task4_durei

January 20, 2025

1 Task 4 by Durei

1.1 Empirical Analysis of ETFs

We consider Nasdaq-100 index ([^NDX](https://uk.finance.yahoo.com/quote/%5ENDX/)) to conduct this analysis via components. Nasdaq-100 ETF such as ^VOO is a type of funds that tracks Nasdaq-100 index by holding the similar weight of stocks or other proxy ways.

[2]:
```
import datetime as dt
from typing import Dict, List

import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
import pandas as pd

import yfinance as yf

from plotly import express as px
from plotly import graph_objects as go
from plotly.subplots import make_subplots
```

1.2 4.a Find the 30 largest holdings.

Nasdaq provides the list of stock names of Nasdaq-100 each business day on their website. We start from loading up the least of symbols of a day (2024/Jan/17).

[3]:
```
# Symbol lists of the day sourced from the Nasdaq website, please refer to the␣
↪above link
ndx_syms_df = pd.read_csv('EODWeightings_20250117_NDX.csv')
# ndx_syms
```

[4]:
```
marketcap_syms = {}
for sym in ndx_syms_df['Security Symbol'].to_list():
    # Let's skip 'GOOG' as we have 'GOOGL' that does almost the same already␣
↪with voting rights.
    if sym == 'GOOG':
        continue
```

```
    _ticker = yf.Ticker(sym)
    marketcap_syms[sym] = _ticker.info['marketCap']

marketcap_syms_df = pd.Series(marketcap_syms).to_frame('Cap')
```

[5]:
```
N_top_syms = 30
top_N_df = marketcap_syms_df.sort_values('Cap', ascending=False).iloc[:
 ↪N_top_syms]
# top_N_df
```

1.3 4.b Fetch at least 6 months of data (~ 120 data points).

[6]:
```
d_start = dt.date(2024,7,17)
d_end = dt.date(2025,1,17)

df_daily_ohlc = yf.download(top_N_df.index.to_list(), start=d_start, end=d_end)
df_daily_ohlc.head(3)
```

```
[*********************100%***********************]  30 of 30 completed
```

[6]:
```
Price                       Adj Close                                          \
Ticker                           AAPL        ADBE        AMAT         AMD
Date
2024-07-17 00:00:00+00:00  228.364136  563.090027  219.133743  159.429993
2024-07-18 00:00:00+00:00  223.674728  556.849976  216.724014  155.770004
2024-07-19 00:00:00+00:00  223.804428  551.000000  209.365463  151.580002

Price                                                                          \
Ticker                           AMGN        AMZN         ARM        ASML
Date
2024-07-17 00:00:00+00:00  330.704315  187.929993  161.699997  928.163513
2024-07-18 00:00:00+00:00  326.073822  183.750000  158.330002  920.286621
2024-07-19 00:00:00+00:00  326.389130  183.130005  163.399994  891.626892

Price                                             …    Volume          \
Ticker                           AVGO        AZN  …      MSFT     NFLX
Date                                              …
2024-07-17 00:00:00+00:00  155.053223  79.272766  …  21778000  4017300
2024-07-18 00:00:00+00:00  159.566238  77.583145  …  20794800  7575800
2024-07-19 00:00:00+00:00  156.415070  78.229179  …  20940400  9815600

Price                                                                           \
Ticker                          NVDA      PDD      PEP      PLTR      QCOM
Date
2024-07-17 00:00:00+00:00  390086200  9823200  7671000  45203500  16156200
2024-07-18 00:00:00+00:00  320979500  5165200  6228600  76485600   9960600
```

```
2024-07-19 00:00:00+00:00  217223800  4888700  5332800  49581600   9195700

Price
Ticker                          TMUS       TSLA      TXN
Date
2024-07-17 00:00:00+00:00  5228500  115584800  7792500
2024-07-18 00:00:00+00:00  4236900  110869000  5497200
2024-07-19 00:00:00+00:00  2494900   87403900  4868000

[3 rows x 180 columns]
```

1.4 4.c Compute the daily returns.

[7]:
```
# Adjust Close is the clean price column after the company stock events␣
↪adjusted to price - dividend payment, stock split, etc.
df_daily_px = df_daily_ohlc['Adj Close']

# Prepare log and pct return - we are going to use log return for the rest of␣
↪analysis
df_daily_logr = np.log((df_daily_px / df_daily_px.shift(1, axis=0)).
↪dropna(axis=0))
df_daily_pctr  = df_daily_px.pct_change().dropna(axis=0)
```

[8]:
```
df_daily_logr.head().iloc[:3,:3]
```

[8]:
```
Ticker                          AAPL      ADBE      AMAT
Date
2024-07-18 00:00:00+00:00 -0.020749 -0.011144 -0.011058
2024-07-19 00:00:00+00:00  0.000580 -0.010561 -0.034543
2024-07-22 00:00:00+00:00 -0.001561  0.006909  0.060932
```

[9]:
```
df_daily_pctr.head().iloc[:3,:3]
```

[9]:
```
Ticker                          AAPL      ADBE      AMAT
Date
2024-07-18 00:00:00+00:00 -0.020535 -0.011082 -0.010997
2024-07-19 00:00:00+00:00  0.000580 -0.010505 -0.033954
2024-07-22 00:00:00+00:00 -0.001560  0.006933  0.062827
```

1.5 4.d Compute the covariance matrix.

[10]:
```
# Covariance
logr_cov = df_daily_logr.cov()
logr_cov.iloc[:3, :3]
```

[10]:
```
Ticker      AAPL      ADBE      AMAT
Ticker
AAPL    0.000175  0.000085  0.000130
ADBE    0.000085  0.000496  0.000215
AMAT    0.000130  0.000215  0.000896
```

[11]:
```python
# Std
logr_std = df_daily_logr.std(axis=0)
logr_std = logr_std.to_frame('std')
logr_std.head(3)
```

[11]:
```
             std
Ticker
AAPL    0.013239
ADBE    0.022265
AMAT    0.029940
```

[12]:
```python
# Correlation matrix
df_daily_logr.corr().iloc[:3,:3]
```

[12]:
```
Ticker     AAPL      ADBE      AMAT
Ticker
AAPL    1.00000  0.288680  0.328800
ADBE    0.28868  1.000000  0.321828
AMAT    0.32880  0.321828  1.000000
```

1.6 4.e Compute the PCA

We do PCA with with sklearn library in this analysis for simplicity of use, but statsmodels library offers more comprehensive analysis service.

[13]:
```python
from sklearn import decomposition, preprocessing
```

[14]:
```python
# apply daily log-return timeseries to PCA
n_pcs = min(6, N_top_syms) # number of principal components to retain
pca = decomposition.PCA(n_components=n_pcs)

df_daily_logr_scaled = preprocessing.scale(df_daily_logr) # we 'standardise'␣
↪data (normalise each timeseries: mean to 0 & std to 1)
df_daily_logr_scaled = pd.DataFrame(df_daily_logr_scaled, columns =␣
↪df_daily_logr.columns, index= df_daily_logr.index)
_ = pca.fit(df_daily_logr_scaled)
```

[15]:
```python
principlecomponents_list = [f'PC{i}' for i in range(pca.n_components)] #␣
↪['PC0', 'PC1', ... ]
domain_list = df_daily_logr.columns # ['AAPL', 'AMZN', ...]
```

```python
# eigenvectors list as columns of dataframe
loadings = pd.DataFrame(pca.components_.T, columns=principlecomponents_list, index=domain_list)
loadings.T.iloc[:3]
```

[15]:

```
Ticker      AAPL      ADBE      AMAT       AMD      AMGN      AMZN       ARM  \
PC0     0.185745  0.152358  0.246796  0.225760  0.095505  0.232218  0.235337
PC1     0.029448  0.169563 -0.216957 -0.182720  0.139838  0.043447 -0.193125
PC2     0.186652  0.008961  0.098052  0.015097  0.176988 -0.091555 -0.008461

Ticker      ASML      AVGO       AZN  …      MSFT      NFLX      NVDA  \
PC0     0.208558  0.214172  0.054078  …  0.238074  0.190617  0.237118
PC1    -0.263556 -0.147598  0.008469  …  0.026057  0.122101 -0.135082
PC2     0.106610 -0.096821  0.559547  …  0.093960 -0.003714 -0.055299

Ticker       PDD       PEP      PLTR      QCOM      TMUS      TSLA       TXN
PC0     0.049872 -0.014525  0.156484  0.246294  0.069734  0.185461  0.212282
PC1    -0.268362  0.327391  0.086289 -0.229051  0.365935  0.015428 -0.094917
PC2     0.103125  0.323107 -0.313966  0.007590 -0.126447 -0.286164 -0.004822

[3 rows x 30 columns]
```

One thing to remind, we can get the principal component vectors as dot product of the scaled data with eigenvectors as discussed in M1-L4.

[16]:

```python
# eigenvalues-squared list as a diagonal matrix
factors = pd.DataFrame(np.diag(pca.explained_variance_), columns=principlecomponents_list, index=principlecomponents_list)
factors
```

[16]:

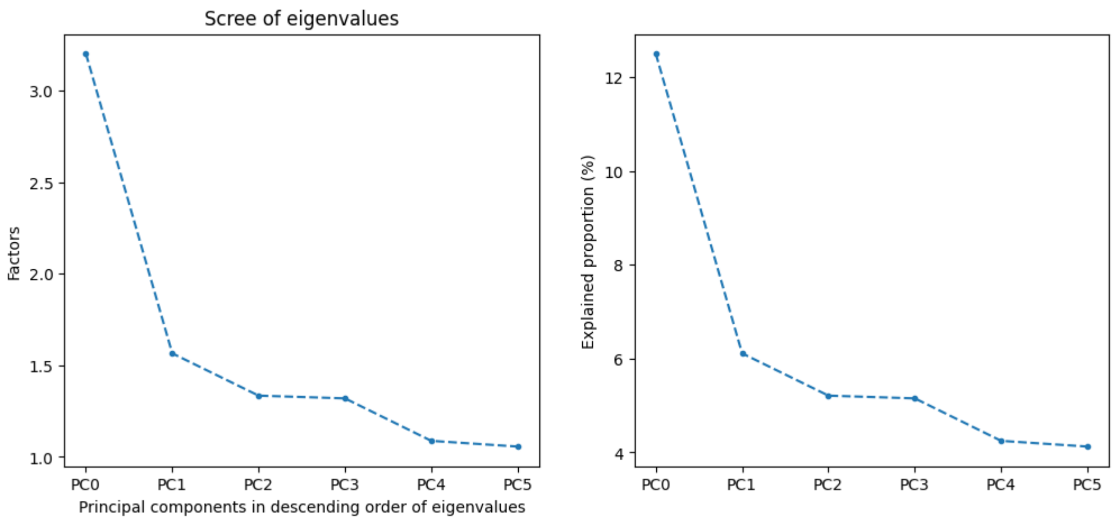
	PC0	PC1	PC2	PC3	PC4	PC5
PC0	10.243207	0.000000	0.000000	0.00000	0.000000	0.000000
PC1	0.000000	2.450485	0.000000	0.00000	0.000000	0.000000
PC2	0.000000	0.000000	1.781195	0.00000	0.000000	0.000000
PC3	0.000000	0.000000	0.000000	1.74189	0.000000	0.000000
PC4	0.000000	0.000000	0.000000	0.00000	1.183618	0.000000
PC5	0.000000	0.000000	0.000000	0.00000	0.000000	1.116709

[17]:

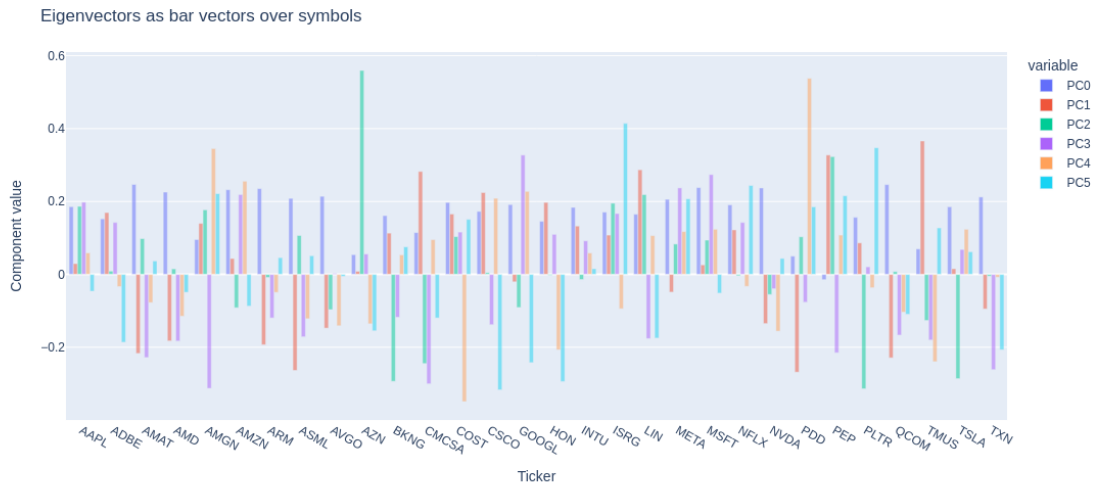
```python
fig, axs = plt.subplots(1,2, figsize=(12, 5))

eigenvalues = np.diagonal(factors.pow(0.5).values)
_1 = pd.Series(eigenvalues, index=principlecomponents_list).plot(marker='.', linestyle='--', ax=axs[0])
axs[0].set_xlabel('Principal components in descending order of eigenvalues')
axs[0].set_ylabel('Factors')
axs[0].set_title('Scree of eigenvalues')
```

```
# Let's quickly call up the full rank PCA for the explained proportion plotting
pca_full = decomposition.PCA(n_components=N_top_syms)
_ = pca_full.fit_transform(df_daily_logr_scaled)
eigenvalues_full = np.sqrt(pca_full.explained_variance_)
_2 = pd.Series(eigenvalues/np.sum(eigenvalues_full)*100,␣
 ↪index=principlecomponents_list).plot(marker='.', linestyle='--', ax=axs[1])
axs[1].set_ylabel('Explained proportion (%)')
```

[17]: Text(0, 0.5, 'Explained proportion (%)')



[18]:
```
px.bar(loadings, barmode='group', opacity=0.5).update_layout(
    yaxis_title='Component value',
    title='Eigenvectors as bar vectors over symbols',
    width=1100, height=500
)
```



One observation in the above is that the first principle component (PC0) represents the level shift (shift on the same direction throughout the entire symbols) is observed. This is a typical phenomenon for panels with certain correlation bound by any synchronisation mechanism - such as index investment of majority buy-sides - that rules the dynamics. This applies to not only stock index but also fixed income yields.

[19]:
```
# Covariance after PCA compressing rank down to 'n_components' as defined above
pca_cov = pd.DataFrame((loadings @ factors) @ loadings.T , columns=domain_list,␣
↪index=domain_list)
# Covariance of original dataset
full_cov = pd.DataFrame(df_daily_logr_scaled, columns=domain_list).cov()
```

Post decomposition sanity check (or determining how much PCs to pick up with some quantitative metric), we can measure the difference between the true covariance and PCA compressed covariance as like below:

[20]:
```
def get_l2_norm(vect):
    return np.sqrt(vect.T @ vect)

def get_dotproduct(vect1, vect2):
    return (vect1.T @ vect2) / (get_l2_norm(vect1) * get_l2_norm(vect2))

def get_dotproduct_angle(vect1, vect2):
    return np.acos( max(min(1.0,get_dotproduct(vect1, vect2)),-1) )

# Get difference in terms of dot product angle between the full covariance␣
↪matrix and truncated covariance matrix via PCA
pca_cov_flatten = pca_cov.values.reshape(-1)
full_cov_flatten = full_cov.values.reshape(-1)

divergence_angle_pca_vs_full = get_dotproduct_angle(pca_cov_flatten,␣
↪full_cov_flatten)
```

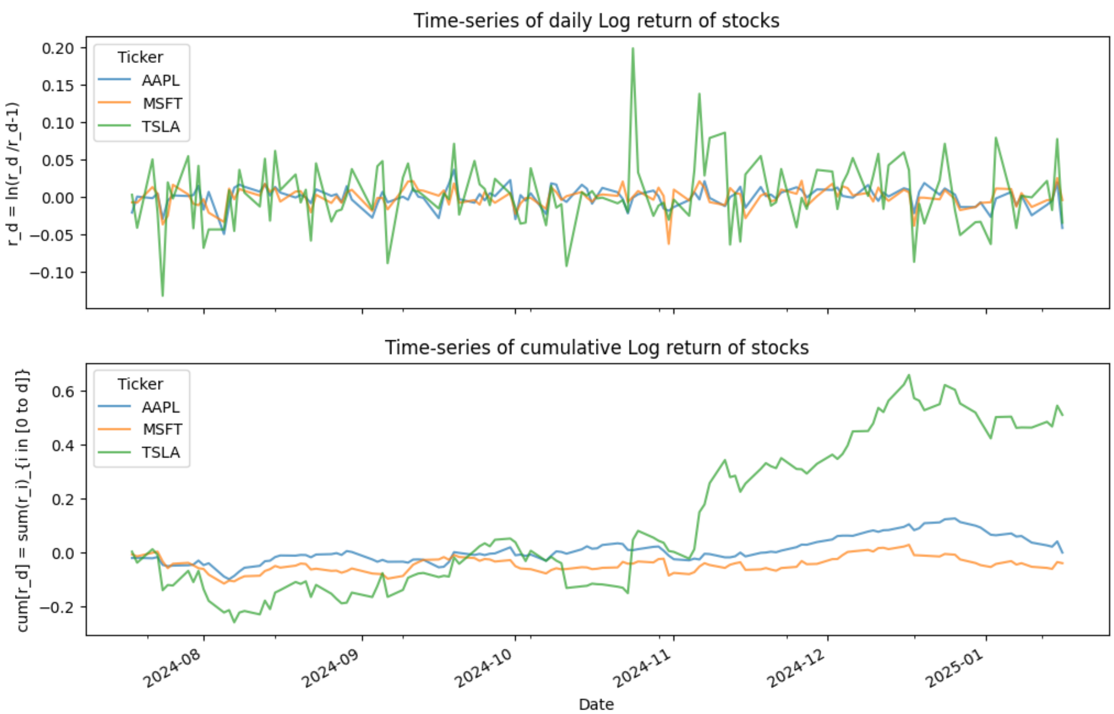
```
print(f'divergence_angle_pca({n_pcs} out of {N_top_syms} PCs used)_vs_full =␣
↪{divergence_angle_pca_vs_full:.3g} radian ({divergence_angle_pca_vs_full/np.
↪pi*180.:.3g} degree)')
```

```
divergence_angle_pca(6 out of 30 PCs used)_vs_full = 0.245 radian (14.1 degree)
```

With just 20% of all the principal components, we can reconstruct the covariance with just 14 degree disimilarity.

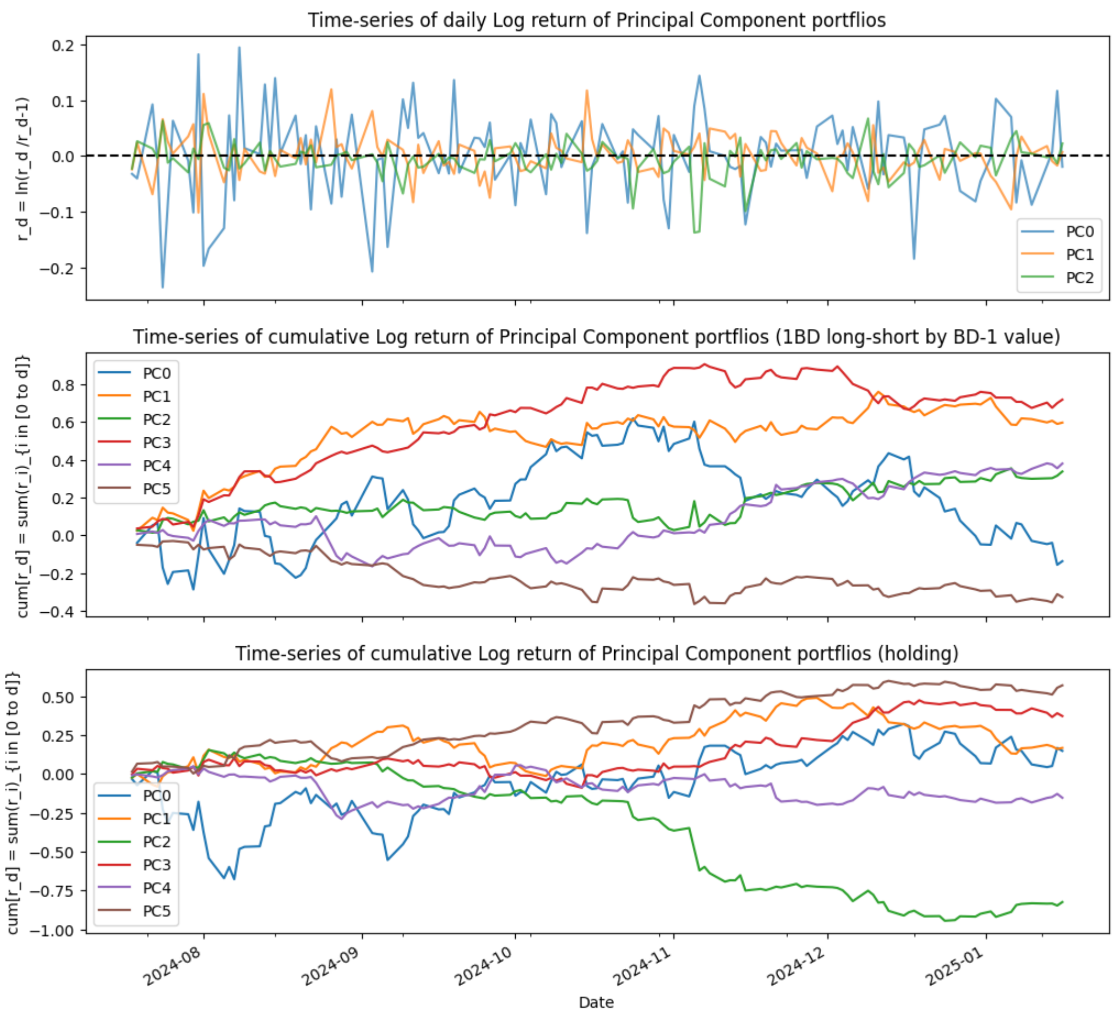
Now we extract example major principal components and present is use for the time-series analysis of returns

[21]:
```
# First, let's see how the per symbol return occurs
fig_dr, axs_dr = plt.subplots(2, 1, figsize=(12,8), sharex=True)
df_daily_logr[['AAPL', 'MSFT', 'TSLA']].plot(ylabel='r_d = ln(r_d /r_d-1)'␣
↪,title='Time-series of daily Log return of stocks', alpha=0.7, ax=axs_dr[0])
df_daily_logr[['AAPL', 'MSFT', 'TSLA']].cumsum(axis=0).plot(ylabel='cum[r_d] =␣
↪sum(r_i)_{i in [0 to d]}' ,title='Time-series of cumulative Log return of␣
↪stocks', alpha=0.7, ax=axs_dr[1])
plt.show()
```



[22]:
```
fig_cr, axs_cr = plt.subplots(3, 1, figsize=(12,12), sharex=True)
# PC_i = X @ V where X is the timeseries matrix and V is the column-wise␣
↪eigenvectors array
```

```python
# unlike 'FD-MScFE600-M1-L4', we get principal components (standisation not applied) to see real returns
df_daily_logr_principal_components = np.log((np.exp(df_daily_logr) - 1) @ loadings + 1) # To get sum(w_sym * r_{date,sym}}, we strip off logarithm then combine, then put logarithm back

# Daily PC portfolios return (a.k.a rel-value)
df_daily_logr_principal_components.iloc[:,:3].plot(ylabel='r_d = ln(r_d / r_d-1)' ,title='Time-series of daily Log return of Principal Component portflios', alpha=0.7, ax=axs_cr[0])
axs_cr[0].axhline(y=0, color='black', linestyle='--')

# Accumulation of PC portfolios P/L via symbols weighting policy that the last Business day's rel-value cheapness/richness tells today's position as we do it every day.
(df_daily_logr_principal_components.iloc[1:] * -np.sign(df_daily_logr_principal_components.shift(1)).iloc[1:]).cumsum(axis=0).plot(ylabel='cum[r_d] = sum(r_i)_{i in [0 to d]}' ,title='Time-series of cumulative Log return of Principal Component portflios (1BD long-short by BD-1 value)', ax=axs_cr[1])

df_daily_logr_principal_components.cumsum(axis=0).plot(ylabel='cum[r_d] = sum(r_i)_{i in [0 to d]}' ,title='Time-series of cumulative Log return of Principal Component portflios (holding)',  ax=axs_cr[2])
plt.show()
```



The cumulative log return timeseris of principal components are particularly interesting in use of relative value factor search. Every principal components can be understood as a *factorised* portfolio of stocks as they are a linear combinations of individual stocks with weights given by eigenvectors. - Any PCA provides orthogonal weights vector space each of which is orthogonal to every others, each this factorised portfolio is *uncorrelated* to others offering us stylised positioning - The daily return of factorised portolio is *mean-reverting* by PCA extracts the data points around its principal axis in covariance space, which tells us whether the portfolio is cheap/rich everyday by checking the divergence of the portfolio from the principal axis. If the market regime changes, one should reset the PC model. - However, the *design of policy* long-short positioning matters on returns and is not straightforward but requires careful backtest based optimisation as well as validation. For instance, the cheapness/richness of the last business day as a long/short indicator for today has higher positive odds to earn positive return but it is not always the case - PC5 gives favourable returns for the last day value lookup for daily eval strategy whereas the PC2 portfolio goes opposite. It's the most naive policy we can take, but we have more options to search on backtesting - change the weighting horizon from 1BD to a few more, combining more past dates to lookup to generate trade sign and weights, and etc. - For the holding strategy (long-short at the beginning and keep the position), one can choose the portfolio with upward cumulative returns like PC3. PCA for this strategy offers a stylised investment as mentioned above, but the direction of cumulative is not statistically based in this way.

1.7 4.f Compute the SVD

[23]:
```
U, S, Vh = np.linalg.svd(df_daily_logr, full_matrices=True) # S is eigenvalues␣
↪array, V_hermitian is eigen(row)vectors matrix
print(U.shape, S.shape, Vh.shape)
```

```
(126, 126) (30,) (30, 30)
```

Check if the decomposition was successful by reconstructing to back to the data matrix: full rank SVD should give 0 disimilarity.

[24]:
```
df_daily_logr_svd = pd.DataFrame(U[:,:N_top_syms] @ np.diag(S) @ Vh,␣
↪index=df_daily_logr.index, columns=df_daily_logr.columns)
df_daily_logr_svd.iloc[:3,:3]
```

[24]:
```
Ticker                         AAPL      ADBE      AMAT
Date
2024-07-18 00:00:00+00:00 -0.020749 -0.011144 -0.011058
2024-07-19 00:00:00+00:00  0.000580 -0.010561 -0.034543
2024-07-22 00:00:00+00:00 -0.001561  0.006909  0.060932
```

[25]:
```
# Get difference in terms of dot product angle between the full covariance␣
↪matrix and truncated covariance matrix via PCA
svd_data_flatten = df_daily_logr_svd.values.reshape(-1)
orig_data_flatten = df_daily_logr.values.reshape(-1)

disimilarity_angle_svd = get_dotproduct_angle(svd_data_flatten,␣
↪orig_data_flatten)
print(f'divergence_angle_pca(full rank)_vs_full = {disimilarity_angle_svd:.3g}␣
↪radian ({disimilarity_angle_svd/np.pi*180.:.3g} degree)')
```

```
divergence_angle_pca(full rank)_vs_full = 0 radian (0 degree)
```

1.8 Summary and further discussion

Index Constituents Extraction In this report, the analysis begins by extracting historical data for the top Nasdaq-100 stocks to form a representative dataset. Historical price data is converted into daily returns, a critical transformation for component analysis. Returns play a central role in this report, as they serve as the foundation for understanding stock movements as well as the source of searching profitable portfolio strategies based on historical return dynamics. This transformation normalizes price data, removing scale dependencies and enabling comparisons across assets with different price levels. By focusing on returns, the analysis highlights volatility and co-movements, essential for uncovering statistically based tradeable patterns.

Returns and Their Importance Returns play a pivotal role in this analysis of stock price dynamics in terms of volatility and return statistics of single names and cointegrated moves of symbols, as well as addressing the historical data-based judgement on portfolio management. Price to return conversion effectively trnasform the stock dynamics from levels to difference where the stochasticity of the dynamics become much more visible. The transformation is useful to aggregate index/portfolio performance in later steps. Subsequently, we 'standardized' price data for meaningful comparisons across assets, and form uncover the volatility and correlations, which are crucial for identifying uncorrelated factors that offer several benefits for the portfolio management. The historical return data analyzed in the report captures the dynamics of past performance, allowing for a deeper understanding of market trends and the construction of return-based portfolios.

PCA Analysis We employed the Principal Component Analysis (PCA)to identify dominant market drivers from the covariance matrix of the returns data. Eigenvectors from PCA of returns data map the data into a set of orthogonal principal components, where the first principal component explains the largest share of the total variance. For example, in the dataset analyzed, the first component captured the broad co-movement of Nasdaq-100 stocks, reflecting market-wide trends. Subsequent components highlighted more niche factors, such as sectoral trends or idiosyncratic stock movements. PCA's ability to isolate these orthogonal patterns makes it a powerful tool for reducing dimensionality and identifying tradeable factors within financial data.

Stylized Portfolios by Factor We demonstrates how PCA results can be used to construct stylized portfolios based on specific factors derived from principal components. By weighting portfolios according to eigenvectors, investors can align their holdings with specific market drivers. For instance, in this analysis, factor-aligned portfolios were created to capture opportunities in sectoral or market-wide movements while filtering out unrelated noise. These portfolios provide a mechanism to isolate and exploit specific return dynamics while avoiding overexposure to other factors. This method is particularly useful for constructing relative value strategies, where returns from uncorrelated factors are key to optimizing performance.

SVD Analysis The report applies the Singular Value Decomposition (SVD) to complement PCA in analyzing the structure of the Nasdaq-100 dataset. As studied in M3-L4 of the lecture, PCA and SVD methods in terms of math are in the same line of the spectral analysis of the stocks' return timeseries matrix, as the SVD outputs constitute the principal covariance decomposition. The SVD outputs and PCA outputs via SVD offer perpendicular vector bases that are uncorrelated to each other, and in portfolio positioning application, it essentially means the factor portfolios by this analysis isolate return dynamics of the portfolio from other factors. Hence, the management portfolio can be stylized with a single consistent view on the market, which is useful in a market with a consistent regime. In this report, singular values quantified the distribution of variance, reinforcing insights from PCA while also revealing structural nuances in the data.

PCA vs. SVD Here the PCA and SVD processes are applied to highlight different aspects of the Nasdaq-100 dataset. While PCA identifies eigenvectors and eigenvalues from the covariance matrix, SVD decomposes the returns matrix directly into singular values and corresponding vectors. As studied in M3-L4 of the lecture, both methods align mathematically through spectral analysis. PCA is particularly effective in isolating co-movements by explaining variance through orthogonal components, making it ideal for identifying dominant market-wide or sector-specific drivers. On the other hand, SVD provided deeper insights into structural patterns, especially when the dataset exhibited asymmetries or irregularities. Together, PCA and SVD complement each other by isolating unique, orthogonal return drivers and ensuring that portfolio construction is free from overlapping factor exposures.

Eigenvectors, Eigenvalues, and Singular Values The report highlights how eigenvectors, eigenvalues, and singular values provide actionable insights into the data. Eigenvectors derived from PCA represent principal components that map stock movements into uncorrelated factors, while eigenvalues measure the variance explained by these components. Singular values from SVD extend this concept, quantifying the relative importance of each component directly from the returns matrix. For the Nasdaq-100 analysis, eigenvalues and singular values both revealed the dominance of specific components, providing clarity on which factors to prioritize when constructing portfolios.

[ ]:

```

```