Python Labs

Yield Curve PCA Decomposition

Kannan Singaravelu
kannan.singaravelu@fitchlearning.com

Dimensionality Reduction

One of the main difficulties in today's environment is being able to visualize data easily. There is too much information, too much news, and too much data. Dimensionality is the number of dimensions, features or input variables associated in a dataset and dimensionality reduction means reducing the number of features in a dataset.

Dimensionality reduction algorithms project high-dimensional data to a low-dimensional space while retaining as much of the variation as possible. There are two main approaches to dimensionality reduction.

- The first one is known as linear projection which involves linearly projecting data from a high-dimensional space to a low-dimensional space. This includes techniques such as principal component analysis (PCA).
- The second approach is known as manifold learning which is also referred to as nonlinear dimensionality reduction. This includes techniques such as Uniform manifold approximation and projection (UMAP).

Dimensionality reduction techniques help to address the curse of dimensionality.

Principal Component

PCA is a linear dimensionality reduction techniqu where the algorithm finds a low-dimensional representation of the data while retaining as much of the variation as possible and help reduce the complexity.

The main concept behind the PCA is to consider the correlation among features. If the correlation is very high among a subset of the features, PCA will attempt to combine the highly correlated features and represent this data with a smaller number of linearly uncorrelated features. The algorithm keeps performing this correlation reduction, finding the directions of maximum variance in the original high-dimensional data and projecting them onto a smaller dimensional space. These newly derived components are known as **principal components**.

Investors often refer to movements in the yield curve in terms of three driving factors:

- Level

- Slope
- Curvature

PCA formalizes this viewpoint and allows us to evaluate when a segment of the yield curve has cheapened or richened beyond that prescribed by recent yield movements. The essence of PCA in the context of rates market is that most yield curve movements can be represented as a set of two to three independent driving factors – the principal components (PCs) – along with their relative weightings. And, with these components, it is possible to reconstruct the original features.

We'll apply PCA to the set of yield curves fitted using the HJM model as discussed during the lecture. The PCs are ordered so that the first PC is the most important in capturing variability in the yield curves, the second PC is next most important, and so on.

The most intuitive way of obtaining PCs is via eigenvalue decomposition of a covariance matrix. The covariance measures the central tendency and talks about deviation from the mean. Intuitively, PCs represent ways in which the forward rates making up a yield curve can deviate from their mean levels.

Load Libraries

In [1]:
```
# Import libraries
import numpy as np
import pandas as pd

# Plot settings
import cufflinks as cf
cf.set_config_file(offline=True)

# scikit
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

In [2]:
```
pd.set_option('display.max_rows', 5000)
pd.set_option('display.max_columns', 100)
pd.set_option('display.width', 1000)
```

Read Data

In [3]:
```
data = pd.read_csv('../data/hjm-pca.txt', index_col=0, sep ='\t')
```

In [4]:
```
data.head()
```

Out[4]:

	0.08	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	5.5	6.0	6.5	7.0	7.5
1	5.77	6.44	6.71	6.65	6.50	6.33	6.15	5.99	5.84	5.71	5.57	5.44	5.30	5.16	5.01	4.86
2	5.77	6.45	6.75	6.68	6.54	6.39	6.23	6.08	5.95	5.82	5.69	5.56	5.43	5.28	5.13	4.97
3	5.78	6.44	6.74	6.68	6.56	6.41	6.26	6.12	5.98	5.84	5.71	5.57	5.43	5.28	5.12	4.96

	0.08	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	5.5	6.0	6.5	7.0	7.5
4	5.74	6.41	6.69	6.62	6.49	6.35	6.20	6.06	5.93	5.79	5.66	5.52	5.38	5.23	5.07	4.91
5	5.74	6.40	6.64	6.55	6.42	6.27	6.13	5.98	5.85	5.72	5.58	5.44	5.30	5.15	5.00	4.83

In [5]:
```
data.shape
```

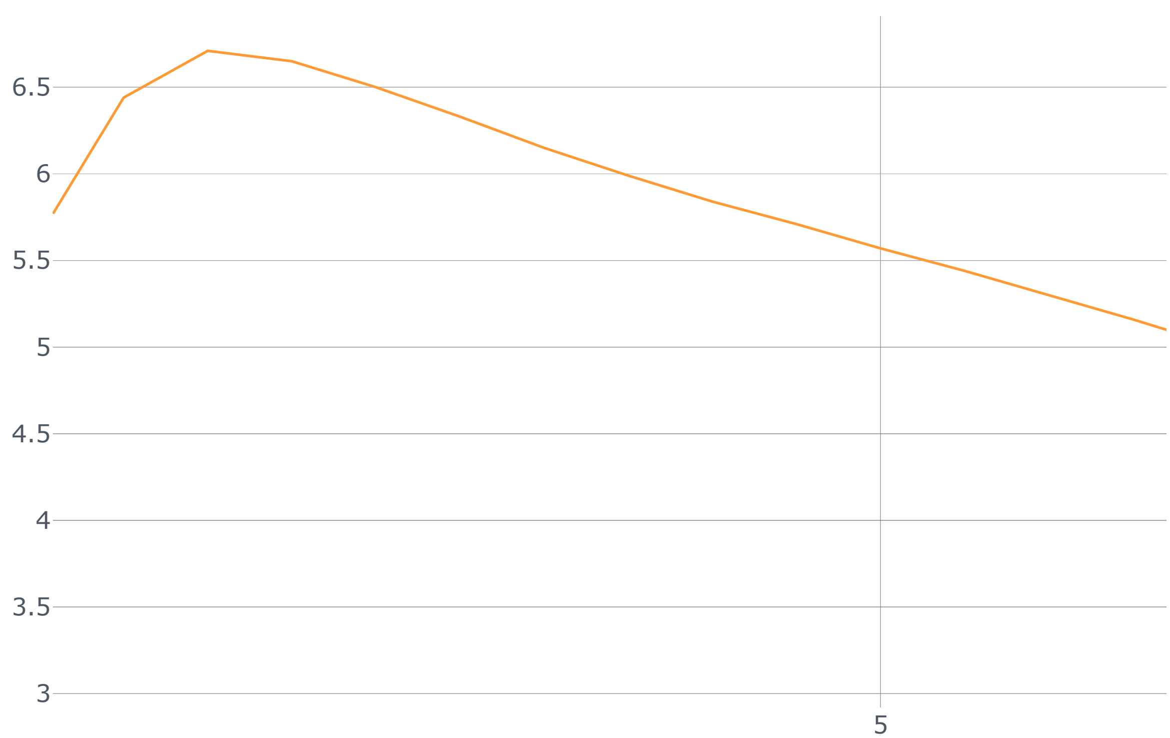
Out[5]:
```
(1264, 51)
```

Representation of a yield curve as 50 forward rates. As the yield curve evolves over time, each forward rate can change. It is understood that adjacent points on the yield curve do not move independently. PCA is a method for identifying the dominant ways in which various points on the yield curve move together.

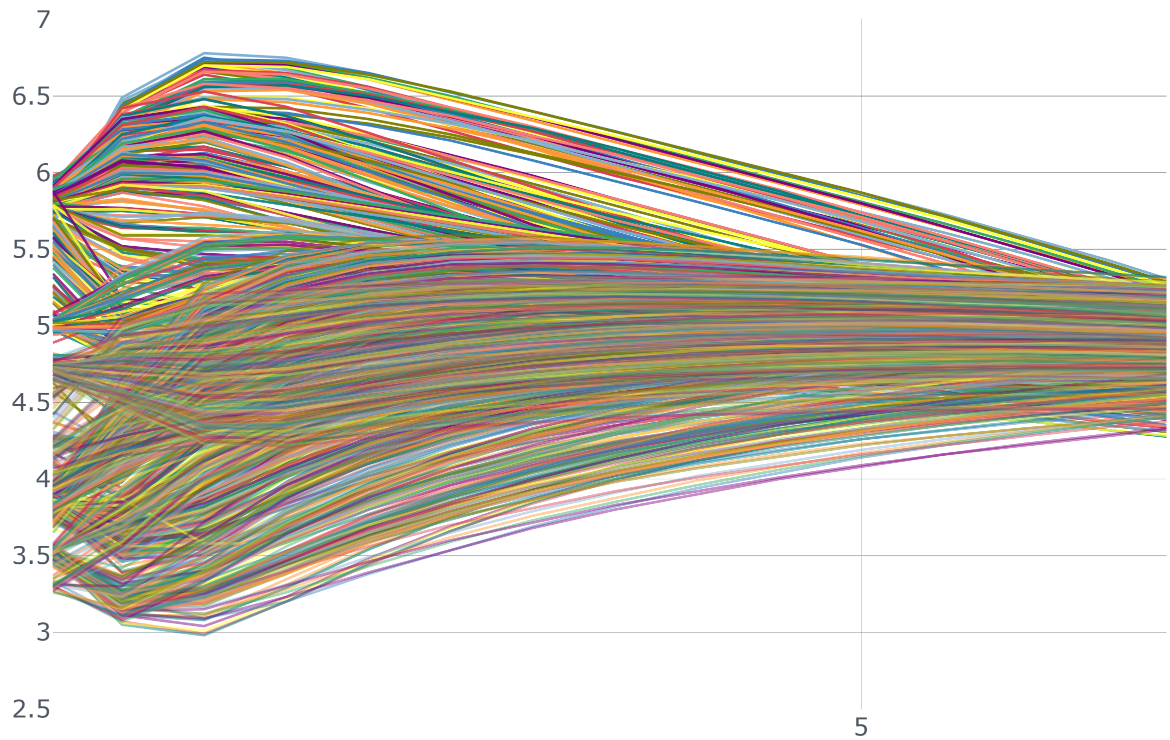
PCA allows us to take a set of yield curves, process them using standard mathematical methods, and then define a reduced form model for the yield curve. This reduced form model retains only a small number of principal components (PCs) but can reproduce the vast majority of yield curves that the full structural model could. This reduced model has fewer sources of uncertainty (i.e. dimensions) than if the 50 points of the yield curve were modelled independently.

Plot Curves

In [6]:
```
# Plot curve
data.iloc[0].iplot(title = 'Representation of a Yield Curve')
```



In [7]:
```
# Plot all curves
data.T.iplot(title='Daily Yield Curves')
```



We'll now produce the volatility chart by taking the first difference (scaling) and calculating historical variance by each individual maturity.

In [8]:
```
diff_ = data.diff(-1)
diff_.dropna(inplace=True)
```

In [9]:
```
diff_.tail()
```

Out[9]:

	0.08	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	5.5	6.0	6.5
1259	0.00	0.03	0.04	0.03	0.02	0.02	0.01	0.01	0.00	0.00	0.00	0.00	-0.01	0.00
1260	0.02	0.01	0.00	0.00	0.00	-0.01	-0.01	-0.01	0.00	-0.01	-0.01	-0.01	0.00	0.00
1261	-0.01	-0.03	-0.08	-0.12	-0.13	-0.13	-0.13	-0.13	-0.14	-0.13	-0.14	-0.14	-0.14	-0.14
1262	0.00	0.00	0.01	0.02	0.01	0.01	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00

	0.08	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	5.5	6.0	6.5
1263	0.02	0.00	0.03	0.03	0.04	0.04	0.05	0.06	0.06	0.06	0.07	0.07	0.06	0.05

In [10]:
```
diff_.shape
```

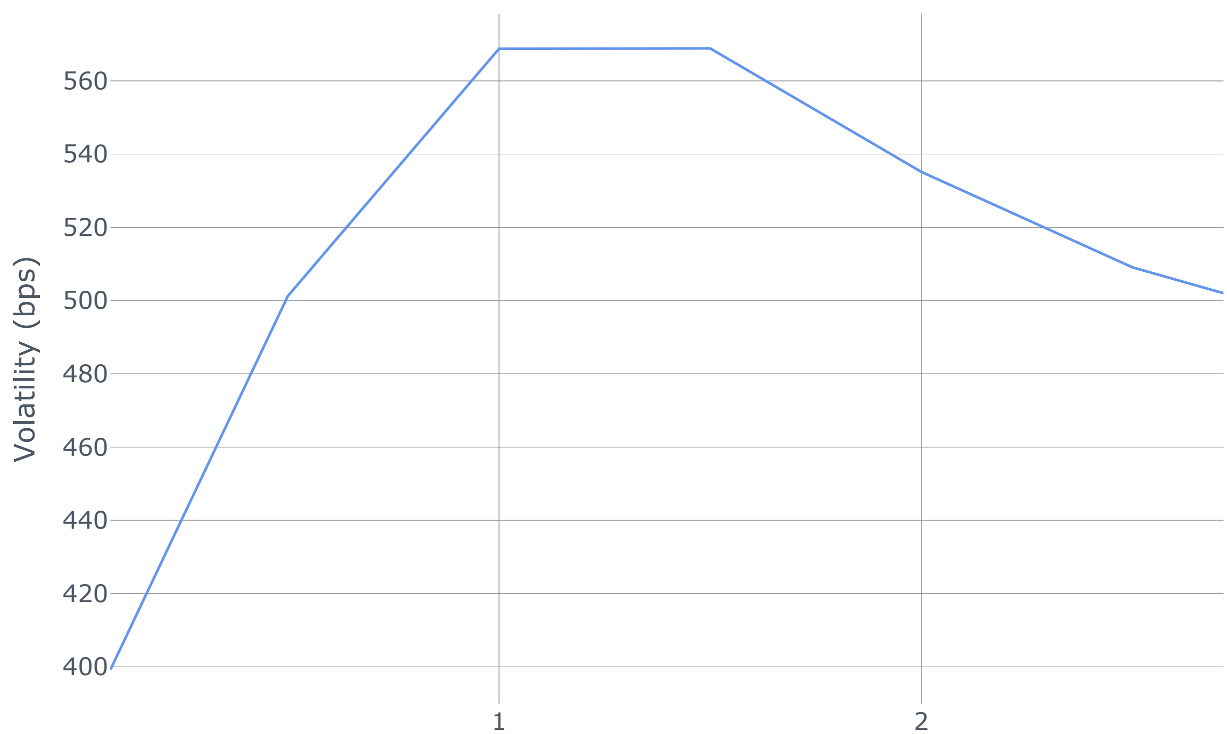
Out[10]:
```
(1263, 51)
```

Derive Volatility

The drift of forward rate is fully determined by volatility of forward rate dynamics.

In [11]:
```
vol = np.std(diff_, axis=0) * 10000
```

In [12]:
```
vol[:21].iplot(title='Volatility of daily UK government yields', xTitle='Tenor',
        color='cornflowerblue')
```



The above volatility plot is of the averaged values, but we can see that different parts of the yield curve move differently. As you can see volatility is very significant, especially at the shorter end of the curve. This means that 1-year and 2-year rates seems to move up and down a lot as compared to other tenors.

It is never all up or all down and PCA help us figure out exactly what is going. Covariance of daily changes shows dependency of different rates. Principal components can be calculated by finding the eigenvalues and eigenvectors of this covariance matrix of below.

Calculate Covariance

In [13]:
```
cov_= pd.DataFrame(np.cov(diff_, rowvar=False)*252/10000, columns=diff_.columns,
cov_.style.format("{:.4%}")
```

Out[13]:

	0.08	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
0.08	0.0040%	0.0009%	0.0002%	-0.0001%	-0.0001%	-0.0000%	0.0001%	0.0001%	0.0002%
0.5	0.0009%	0.0063%	0.0055%	0.0041%	0.0035%	0.0033%	0.0031%	0.0029%	0.0028%
1.0	0.0002%	0.0055%	0.0082%	0.0077%	0.0068%	0.0061%	0.0056%	0.0052%	0.0048%
1.5	-0.0001%	0.0041%	0.0077%	0.0082%	0.0075%	0.0069%	0.0063%	0.0058%	0.0055%
2.0	-0.0001%	0.0035%	0.0068%	0.0075%	0.0072%	0.0067%	0.0063%	0.0059%	0.0056%
2.5	-0.0000%	0.0033%	0.0061%	0.0069%	0.0067%	0.0065%	0.0062%	0.0060%	0.0058%
3.0	0.0001%	0.0031%	0.0056%	0.0063%	0.0063%	0.0062%	0.0061%	0.0060%	0.0058%
3.5	0.0001%	0.0029%	0.0052%	0.0058%	0.0059%	0.0060%	0.0060%	0.0060%	0.0059%
4.0	0.0002%	0.0028%	0.0048%	0.0055%	0.0056%	0.0058%	0.0058%	0.0059%	0.0059%
4.5	0.0002%	0.0027%	0.0045%	0.0051%	0.0054%	0.0055%	0.0057%	0.0058%	0.0059%
5.0	0.0002%	0.0026%	0.0042%	0.0049%	0.0051%	0.0054%	0.0056%	0.0058%	0.0059%
5.5	0.0002%	0.0025%	0.0040%	0.0046%	0.0049%	0.0052%	0.0054%	0.0057%	0.0058%
6.0	0.0002%	0.0024%	0.0038%	0.0044%	0.0047%	0.0051%	0.0053%	0.0056%	0.0058%
6.5	0.0002%	0.0022%	0.0036%	0.0042%	0.0046%	0.0049%	0.0052%	0.0055%	0.0057%
7.0	0.0002%	0.0021%	0.0035%	0.0041%	0.0044%	0.0048%	0.0051%	0.0054%	0.0056%
7.5	0.0002%	0.0020%	0.0033%	0.0039%	0.0043%	0.0046%	0.0049%	0.0052%	0.0055%
8.0	0.0002%	0.0019%	0.0032%	0.0038%	0.0041%	0.0044%	0.0047%	0.0050%	0.0053%
8.5	0.0002%	0.0018%	0.0031%	0.0036%	0.0039%	0.0042%	0.0045%	0.0048%	0.0051%
9.0	0.0002%	0.0017%	0.0029%	0.0035%	0.0038%	0.0041%	0.0043%	0.0046%	0.0049%
9.5	0.0002%	0.0016%	0.0028%	0.0034%	0.0036%	0.0039%	0.0042%	0.0044%	0.0047%
10.0	0.0001%	0.0015%	0.0027%	0.0032%	0.0035%	0.0037%	0.0040%	0.0042%	0.0044%
10.5	0.0001%	0.0014%	0.0026%	0.0031%	0.0033%	0.0035%	0.0037%	0.0040%	0.0042%
11.0	0.0001%	0.0013%	0.0025%	0.0029%	0.0031%	0.0034%	0.0036%	0.0038%	0.0040%
11.5	0.0001%	0.0012%	0.0023%	0.0028%	0.0030%	0.0032%	0.0033%	0.0035%	0.0037%
12.0	0.0001%	0.0011%	0.0022%	0.0027%	0.0029%	0.0030%	0.0032%	0.0033%	0.0035%
12.5	0.0001%	0.0011%	0.0021%	0.0026%	0.0027%	0.0029%	0.0030%	0.0032%	0.0033%
13.0	0.0001%	0.0010%	0.0020%	0.0025%	0.0026%	0.0028%	0.0029%	0.0030%	0.0031%
13.5	0.0001%	0.0009%	0.0020%	0.0024%	0.0025%	0.0027%	0.0028%	0.0029%	0.0030%
14.0	0.0001%	0.0009%	0.0019%	0.0023%	0.0024%	0.0026%	0.0026%	0.0028%	0.0029%

	0.08	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
14.5	0.0001%	0.0008%	0.0018%	0.0022%	0.0023%	0.0025%	0.0025%	0.0026%	0.0027%
15.0	0.0001%	0.0008%	0.0018%	0.0022%	0.0023%	0.0024%	0.0025%	0.0026%	0.0027%
15.5	0.0001%	0.0008%	0.0017%	0.0021%	0.0022%	0.0023%	0.0024%	0.0025%	0.0026%
16.0	0.0001%	0.0007%	0.0017%	0.0021%	0.0022%	0.0023%	0.0024%	0.0024%	0.0025%
16.5	0.0001%	0.0008%	0.0017%	0.0021%	0.0022%	0.0023%	0.0024%	0.0025%	0.0025%
17.0	0.0001%	0.0008%	0.0017%	0.0021%	0.0022%	0.0023%	0.0023%	0.0024%	0.0025%
17.5	0.0001%	0.0008%	0.0017%	0.0021%	0.0022%	0.0023%	0.0024%	0.0025%	0.0025%
18.0	0.0001%	0.0008%	0.0017%	0.0021%	0.0022%	0.0023%	0.0024%	0.0024%	0.0025%
18.5	0.0001%	0.0008%	0.0017%	0.0021%	0.0022%	0.0023%	0.0024%	0.0025%	0.0026%
19.0	0.0001%	0.0008%	0.0017%	0.0021%	0.0022%	0.0024%	0.0024%	0.0025%	0.0026%
19.5	0.0001%	0.0009%	0.0018%	0.0022%	0.0023%	0.0024%	0.0025%	0.0026%	0.0026%
20.0	0.0000%	0.0009%	0.0018%	0.0022%	0.0023%	0.0024%	0.0025%	0.0026%	0.0027%
20.5	0.0001%	0.0009%	0.0018%	0.0022%	0.0024%	0.0025%	0.0026%	0.0027%	0.0028%
21.0	0.0001%	0.0010%	0.0019%	0.0023%	0.0025%	0.0026%	0.0027%	0.0028%	0.0029%
21.5	0.0001%	0.0010%	0.0019%	0.0023%	0.0025%	0.0026%	0.0027%	0.0028%	0.0029%
22.0	0.0001%	0.0010%	0.0020%	0.0025%	0.0026%	0.0027%	0.0028%	0.0029%	0.0030%
22.5	0.0001%	0.0011%	0.0021%	0.0025%	0.0026%	0.0028%	0.0029%	0.0030%	0.0031%
23.0	0.0001%	0.0012%	0.0021%	0.0026%	0.0027%	0.0028%	0.0029%	0.0031%	0.0032%
23.5	0.0001%	0.0012%	0.0022%	0.0026%	0.0028%	0.0029%	0.0030%	0.0032%	0.0033%
24.0	0.0001%	0.0012%	0.0022%	0.0027%	0.0028%	0.0030%	0.0031%	0.0032%	0.0033%
24.5	0.0001%	0.0013%	0.0023%	0.0027%	0.0029%	0.0031%	0.0032%	0.0033%	0.0034%
25.0	0.0001%	0.0013%	0.0024%	0.0028%	0.0030%	0.0032%	0.0033%	0.0034%	0.0035%

Eigen Decomposition

In [14]:
```
# Perform eigen decomposition
eigenvalues, eigenvectors = np.linalg.eig(cov_)

# Sort values (good practice)
idx = eigenvalues.argsort()[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:,idx]

# Format into a DataFrame
df_eigval = pd.DataFrame({"Eigenvalues": eigenvalues})

eigenvalues
```

Out[14]:
```
array([2.02898049e-03, 4.63398406e-04, 1.63446845e-04, 8.51547101e-05,
       5.10538526e-05, 3.32765289e-05, 1.58231855e-05, 4.49832087e-06,
       1.94407432e-06, 8.99455051e-07, 6.04790270e-07, 5.90792253e-07,
       5.89198637e-07, 5.57023543e-07, 5.55577838e-07, 5.37017622e-07,
```

```
5.25225242e-07, 5.09484922e-07, 5.02130032e-07, 4.95037888e-07,
4.85536393e-07, 4.74757652e-07, 4.66830631e-07, 4.56358980e-07,
4.53910470e-07, 4.45678829e-07, 4.35704316e-07, 4.34084479e-07,
4.26484963e-07, 4.13347804e-07, 4.01916308e-07, 3.97702101e-07,
3.90292851e-07, 3.86498129e-07, 3.76760528e-07, 3.73179456e-07,
3.63351112e-07, 3.57997757e-07, 3.48773694e-07, 3.42142905e-07,
3.35540502e-07, 3.27434287e-07, 3.20549997e-07, 3.13802097e-07,
3.06870950e-07, 3.04664148e-07, 2.99586146e-07, 2.88553566e-07,
2.83944056e-07, 2.67537628e-07, 2.48780504e-07])
```

Explained Variance

In [15]:

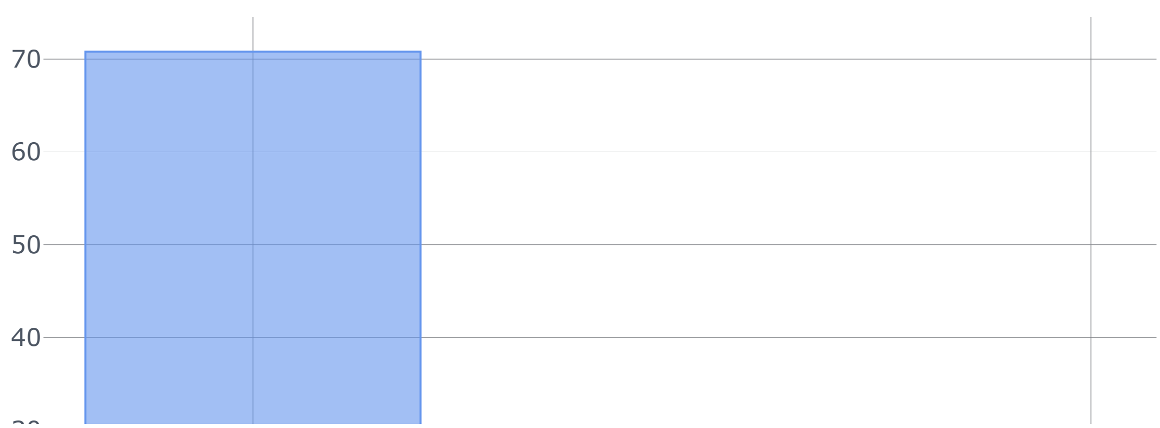
```
# Work out explained proportion
df_eigval["Explained proportion"] = df_eigval["Eigenvalues"] / np.sum(df_eigval[
df_eigval = df_eigval[:10]

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

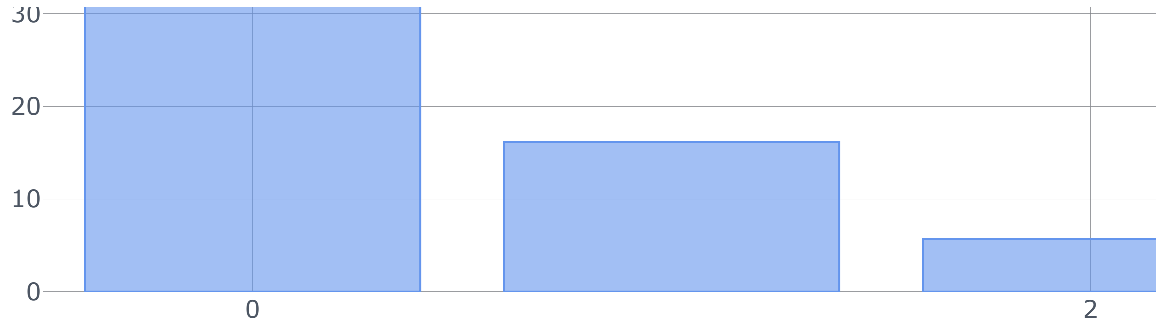
Out[15]:

	Eigenvalues	Explained proportion
0	0.002029	70.81%
1	0.000463	16.17%
2	0.000163	5.70%
3	0.000085	2.97%
4	0.000051	1.78%
5	0.000033	1.16%
6	0.000016	0.55%
7	0.000004	0.16%
8	0.000002	0.07%
9	0.000001	0.03%

In [16]:

```
(df_eigval['Explained proportion'][:10]*100).iplot(kind='bar',
                                                   title='Percentage of overall varian
                                                   color='cornflowerblue')
```





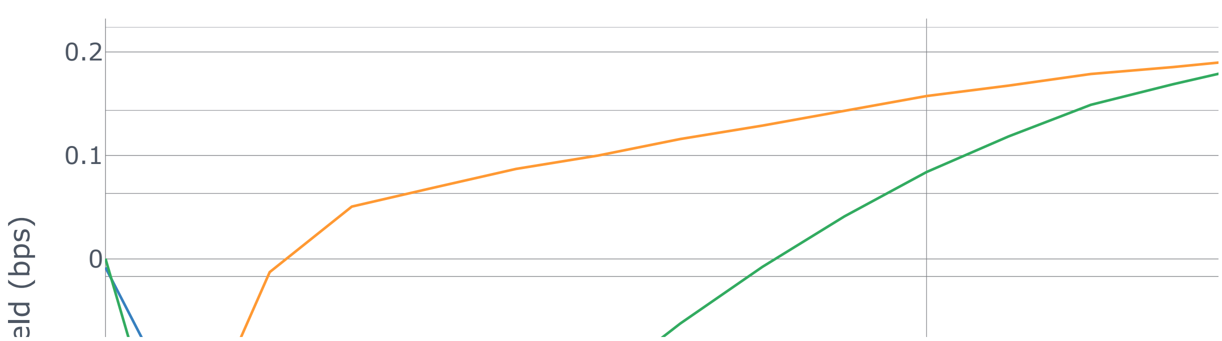
Visualize PCs

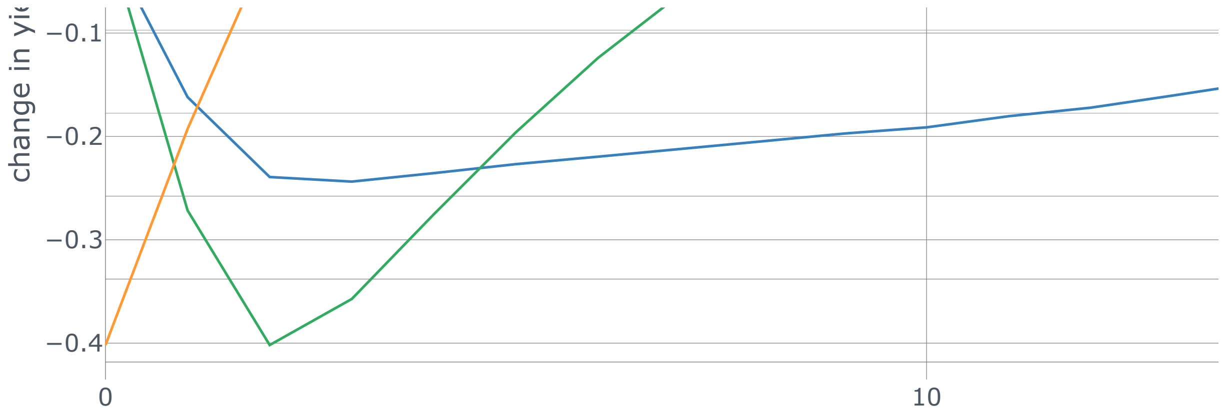
In [17]:
```
# Subsume first 3 components into a dataframe
pcadf = pd.DataFrame(eigenvectors[:,0:3], columns=['PC1','PC2','PC3'])
pcadf[:10]
```

Out[17]:

	PC1	PC2	PC3
0	0.004091	-0.008275	0.000235
1	0.056204	-0.161934	-0.271539
2	0.101034	-0.239236	-0.401805
3	0.116817	-0.243675	-0.357226
4	0.121388	-0.235475	-0.275176
5	0.125890	-0.226757	-0.195816
6	0.129107	-0.219537	-0.123907
7	0.133088	-0.211509	-0.062428
8	0.136317	-0.204675	-0.007698
9	0.139725	-0.197136	0.041132

In [18]:
```
pcadf.iplot(title='First Three Principal Components', secondary_y='PC1', seconda
            yTitle='change in yield (bps)')
```





One of the key interpretations of PCA as applied to interest rates are the components of the yield curve. We can attribute the first three principal components to

- Parallel shifts in yield curve (shifts across the entire yield curve)
- Changes in short/long rates (steepening/flattening of the curve)
- Changes in curvature of the model (twists)

The first PC represents the situation that all forward rates in the yield curve move in the same direction but points around the 15 year term move more than points at the shorter or longer parts of the yield curve. This corresponds to a general rise (or fall) of all of the forward rates in the yield curve, but cannot be called a uniform or parallel shift. The impact of the first PC can be easily observed amongst the yield curves as it contributes more than 71% of the variability.

The second PC represents situations in which the short end of the yield curve moves up at the same time as the long end moves down, or vice versa. This is often described as a tilt in the yield curve, although in practice there is more subtle definition to the shape. This reflects the particular yield curves that were used for the analysis, as well as the structural model and calibration that were used to create them. In this excample, the influence of the second PC accounts for about 16.27% of the variability in the yield curves.

The third PC is further interpreted as a higher order buckling in which the short end and long end move up at the same time as a region of medium term rates move down, or vice versa. In this particular example, this type of movement is only responsible for about 5.75% of the variability.

Having identified the most important factors, we can use their functional form to predict the most likely evolution of the yeild curve. Thus, a simple linear regression is fitted for the shift factor as it simply moves the curve up and down. Second degree polynomial is fitted for the tilt factor and higher degree can approximate flexing. Thus, yield curve can be approximated by linear combination of first three loadings.

UK Government Bond Rates

The purpose of applying PCA to financial markets is to explain the price changes of different assets through a smaller set of factors. This is achieved via the dimensionality reduction of the observations where we pick meaningful factors (among many) explaining the most of the price changes. We'll now apply the principal component analysis to UK government bond spot rates [1] from 0.5 years up to 10 years to maturity.

We'll adopt how two methods to decompose the yield curve: one using eigen decomposition as we above and another by applying direct functions from scikit-learn.

Read Data

In [19]:
```
# Import Bank of England spot curve data from excel
df = pd.read_excel("../data/GLC Nominal month end data_1970 to 2015.xlsx",
                   index_col=0, header=3, sheet_name="4. spot curve", skiprows=[

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

df.head()
```

Out[19]:

	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5
years:									
1970-01-31	NaN	8.635354	8.707430	8.700727	8.664049	8.618702	8.572477	8.528372	8.487617
1970-02-28	NaN	8.413131	8.397269	8.370748	8.337633	8.301590	8.265403	8.230804	8.198713
1970-03-31	NaN	7.744187	7.782761	7.795017	7.793104	7.784963	7.775288	7.766459	7.759564
1970-04-30	NaN	7.606512	7.864352	7.973522	8.002442	7.992813	7.967524	7.938335	7.911422
1970-05-31	NaN	7.391107	7.735838	7.862182	7.877510	7.840673	7.782249	7.718053	7.656856

Drop Missing Value

In [20]:
```
# Drop nan values
df = df.dropna(how="any")
df.shape
```

Out[20]: (456, 20)

Scale Data

In [21]:
```
# Standarized data
scaler = StandardScaler()
scaler.fit(df)

df1 = pd.DataFrame(scaler.transform(df))
df1.head()
```

Out[21]:

	0	1	2	3	4	5	6	7	8
0	0.046393	0.137629	0.114852	0.075536	0.040097	0.011058	-0.011958	-0.029713	-0.042998
1	0.052706	0.089935	0.084148	0.064262	0.042095	0.021662	0.004165	-0.010140	-0.021293
2	0.070345	0.094904	0.076732	0.052266	0.029015	0.008896	-0.007742	-0.021056	-0.031301
3	0.107418	0.167307	0.168964	0.144128	0.112896	0.082938	0.056761	0.035002	0.017677
4	0.066680	0.069013	0.061107	0.056853	0.057613	0.062163	0.069256	0.077907	0.087515

Covariance Matrix

In [22]:

```
# Create a covariance matrix
cov_matrix_array = np.cov(df1, rowvar=False)
pd.DataFrame(cov_matrix_array) #, index=range(1,21), columns=range(1,21))
```

Out[22]:

	0	1	2	3	4	5	6	7	8
0	1.002198	0.998543	0.992527	0.986072	0.979831	0.973998	0.968590	0.963540	0.958753
1	0.998543	1.002198	1.000387	0.996565	0.992087	0.987522	0.983078	0.978799	0.974651
2	0.992527	1.000387	1.002198	1.001082	0.998610	0.995567	0.992311	0.988984	0.985621
3	0.986072	0.996565	1.001082	1.002198	1.001480	0.999820	0.997672	0.995249	0.992637
4	0.979831	0.992087	0.998610	1.001480	1.002198	1.001708	1.000521	0.998899	0.996966
5	0.973998	0.987522	0.995567	0.999820	1.001708	1.002198	1.001839	1.000924	0.999604
6	0.968590	0.983078	0.992311	0.997672	1.000521	1.001839	1.002198	1.001913	1.001154
7	0.963540	0.978799	0.988984	0.995249	0.998899	1.000924	1.001913	1.002198	1.001957
8	0.958753	0.974651	0.985621	0.992637	0.996966	0.999604	1.001154	1.001957	1.002198
9	0.954135	0.970581	0.982212	0.989867	0.994781	0.997959	1.000013	1.001291	1.001983
10	0.949610	0.966533	0.978739	0.986948	0.992377	0.996036	0.998547	1.000264	1.001380
11	0.945122	0.962470	0.975185	0.983885	0.989774	0.993869	0.996800	0.998924	1.000437
12	0.940638	0.958369	0.971542	0.980687	0.986993	0.991487	0.994805	0.997310	0.999196
13	0.936137	0.954214	0.967809	0.977362	0.984054	0.988917	0.992596	0.995455	0.997692
14	0.931605	0.949998	0.963986	0.973920	0.980972	0.986181	0.990197	0.993389	0.995957
15	0.927030	0.945715	0.960075	0.970370	0.977763	0.983298	0.987632	0.991138	0.994018
16	0.922406	0.941361	0.956077	0.966718	0.974436	0.980282	0.984917	0.988720	0.991894
17	0.917724	0.936932	0.951991	0.962967	0.970999	0.977142	0.982064	0.986148	0.989601
18	0.912977	0.932423	0.947816	0.959119	0.967455	0.973884	0.979080	0.983432	0.987149
19	0.908156	0.927829	0.943548	0.955170	0.963803	0.970509	0.975969	0.980576	0.984544

Eigen Decomposition

In [23]:

```
# Perform eigen decomposition
```

```
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix_array)

# Sort values (good practice)
idx = eigenvalues.argsort()[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:,idx]

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

eigenvalues
```

Out[23]:

```
array([1.97040530e+01, 3.10533489e-01, 2.49445560e-02, 3.50469991e-03,
       8.07379451e-04, 1.02976761e-04, 8.61793663e-06, 1.21179203e-06,
       9.36917905e-08, 1.49106971e-08, 2.84101550e-09, 5.91403231e-10,
       1.35317307e-10, 3.39594794e-11, 7.77568486e-12, 2.56434189e-12,
       1.11994739e-12, 2.91388013e-13, 6.64015542e-14, 1.04185425e-14])
```

In [24]:

```
# Format into a DataFrame
df_eigvec = pd.DataFrame(eigenvectors) #, index=range(1,21))

eigenvectors[:,0]
```

Out[24]:

```
array([0.2163711 , 0.21967968, 0.22191335, 0.2233227 , 0.22419027,
       0.22472894, 0.22506385, 0.22526262, 0.22535956, 0.2253718 ,
       0.22530883, 0.22517738, 0.2249833 , 0.22473203, 0.22442859,
       0.2240774 , 0.22368207, 0.22324536, 0.22276915, 0.22225453])
```

In [25]:

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

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

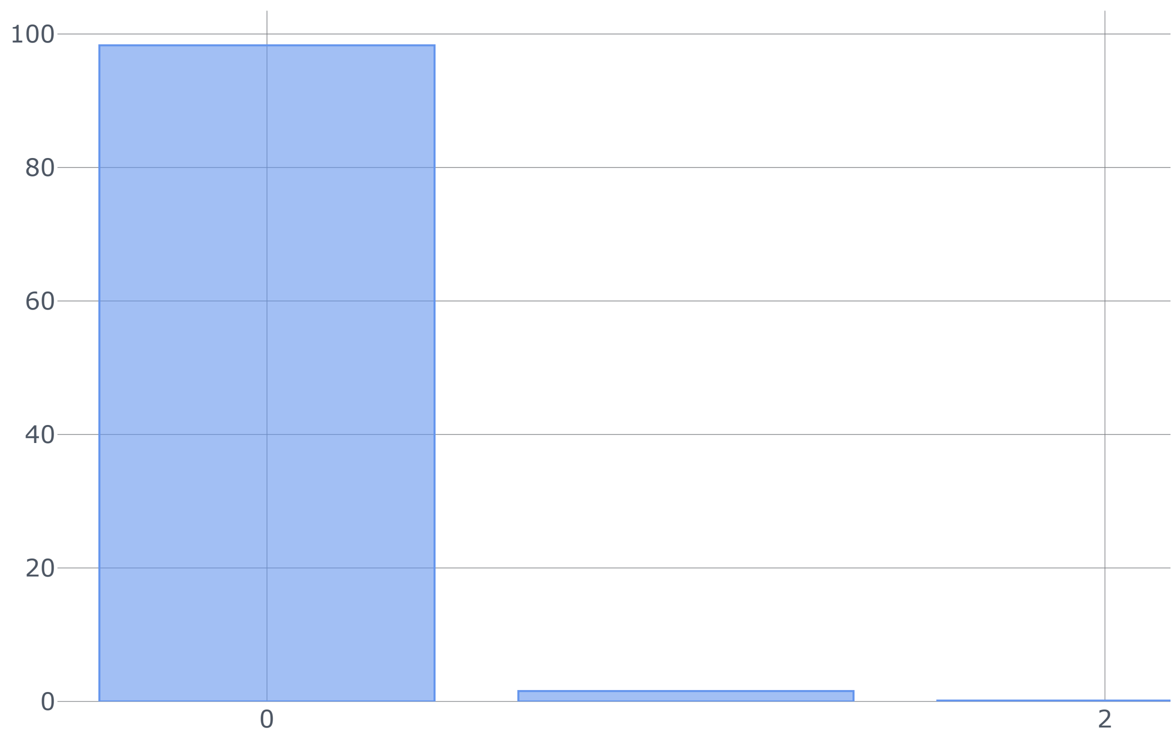
Out[25]:

	Eigenvalues	Explained proportion
0	19.704053	98.30%
1	0.310533	1.55%
2	0.024945	0.12%
3	0.003505	0.02%
4	0.000807	0.00%
5	0.000103	0.00%
6	0.000009	0.00%
7	0.000001	0.00%
8	0.000000	0.00%
9	0.000000	0.00%
10	0.000000	0.00%
11	0.000000	0.00%
12	0.000000	0.00%
13	0.000000	0.00%
14	0.000000	0.00%

	Eigenvalues	Explained proportion
15	0.000000	0.00%
16	0.000000	0.00%
17	0.000000	0.00%
18	0.000000	0.00%
19	0.000000	0.00%

In [26]:

```
(df_eigval['Explained proportion'][:10]*100).iplot(kind='bar',
                                             title='Percentage of overall varian
                                             color='cornflowerblue')
```



Visualize PCs

In [27]:

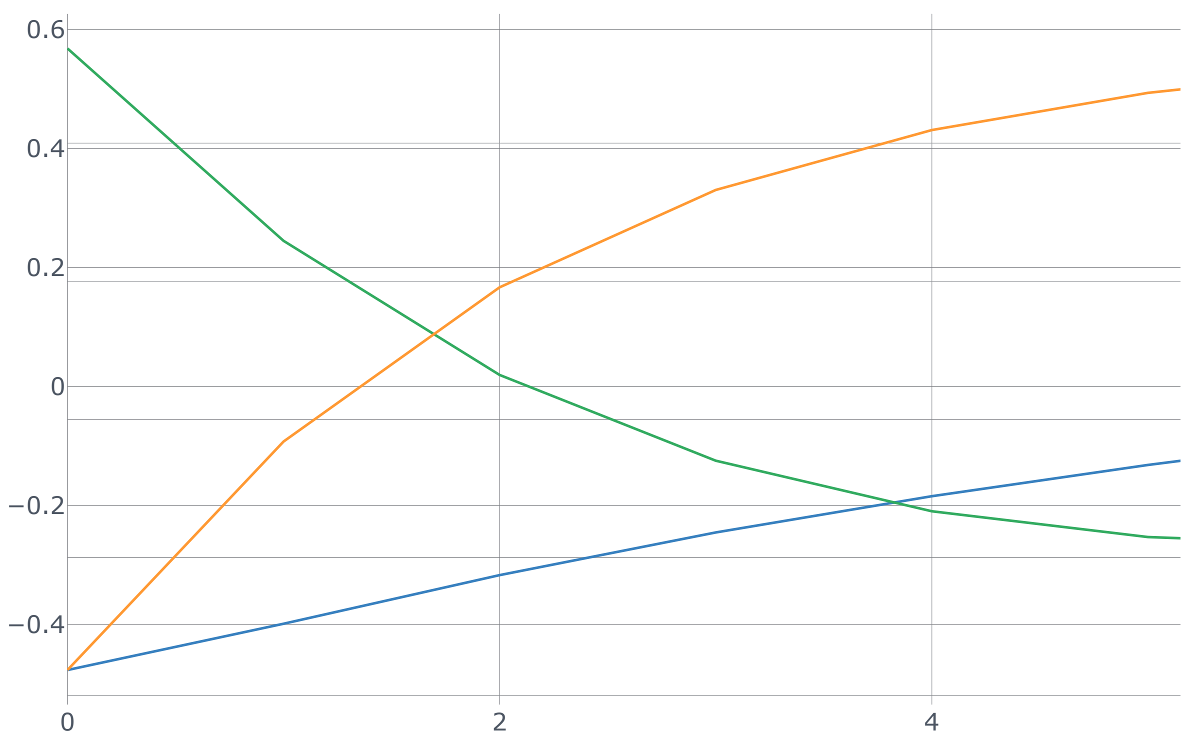
```
# Subsume first 3 components into a dataframe
pcdf = pd.DataFrame(eigenvectors[:,0:3], columns=['PC1','PC2','PC3'])
pcdf[:10]
```

Out[27]:

	PC1	PC2	PC3
0	0.216371	-0.476742	0.568237

	PC1	PC2	PC3
1	0.219680	-0.398921	0.244808
2	0.221913	-0.317188	0.019318
3	0.223323	-0.245476	-0.124801
4	0.224190	-0.184527	-0.209822
5	0.224729	-0.131987	-0.253173
6	0.225064	-0.085706	-0.266431
7	0.225263	-0.044103	-0.257638
8	0.225360	-0.006087	-0.232802
9	0.225372	0.029085	-0.196639

In [28]:
```
pcdf.iplot(title='First Three Principal Components', secondary_y='PC1', secondar
```



PCA Decomposition using Sklearn

In [29]:
```
# Scale and fit the model
pipe = Pipeline([("scaler", StandardScaler()), ("pca", PCA())])
pipe.fit(df)
```

Out[29]:
```
Pipeline(steps=[('scaler', StandardScaler()), ('pca', PCA())])
```

In [30]:
```
# eigenvectors
pipe['pca'].components_[0]
```

Out[30]:
```
array([0.2163711 , 0.21967968, 0.22191335, 0.2233227 , 0.22419027,
       0.22472894, 0.22506385, 0.22526262, 0.22535956, 0.2253718 ,
       0.22530883, 0.22517738, 0.2249833 , 0.22473203, 0.22442859,
       0.2240774 , 0.22368207, 0.22324536, 0.22276915, 0.22225453])
```

In [31]:
```
# eigen values
pipe['pca'].explained_variance_
```

Out[31]:
```
array([1.97040530e+01, 3.10533489e-01, 2.49445560e-02, 3.50469991e-03,
       8.07379451e-04, 1.02976761e-04, 8.61793663e-06, 1.21179203e-06,
       9.36917912e-08, 1.49106974e-08, 2.84101504e-09, 5.91403577e-10,
       1.35317277e-10, 3.39584209e-11, 7.77565136e-12, 2.56435621e-12,
       1.12010259e-12, 2.91862679e-13, 6.62857318e-14, 1.05315377e-14])
```

In [32]:
```
# eigen values proportion
pipe['pca'].explained_variance_ratio_
```

Out[32]:
```
array([9.83042118e-01, 1.54926247e-02, 1.24449265e-03, 1.74850708e-04,
       4.02804441e-05, 5.13754673e-06, 4.29951882e-07, 6.04567298e-08,
       4.67431634e-09, 7.43899926e-10, 1.41739237e-10, 2.95053319e-11,
       6.75102643e-12, 1.69419753e-12, 3.87929975e-13, 1.27936631e-13,
       5.58823112e-14, 1.45611315e-14, 3.30701842e-15, 5.25422109e-16])
```

In [33]:
```
df2 = pd.DataFrame({'Eigenvalues': pipe['pca'].explained_variance_,
                    'Explained proportion': pipe['pca'].explained_variance_ratio
#Format as percentage
df2.style.format({"Explained proportion": "{:.2%}"})
```

Out[33]:

	Eigenvalues	Explained proportion
0	19.704053	98.30%
1	0.310533	1.55%
2	0.024945	0.12%
3	0.003505	0.02%
4	0.000807	0.00%
5	0.000103	0.00%
6	0.000009	0.00%
7	0.000001	0.00%
8	0.000000	0.00%
9	0.000000	0.00%
10	0.000000	0.00%
11	0.000000	0.00%
12	0.000000	0.00%
13	0.000000	0.00%
14	0.000000	0.00%

	Eigenvalues	Explained proportion
15	0.000000	0.00%
16	0.000000	0.00%
17	0.000000	0.00%
18	0.000000	0.00%
19	0.000000	0.00%

PCA Projections

In [34]:

```
# Calculate principal components
principal_components = df1.dot(eigenvectors)
principal_components.index = df.index
principal_components.head()
```

Out[34]:

	0	1	2	3	4	5	6	7
years:								
1970-07-31	-0.032874	-0.205143	0.067556	-0.146754	-0.004555	-0.021764	-0.001972	0.000829
1970-08-31	0.031912	-0.126972	0.062992	-0.105436	-0.031698	-0.008191	-0.003187	0.001165
1970-09-30	-0.009340	-0.150541	0.078729	-0.089413	-0.024909	-0.010715	-0.002032	0.001371
1970-10-31	0.220205	-0.213823	0.048540	-0.144961	-0.037556	-0.004846	-0.004420	0.001127
1971-01-31	0.534111	0.220168	0.126862	-0.031169	-0.011036	-0.010601	-0.002910	0.000232

In [35]:

```
principal_components.shape
```

Out[35]: (456, 20)

PC1 : Yield

In [36]:

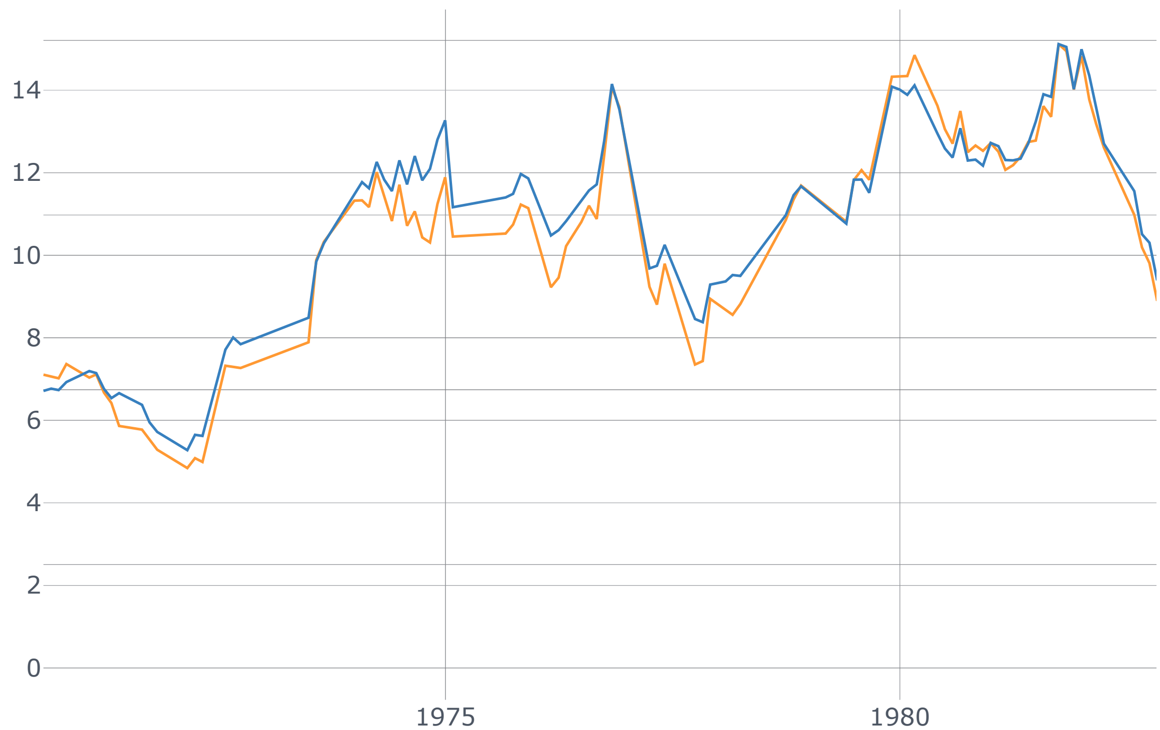
```
level = pd.DataFrame({'10Y': df[2.0],
                      'PC1': principal_components[0]})
level.head()
```

Out[36]:

	10Y	PC1
years:		
1970-07-31	7.106046	-0.032874
1970-08-31	7.063535	0.031912
1970-09-30	7.018305	-0.009340
1970-10-31	7.364677	0.220205

	10Y	PC1
years:		
1971-01-31	7.035600	0.534111

In [37]:
```
level.iplot(title='PC1 : Yield', secondary_y='PC1')
```



PC2 : Slope

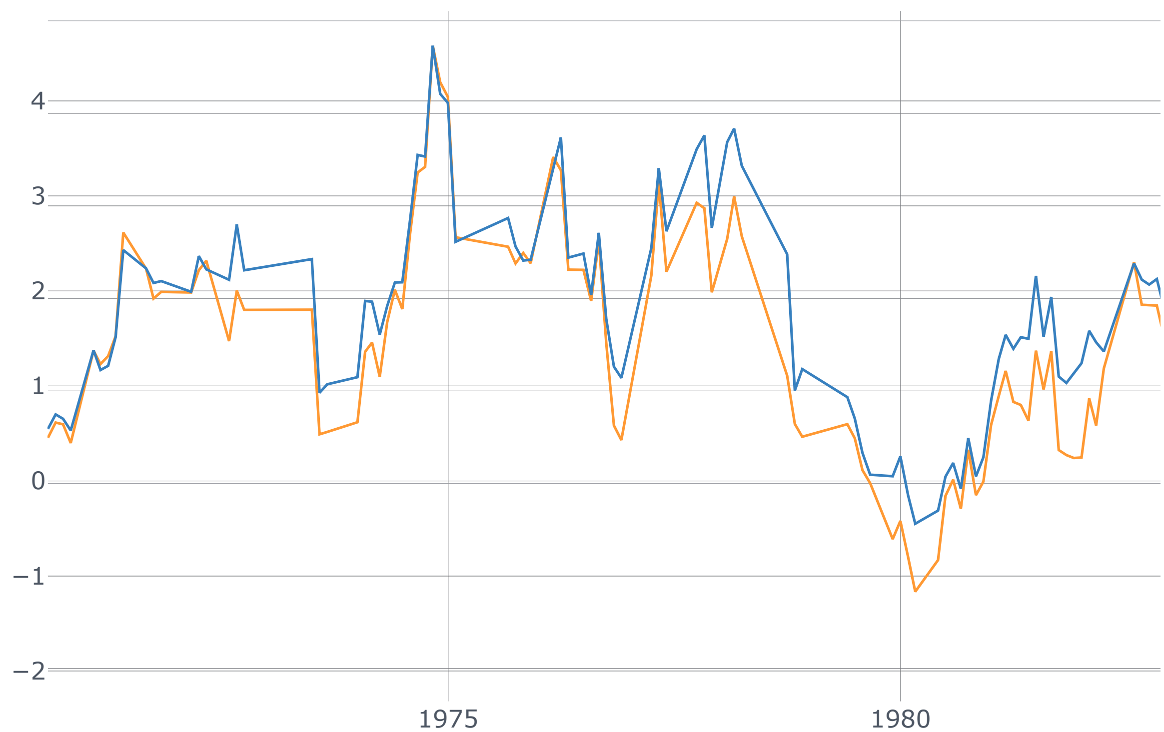
In [38]:
```
# Calculate 10Y-2M slope
slope = pd.DataFrame(df)
slope = slope[[2,10]]
slope['slope'] = slope[10] - slope[2]
slope['PC2'] = principal_components[1]
slope.head()
```

Out[38]:

	2.0	10.0	slope	PC2
years:				
1970-07-31	7.106046	7.559915	0.453869	-0.205143
1970-08-31	7.063535	7.678102	0.614567	-0.126972
1970-09-30	7.018305	7.614256	0.595951	-0.150541

	2.0	10.0	slope	PC2
years:				
1970-10-31	7.364677	7.764320	0.399643	-0.213823
1971-01-31	7.035600	8.408105	1.372505	0.220168

In [39]:
```
slope[['slope', 'PC2']].iplot(title='PC2 : Slope', secondary_y='PC2')
```



In [40]:
```
# Verify the correlation
np.corrcoef(principal_components[1], slope['slope'])
```

Out[40]:
```
array([[1.        , 0.95856134],
       [0.95856134, 1.        ]])
```

When running the correlation between the second principal component and the slope (10Y - 2Y) of the yield curve, a high correlation of 95.85% shows us that the second principal component represent the slope.

References

[1] [The Bank of England](https://www.bankofengland.co.uk)

[2] Scikit-learn PCA Decomposition

[3] Paul Wilmott (2007), Paul Wilmott introduces Quantitative Finance

---

Kannan Singaravelu
https://github.com/kannansingaravelu
Certificate in Quantitative Finance, June 2020