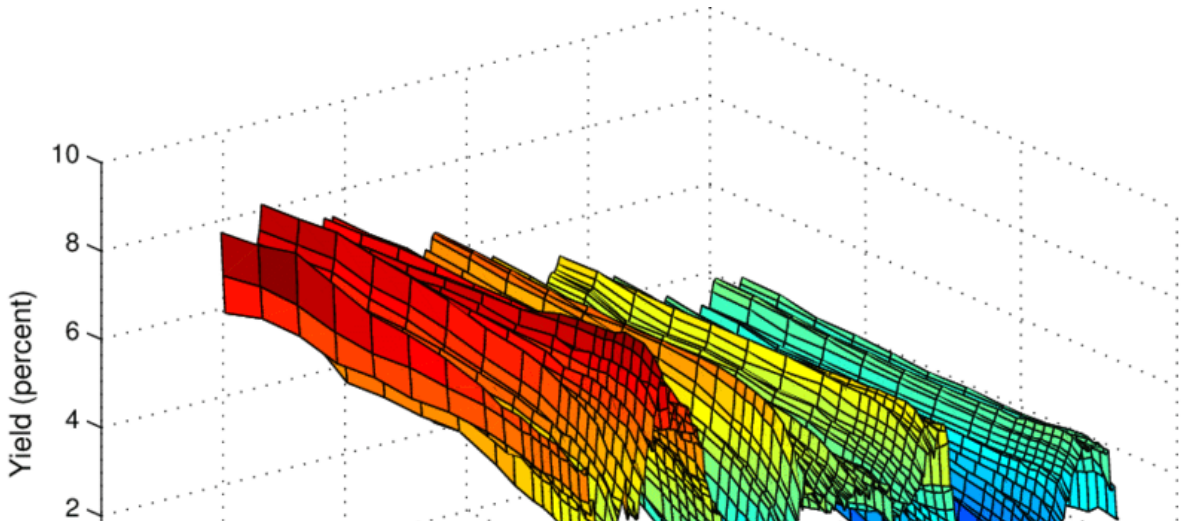
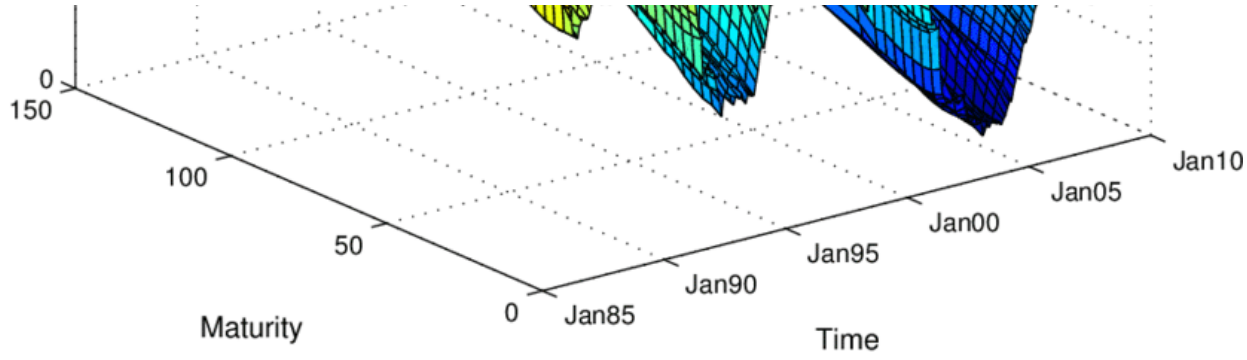
Applying PCA to the yield curve — the hard way

Learn how to apply one of the most popular applications of principal components analysis using current financial data in python.

Nathan Thomas Feb 25, 2020 · 6 min read ★





Source: Diaa Noureldin, ResearchGate.

Nathan Thomas

Data Scientist.

Follow

The yield curve is a line that plots the various interest rates of ith equal credit quality and different maturities. Government bonds are said to have negligible default risk, as the government can simply borrow more money to finance their repayments. According to the expectations theory of interest rates, the yield curve is made up of two aspects:

63

1. *An average* of market expectations concerning future short-term interest rates.

2. *The term premium* — the extra compensation an investor receives for holding a longer-term bond. This is essentially because of the time value of money — £100 is worth more today than it is worth tomorrow, due to its potential earning capacity due to interest. Therefore, for a fixed-income investment to be worth the extra time the investor must part with their cash, the bond issuer must pay the investor some extra amount.

We can model these aspects of the yield curve using principal

components decomposition. Data has two main properties: noise and signal. Principal components analysis aims to extract the signal and reduce the dimensionality of a dataset; by finding the **least amount of variables that explain the largest proportion of the data.** It does this by transforming the data from a correlation/covariance matrix onto a subspace with fewer dimensions, where all explanatory variables are orthogonal (perpendicular) to each other, i.e there is no multicollinearity *(Caveat: these are statistical properties and do not necessarily have an economic interpretation)*.

For this analysis I will use various UK government bond spot rates from 0.5 years up to 10 years to maturity (provided by the Bank of England), to see if we can model the yield curve and its slope. When finding the principal components of the yield curve, the main theory held by econometricians is that:

PC1 = constant ≈ long term interest rate ≈ R*

PC2 = slope ≈ term premium

PC3 = curvature

There is a function in scikit-learn to perform PCA, however, to best understand it let's do it manually (it isn't actually *that* hard)!

```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np


# Import Bank of England spot curve data from excel
df = pd.read_excel("GLC Nominal month end data_1970 to
2015.xlsx",
                   index_col=0, header=3, dtypes =
"float64", sheet_name="4. spot curve", skiprows=[4])


# Select all of the data up to 10 years
df = df.iloc[:,0:20]


df.head()
```

After importing the relevant modules needed, and importing the spot curve data from the Bank of England, we can see in the DataFrame that there are some *nan* values due to missing data. We can drop these as they won't greatly affect the analysis.

years:	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
1970-07-31	0.046342	0.137478	0.114726	0.075453	0.040053	0.011046	-0.011945	-0.029681	-0.042951	-0.052381
1970-08-31	0.052648	0.089836	0.084056	0.064191	0.042049	0.021638	0.004160	-0.010129	-0.021270	-0.029342
1970-09-30	0.070268	0.094800	0.076648	0.052209	0.028983	0.008886	-0.007734	-0.021033	-0.031266	-0.038647
1970-10-31	0.107300	0.167124	0.168778	0.143970	0.112772	0.082847	0.056698	0.034964	0.017657	0.004651
1971-01-31	0.066607	0.068937	0.061040	0.056791	0.057549	0.062095	0.069180	0.077821	0.087419	0.097681
1971-02-28	0.114194	0.117087	0.098703	0.077304	0.061957	0.053468	0.050838	0.052643	0.057596	0.064861

Bank of England spot curve data. Source: Bank of England.

The next step is to standardize the data into z-scores, assuming a mean of 0 and a variance of 1. We can do this by using the formula:

$$z = \frac{x - u}{\sigma}$$

We then form a covariance matrix from the standardized data using numpy. Although correlation matrices are more commonly used in finance, due to our data being standardized, a correlation matrix and covariance matrix *will actually yield the same result*. This is because a correlation matrix is simply a standardized covariance matrix.

$$\rho(x, y) = \frac{Cov(x, y)}{\sigma(x) * \sigma(y)}$$

We can then use the linalg.eig package from numpy to create the eigenvalues and eigenvectors from the matrix. This performs eigendecomposition on our standardized data. Eigenvalues are scalars of the linear transformation that has been applied in np.linalg.eig. We can use eigenvalues to find the proportion of the total variance that each principal component explains using this formula:

$$\lambda_i = \frac{\lambda i}{\lambda_1 + \lambda_2 + ...\lambda_n}$$

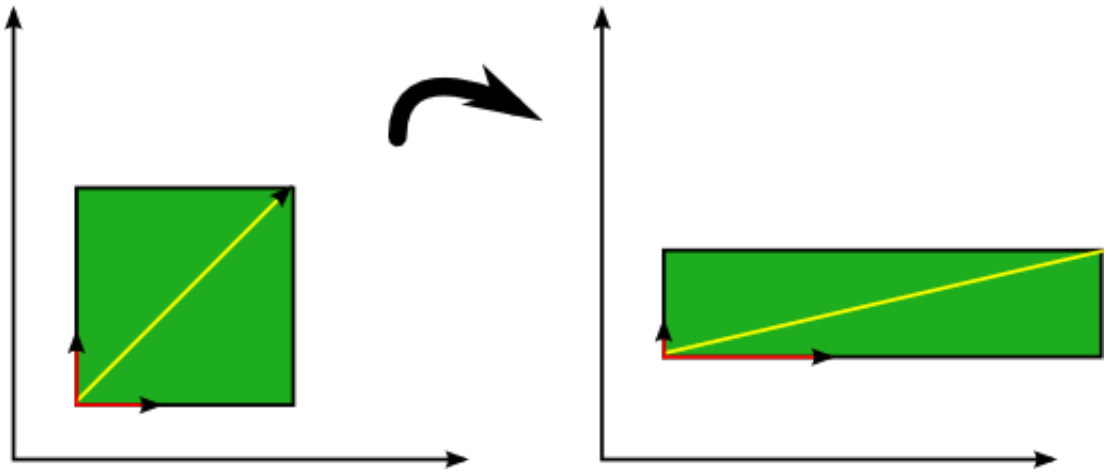
```
# Perform eigendecomposition


eigenvalues, eigenvectors =
np.linalg.eig(corr_matrix_array)


# Put data into a DataFrame
df_eigval = pd.DataFrame({"Eigenvalues":eigenvalues},
index=range(1,21))

# Work out explained proportion
df_eigval["Explained proportion"] =
df_eigval["Eigenvalues"] / np.sum(df_eigval["Eigenvalues"])


#Format as percentage
df_eigval.style.format({"Explained proportion": "{:.2%}"})
```

	Eigenvalues	Explained proportion
1	19.6608	98.30%
2	0.309852	1.55%
3	0.0248899	0.12%
4	0.00349701	0.02%
5	0.000805609	0.00%
6	0.000102751	0.00%
7	8.59904e-06	0.00%
8	1.20913e-06	0.00%
9	9.34863e-08	0.00%
10	1.4878e-08	0.00%

11	2.83478e-09	0.00%
12	5.90107e-10	0.00%
13	1.3502e-10	0.00%
14	3.38837e-11	0.00%
15	7.75823e-12	0.00%
16	2.55909e-12	0.00%
17	1.11736e-12	0.00%
18	2.91278e-13	0.00%
19	6.58629e-14	0.00%
20	1.03114e-14	0.00%

DataFrame of the eigenvalues. One eigenvalue for each principal component.

Eigenvectors are the coefficients of these linear transformations, leaving the direction unchanged. This diagram and equation should help to explain the concept visually:



The object gets transformed, however, the direction of the vectors stays the same. Source: Vision Dummy

$$Av = \lambda v$$

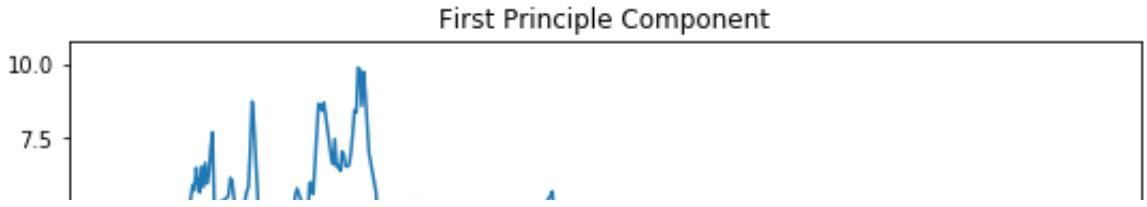
To form a time series for the principal components, we simply need to calculate the dot product between the eigenvectors and the standardized data.

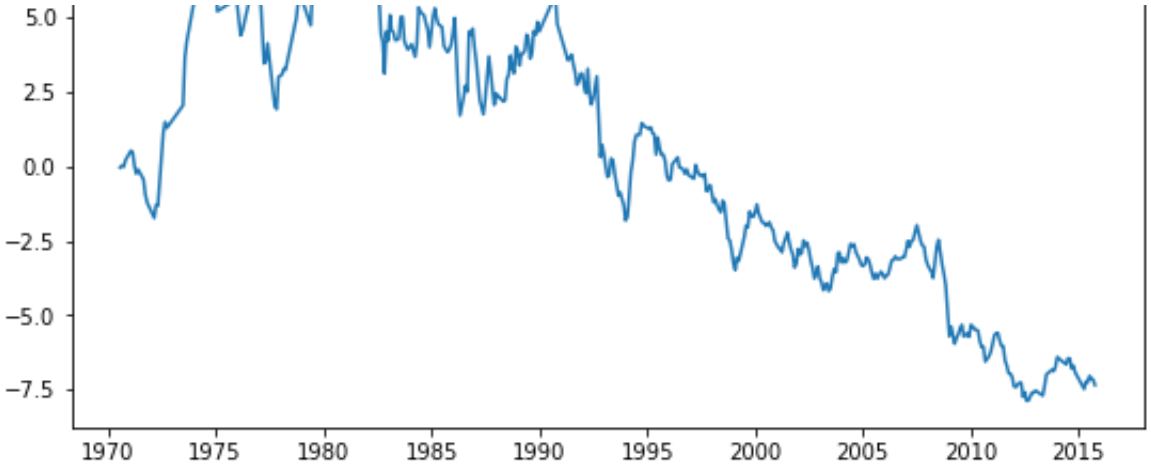
```
principal_components = df_std.dot(eigenvectors)
principal_components.head()
```

years:	0	1	2	3	4	5	6	7	8	9	10	
1970-07-31	-0.032838	-0.204918	0.067482	-0.146593	-0.004550	-0.021740	0.001970	0.000828	0.000339	0.000108	0.000050	-
1970-08-31	0.031877	-0.126833	0.062923	-0.105320	-0.031663	-0.008182	0.003184	0.001163	0.000124	-0.000057	0.000016	-
1970-09-30	-0.009330	-0.150376	0.078643	-0.089315	-0.024882	-0.010703	0.002030	0.001370	0.000030	-0.000024	0.000017	-
1970-10-31	0.219963	-0.213589	0.048486	-0.144802	-0.037515	-0.004840	0.004415	0.001125	0.000176	-0.000110	-0.000008	-
1971-01-31	0.533525	0.219927	0.126723	-0.031135	-0.011024	-0.010590	0.002907	0.000231	-0.000051	-0.000002	0.000012	

Extract of the principal components time series

When we plot the first principal component, we can see that it looks very similar to the actual 10-year yield curve. This makes sense, as according to our eigenvalues, the first principal component explains 98% of the data.





The second principal component represents the slope — this should have a correlation with the slope of the actual yield curve. One way we can calculate the slope is the 10-year spot minus the 2-year spot rate.

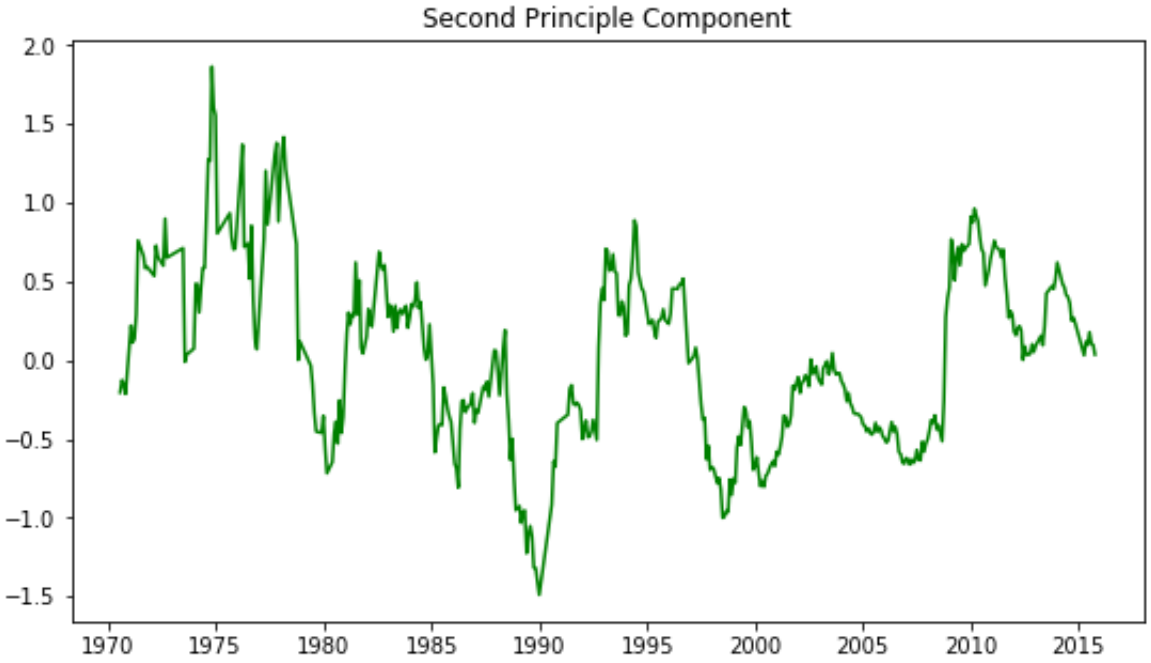
```
df_s = pd.DataFrame(data = df)
df_s = df_s[[2,10]]
df_s["slope"] = df_s[10] - df_s[2]
df_s.head()
```

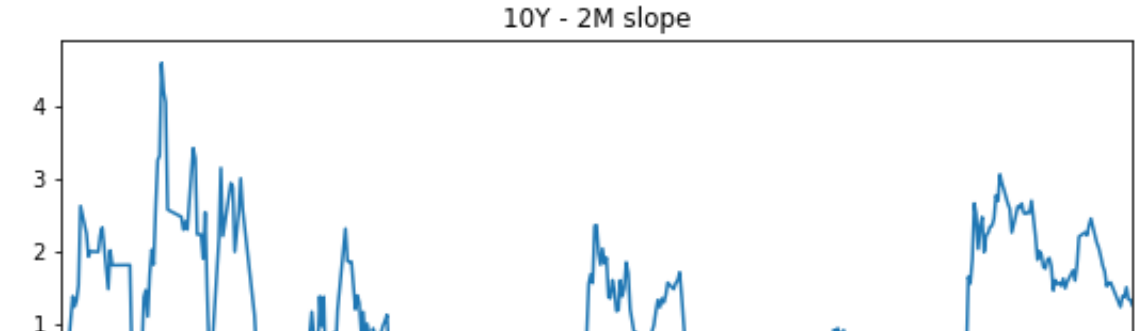
years:	2.0	10.0	slope
1970-07-31	7.106046	7.559915	0.453869
1970-08-31	7.063535	7.678102	0.614567
1970-09-30	7.018305	7.614256	0.595951

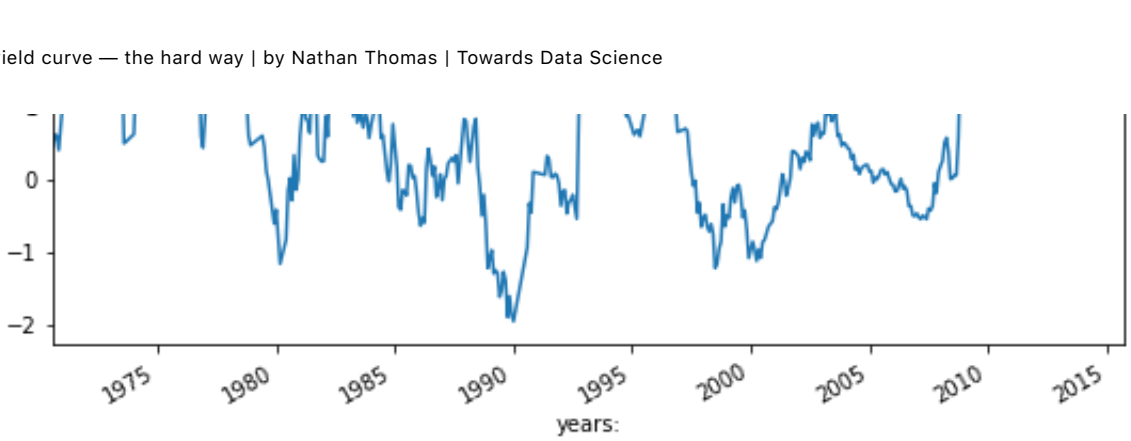
1970-10-31	7.364677	7.764320	0.399643
1971-01-31	7.035600	8.408105	1.372505

Spot rates, where slope = 10-year minus 2-year spot rates.

Simply from visual inspection, we can see that the slope looks almost identical to our second principal component. Let's verify with a correlation:








```
np.corrcoef(principal_components[1], df_s["slope"])

array([[1.        , 0.95856134],
       [0.95856134, 1.        ]])
```

When running the correlation between the second principal component and the 10Y-2M slope of the yield curve, the high correlation of 0.96 shows us that the second principal component does, in fact, represent the slope!

Thanks for reading! *Please feel free to leave any comments for any insights you may have. The full Jupyter Notebook which contains the source code I used to do this project can be found on my Github repository.*

References:

[1] Alexander, Carol, (2008). "*Market Risk Analysis II, Practical Financial Econometrics*".

[2] Adongo, Felix Atanga et al., 2018. *Principal Component and Factor Analysis of Macroeconomic Indicators.* Available at: https://pdfs.semanticscholar.org/8736/5855217edbc53e5e29c5c5872db7efb907cc.pdf

Disclaimer: All views expressed in this article are my own, and are ***not*** *in any way associated with Vanguard or any other financial entity. I am not a trader and am not making any money from the methods used in this article. This is not financial advice.*