216	0.000296	0.000337	0.000226	...	0.00002	0.000204	0.0003	0.000224	0.000418

Pearson correlation + exponential variance covariance matrix:

	AAPL	MSFT	AMZN	TSLA	GOOGL	...	LMT	SYK	GM	TFC	TJX
AAPL	0.000053	0.000067	0.000085	0.000141	0.000252	...	0.000001	0.000063	0.000009	0.000066	0.000016
MSFT	0.000067	0.00013	0.000146	0.000187	0.000422	...	0.000001	0.000102	0.000013	0.0001	0.000026
AMZN	0.000085	0.000146	0.000284	0.000286	0.000564	...	0.000001	0.000141	0.000018	0.000139	0.000038
TSLA	0.000141	0.000187	0.000286	0.000909	0.000707	...	0.000001	0.00016	0.000032	0.000169	0.000056
GOOGL	0.000252	0.000422	0.000564	0.000707	0.001946	...	0.000002	0.00038	0.000047	0.000376	0.000093
...	...	...	...	...	...	...	...	...	...	...	...
LMT	0.000001	0.000001	0.000001	0.000001	0.000002	...	0.0	0.000001	0.0	0.0	0.0
SYK	0.000063	0.000102	0.000141	0.00016	0.00038	...	0.000001	0.000222	0.000014	0.000135	0.000034
GM	0.000009	0.000013	0.000018	0.000032	0.000047	...	0.0	0.000014	0.000003	0.000017	0.000005
TFC	0.000066	0.0001	0.000139	0.000169	0.000376	...	0.0	0.000135	0.000017	0.00025	0.000041
TJX	0.000016	0.000026	0.000038	0.000056	0.000093	...	0.0	0.000034	0.000005	0.000041	0.000023

After simulating 25,000 draws from each matrix using 4 different methods and then calculating the implied covariance matrices, we compare the simulated matrix to the input matrices as well as the runtime for various simulation methods:

Exponential correlation + exponential variance covariance matrix:

Direct Simulation: Frobenius Norm = 0.00009601201991335188 Time = 0.10437202453613281
PCA 100%: Frobenius Norm = 0.020657524746838945, Time = 0.026415109634399414
PCA 75%: Frobenius Norm = 0.02065750333195619, Time = 0.014809131622314453
PCA 50%: Frobenius Norm = 0.020657524688872737, Time = 0.016313791275024414

Exponential correlation + var() covariance matrix:

Direct Simulation: Frobenius Norm = 0.0002905593579213646, Time = 0.11176300048828125

PCA 100%: Frobenius Norm = 0.04338936847145384, Time = 0.01790595054626465
PCA 75%: Frobenius Norm = 0.04338947293776888, Time = 0.013150930404663086
PCA 50%: Frobenius Norm = 0.04338940193901488, Time = 0.012130022048950195

Pearson correlation + var() covariance matrix:

Direct Simulation: Frobenius Norm = 0.0002497281085953464, Time = 0.10252022743225098
PCA 100%: Frobenius Norm = 0.027090905627275185, Time = 0.013296842575073242
PCA 75%: Frobenius Norm = 0.02709091287492097, Time = 0.015106916427612305
PCA 50%: Frobenius Norm = 0.027090907736492493, Time = 0.014793872833251953

Pearson correlation + exponential variance covariance matrix:

Direct Simulation: Frobenius Norm = 0.00018140554424758059, Time = 0.10321402549743652
PCA 100%: Frobenius Norm = 0.011204737505728424, Time = 0.013047218322753906
PCA 75%: Frobenius Norm = 0.011204734289381822, Time = 0.01413726806640625
PCA 50%: Frobenius Norm = 0.011204737945005226, Time = 0.013210058212280273

The direct simulation is $6-9$ times slower than the 50% PCA but has accuracy (norm) 215, 143, 108, 62 times more accurate (relative to the 50% PCA) across the various covariance matrix inputs defined above (respectively), clearly accuracy scales faster than runtime and thus the trade off for higher runtime in favor of higher accuracy is worthwhile.