APM466 – Mathematical Finance
Assignment 1 – Yield Curves

Zikun Chen
1001117882

January 2019

Fundamental Questions

1. Bonds with an initial term of 30 years should not be used to calculate points on the yield curve less than 5 years out. This is because in order to properly perform the bootstrapping technique, before it is possible to bootstrap longer term rates, we have to build up the shorter end of the term structure from shorter term bonds. Shorter term bonds have less payments to be discounted. If we use long term bonds like the 30 year bond, we will not be able to bootstrap the rates within 5 years properly because we do not have the proper long term rates to discount coupon payments occurring after 5 years and price the bond accurately. Additionally, longer term bonds have a less liquid market where majority of the investors are long term players in the financial market. This can influence the accuracy of the quoted bond prices, which is a factor in deriving the curve.

   The on-the-run Canadian 5-year bonds are government bonds with maturity of five years that were most recently issued, in contrary to off-the-run ones that are issued before the most recent issue date. Since we know that the 10-year Canadian bond is maturing in around 5 years, we can conclude that it is off-the-run since the Canadian government issues new bonds quarterly. Even though the off-the-run 10-year bond has maturity closer to 5 years, which is the point in time that we are interested in, it is clear that the 5-year on-the-run bond is a better choice based on two reasons–coupon payments and bond prices, both of which are important factors while calculating the yield at the 5-year time point.

   The 10-year bond has a coupon rate based on the specific economic conditions when it was issued, around five years ago. These economic conditions change over-time. Therefore, using coupon payments of the 10-year bond issued 5 years ago to forecast the yield curve today can be problematic. Furthermore, off-the-run government bonds suffer from price distortion because they are thinly traded compared to on-the-run one. There is

a liquidity premium in place when we consider the price of off-the-run bonds. On-the-run government bonds have a higher price in the market compared to those with similar maturities because they are more liquid, meaning that they are easier to transact on the market due to larger trading activities, reflecting market conditions better. [1]

2. Original dataset acquired on 2019-01-18:

	Coupon	Eff. Maturity	Price	Yield
0	0.50	2019-Feb-01	99.93	1.61
1	1.75	2019-Mar-01	100.02	1.66
2	0.75	2019-May-01	99.72	1.67
3	3.75	2019-Jun-01	100.79	1.70
4	0.75	2019-Aug-01	99.43	1.77
5	1.75	2019-Sep-01	99.97	1.80
6	1.25	2019-Nov-01	99.55	1.82
7	1.25	2020-Feb-01	99.39	1.83
8	1.50	2020-Mar-01	99.62	1.84
9	1.75	2020-May-01	99.88	1.84
10	3.50	2020-Jun-01	102.26	1.85
11	1.75	2020-Aug-01	99.82	1.87
12	0.75	2020-Sep-01	98.21	1.86
13	2.00	2020-Nov-01	100.22	1.88
14	2.25	2021-Feb-01	100.76	1.87
15	0.75	2021-Mar-01	97.65	1.87
16	3.25	2021-Jun-01	103.21	1.87
17	9.75	2021-Jun-01	118.44	1.84
18	0.75	2021-Sep-01	97.12	1.87
19	0.50	2022-Mar-01	95.82	1.88
20	2.75	2022-Jun-01	102.90	1.86
21	9.25	2022-Jun-01	124.24	1.85
22	1.00	2022-Sep-01	96.91	1.88
23	1.75	2023-Mar-01	99.49	1.88
24	1.50	2023-Jun-01	98.44	1.87
25	8.00	2023-Jun-01	125.76	1.86
26	2.00	2023-Sep-01	100.56	1.87
27	2.25	2024-Mar-01	101.80	1.88
28	2.50	2024-Jun-01	103.16	1.88
29	2.25	2025-Jun-01	102.11	1.90
30	9.00	2025-Jun-01	142.65	1.89
31	1.50	2026-Jun-01	97.13	1.92
32	1.00	2027-Jun-01	92.80	1.93
33	8.00	2027-Jun-01	146.85	1.93
34	2.00	2028-Jun-01	100.48	1.94
35	2.25	2029-Jun-01	102.68	1.96
36	5.75	2029-Jun-01	135.26	1.98

37	5.75	2033-Jun-01	145.87	2.05
38	5.00	2037-Jun-01	143.82	2.11
39	4.00	2041-Jun-01	132.59	2.16
40	3.50	2045-Dec-01	127.22	2.16
41	2.75	2048-Dec-01	113.17	2.15
42	2.00	2051-Dec-01	96.46	2.15
43	2.75	2064-Dec-01	118.52	2.12

The pre-processing include the following steps:

1. Use JANSTART and FEBSTART constant variables to specify today's date (T0) used for discounting, rounded to the nearest month. For example, for any data acquired on 2019-01-05 and after, today's date is consider to be 2019-02-01 (FEBSTART) and time to maturity (Column T) of a bond maturing on 2019-03-01 is 1 month. Column T above reflects this. 2. Turn effective maturities into time to maturity (Column T) in years
3. Turn quoted clean prices into dirty price by using the formula:

Dirty Price = Clean Price + Accrued Interest (AI)

where $AI = \frac{n}{365} *$ Annual Coupon Payment, n is the number of days since the last payment
4. Turn coupon rates into semiannual rates
5. Filter for bounds that have matures every six months starting from March 1st, 2019. 6. Delete irrelevant columns

The following example shows the above bond dataset after pre-processing:

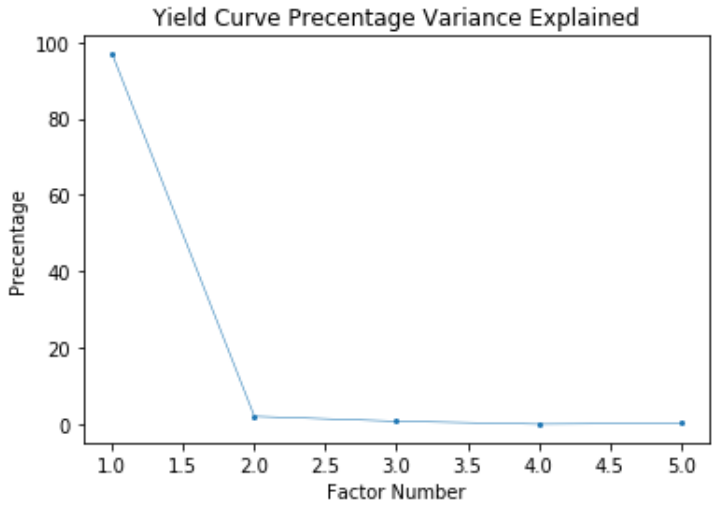
	Coupon	Price	Yield	T
1	0.875	100.753562	1.66	0.083333
5	0.875	100.703562	1.80	0.583333
8	0.750	100.248767	1.84	1.083333
12	0.375	98.524384	1.86	1.583333
15	0.375	97.964384	1.87	2.083333
18	0.375	97.434384	1.87	2.583333
19	0.250	96.029589	1.88	3.083333
22	0.500	97.329178	1.88	3.583333
23	0.875	100.223562	1.88	4.083333
26	1.000	101.398356	1.87	4.583333
27	1.125	102.743151	1.88	5.083333

Here, the column T indicates in how many years the bond is going to mature starting from today (2019-02-01, since 2019-01-18 is rounded to the nearest month).

As we can see, between each pair of consecutive bonds, the difference in time to maturity is half a year. The reason why I chose these particular

bonds is because the fact that coupon payments of bonds happen every six months. To bootstrap a new spot rate, we need previous spot rates at each six-month time point to discount the payments properly. Therefore, 11 bonds maturing in March and September each year for a little more than 5 years from 2019-02-01 are chosen to bootstrap spot rates every six months, from which 1 to 5 year spot rates can be linearly interpolated.

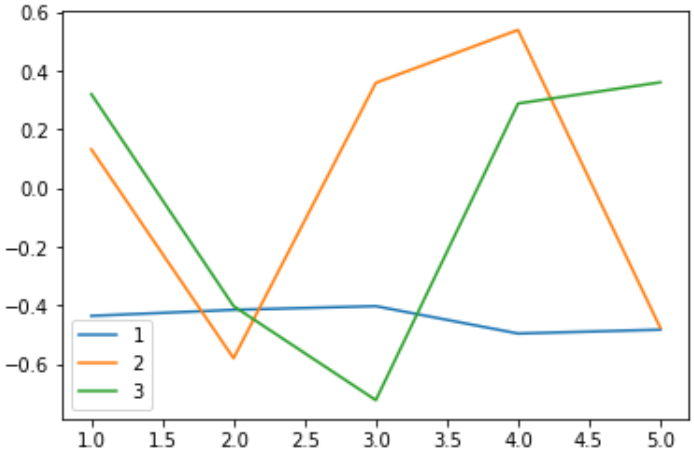
3. (a) As we can see from the eigenvalue distribution of the covariance of log-return of yields, the factor components corresponding to the first three eigenvalues can explain 99.66% of the variance of the yield curve dynamic.




```
Eigenvalues:

[7.43882409e-04 1.58528516e-05 6.15410977e-06 4.74666447e-07
 2.14861944e-06]

Eigenvectors (By Columns):

[[-0.43502931  0.13266411  0.3209918  -0.38315746 -0.73709183]
 [-0.41443085 -0.57937123 -0.4017479   0.49570742 -0.29231633]
 [-0.40184111  0.35929959 -0.72229408 -0.38950092  0.18975726]
 [-0.49519691  0.5397641   0.28891041  0.57773753  0.21490664]
 [-0.48197543 -0.47569711  0.36108852 -0.34924619  0.53763754]]
```

The 3 principle factors implied by the largest eigenvalues provides us with information about the yield dynamics [4]. As we can see from the following plot of different components, the first component represents "level" in the yield curve dynamic, meaning that all yields move up and down together in a parallel fashion. The second component represents the zig-zag

pattern of the curve. The third component holds information about the curvature with the seesaw-like rotation. Effectively, we decomposed the yield curve into three independent structure explaining over 99% of the rate movement.



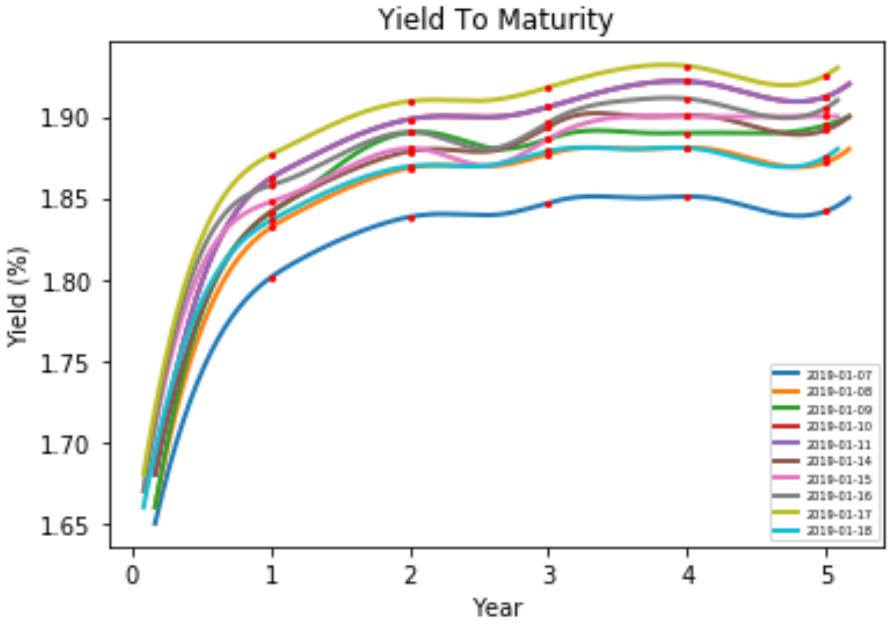
3. (b) The eigenvalue distribution of the yield log-return covariance matrix will be very different from those of individual US stock returns in SP500. We are looking only at 5 random variables for the yield curve. There are more stocks within the stock index, therefore more complex correlations dynamics can arise. More eigenvalues are expected to contribute to explaining the variance in the overall index.

   Although the US equity market can slump together in events like federal reserve rate hikes. The extent of the drop for individual stocks might not be the same. More often, certian events cause stock prices to react differently. For example, a drop in crude oil price can trigger very opposite responses for suppliers and consumers of oil. [3] Therefore, the covariance matrix of the log-returns of US stocks will produce very different eigenvalue distributions than what we found in the 1-5 year yield curve.

   Hedge funds are expected by investors to have exceptional performance compared to the market. They are less regulated and are able to take on more risk and deploy their own investment strategies to try and produce these outstanding returns. Therefore, the correlations between different hedge funds will not be as consistent as those between short-term and long-term yields. There are greater volatility in returns of different hedge funds returns. We certainly do not expect them to move together in one direction.

Empirical Questions

5. (a) To produce the yield curve, we first graph the yields given from the dataset against their corresponding time to maturities. We then use cubic spline interpolation to produce the following yield to maturity curve at 1 to 5 year time points (marked with red) since it is an accessible method and one that gives reasonable accuracy for the spot rate curve [2].



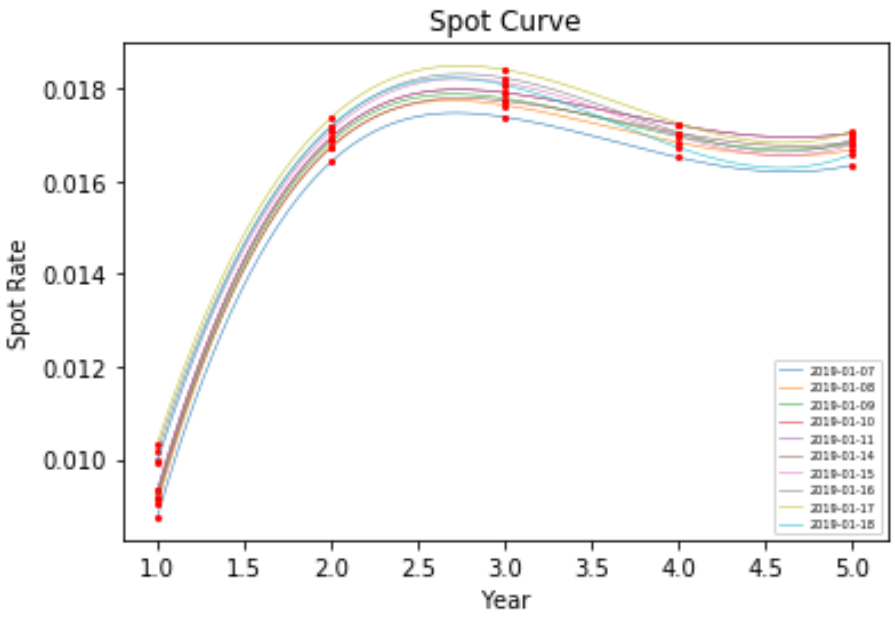
5. (b) **Relevant Formulas:**
Given:
- Maturity t_m
- Fixed semiannual coupon payment C_m at future times $\{t_1, \cdots, t_{m-1}\}$
- Zero rates $\{r_1, \cdots, r_{m-1}\}$ at future times $\{t_1, \cdots, t_{m-1}\}$

The price of the bond (maturing at t_m) at time t_0 (today) is given by:

$$P_m = \sum_{i=1}^{m-1} C_m e^{-r_i t_i} + (C_m + 100) e^{-r_m t_m}$$

Therefore, if the price of the bond P is known, then:

$$r_m = \frac{-ln[\frac{P_m - \sum_{i=1}^{m-1} C_m e^{-r_i t_i}}{C_m + 100}]}{t_m}$$



(Note that if the bond is a zero coupon bond, then the accumulated discounted coupon payments before maturity $\sum_{i=1}^{m-1} C_m e^{-r_i t_i} = 0$, and we obtain the formula $r_m = \frac{-ln[P_m/(C_m+100)]}{t_m}$)

Pseudo-Code:

Input:
Data Table (Data Frame) with 3 Columns – Semiannual coupon payment with principle of $100, dirty price of the bond, and time to maturity given by fraction of the year. Each row is a different bond with different maturities specified in Section 2.1.2 above.

Output:
A list of bootstrapped spot rates from year 1 to year 5.

Algorithm:

```
spot_rates(bonds):
    C = bonds[coupon]
    P = bonds[price]
    T = bonds[maturity]
    n = number of bonds # 10 bonds

    # spot rates corresponding to each maturity in bonds
    rates = list of size n
```

```
    for m from 0 to n-1:
        # accumulated discounted coupon payments before maturity
        payments = 0
        if m > 0:
            for i in range(m-1):
                payments += C[m] * np.exp(-rates[i] * T[i])
        rate = - np.log((P[m] - payments)/(100 + C[m])) / maturity[m]
        rates[m] = rate
    return spline_interpolate(rates, [1 to 5 year points]), [array of 1 to 5]
```

5. (c) **Relevant Formulas:**
   Given m year spot rates $\{r_1, \cdots, r_m\}$ at future times $\{t_1, \cdots, t_m\}$

   The 1yr-jyr forward rate is give by:

$$r_{1,j} = \frac{r_j t_j - r_1 t_1}{t_j - t_1}$$

```
forward_rates(bonds):
    # use the spot_rate function from part b)
    spot_rates = spot_rates(bonds)
    m = length(spot_rates)
    forward_rates = list of size m
    for i from 1 to m:
        riti = spot_rates[i]*five_years[i]
        r0t0 = spot_rates[0]*five_years[0]
        forward_rates[i-1] = (riti - r0t0)/(five_years[i]-five_years[0])
    return forward_rates
```

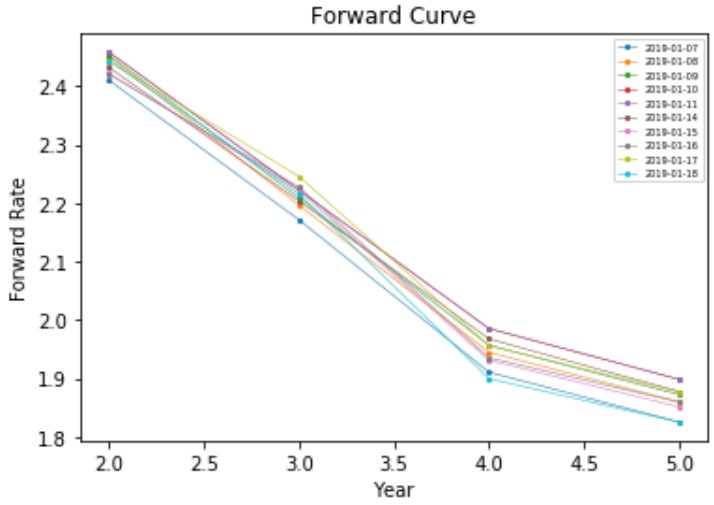
6. Covariance matrix of daily log-return of yield:

```
[[0.00014293 0.00013247 0.00012914 0.00016151 0.0001549 ]
 [0.00013247 0.00013438 0.00012216 0.00014699 0.00015164]
 [0.00012914 0.00012216 0.00012553 0.0001498  0.00014004]
 [0.00016151 0.00014699 0.0001498  0.0001878  0.00017427]
 [0.0001549  0.00015164 0.00014004 0.00017427 0.00017787]]
```

   Covariance matrix of daily log-return of forward rates:

```
[[5.80162383e-05 3.70345279e-05 7.27702769e-05 7.26772454e-05]
 [3.70345279e-05 6.76431486e-05 9.28378008e-05 9.66728970e-05]
 [7.27702769e-05 9.28378008e-05 2.47198036e-04 2.34307831e-04]
 [7.26772454e-05 9.66728970e-05 2.34307831e-04 2.25569980e-04]]
```

7. Eigenvalues and eigenvectors of covariance matrix of daily log-return of yield:




```
Eigenvalues:

[7.43882409e-04 1.58528516e-05 6.15410977e-06 4.74666447e-07
 2.14861944e-06]

Eigenvectors (By Columns):

[[-0.43502931  0.13266411  0.3209918  -0.38315746 -0.73709183]
 [-0.41443085 -0.57937123 -0.4017479   0.49570742 -0.29231633]
 [-0.40184111  0.35929959 -0.72229408 -0.38950092  0.18975726]
 [-0.49519691  0.5397641   0.28891041  0.57773753  0.21490664]
 [-0.48197543 -0.47569711  0.36108852 -0.34924619  0.53763754]]
```

Eigenvalues and eigenvectors of covariance matrix of daily log-return of forward rates:

```
Eigenvalues:

[5.36269654e-04 3.70846708e-05 2.44288493e-05 6.44228446e-07]

Eigenvectors (By Columns):

[[ 0.2225233   0.80761164 -0.54566431 -0.02229962]
 [ 0.28411846  0.46723844  0.81519992 -0.19082462]
 [ 0.67168568 -0.33228413 -0.19198561 -0.63369324]
 [ 0.64699183 -0.13798163  0.02900139  0.74934743]]
```

Appendix: Python Code

<bootstrap.py>

```
import pandas as pd
import numpy as np
from scipy.interpolate import make_interp_spline

# CONSTANTS
PAYMENT_PERIOD = 6
PERIOD = 5 # in years
DAYS_IN_YEAR = 365
FIVE_YEARS = [i for i in range(1,6)]

class bootstrap():

    def __init__(self, date):
        self.date = date

    def preprocess(raw_bonds, T0):

        bonds = raw_bonds

        # Format Maturity
        bonds['Eff. Maturity'] = pd.to_datetime(bonds['Eff. Maturity'])

        # Today's Date
        bonds['T0'] = pd.to_datetime(T0)

        # Time to Maturity In Month
        bonds['T'] = round((bonds['Eff. Maturity'] - bonds['T0'])/np.timedelta64(1, 'M'))

        # Filter desired bonds for rate calculations
        start_maturity = int(bonds['T'][1])
        threshold = start_maturity + 12 * PERIOD
        bonds = bonds[bonds['T'] <= threshold]
        desired_maturities =  [maturity for maturity in \
                               range(start_maturity, threshold + 1, PAYMENT_PERIOD)]
        bonds = bonds[bonds['T'].isin(desired_maturities)]

        # Calculate Clean Price
        bonds['Last_Coupon_T'] = np.repeat((bonds.loc[[1]]['Eff. Maturity'] - \
                 pd.DateOffset(months=PAYMENT_PERIOD)).values, bonds.shape[0])
        bonds['Accured_Interest'] = ((bonds['T0'] - bonds['Last_Coupon_T']) \
                /np.timedelta64(1, 'D'))/ DAYS_IN_YEAR * bonds['Coupon']
        bonds['Price'] = bonds['Price'] + bonds['Accured_Interest']
```

```
    # Delete irrelevant columns
    bonds = bonds.drop(columns="T0")
    bonds = bonds.drop(columns="Eff. Maturity")
    bonds = bonds.drop(columns="Last_Coupon_T")
    bonds = bonds.drop(columns="Accured_Interest")

    # Adjust Coupon to reflect semiannual payment
    bonds['Coupon'] = bonds['Coupon']/2
    bonds['T'] = bonds['T']/12

    return bonds


def ytm(bonds):
    """Return 5 year ytm and maturity from a set of bonds"""
    ytm = bonds['Yield'].values
    maturity = bonds['T'].values
    return ytm, maturity


def spot_rates(bonds):
    """Return 5 year zero rates and maturity from a set of bonds"""
    principle = 100

    coupon = bonds['Coupon'].values
    price = bonds['Price'].values
    maturity = bonds['T'].values

    n = coupon.shape[0]
    rates = [0]*n

    for m in range(n):
        payments = 0
        if m > 0:
            for i in range(m-1):
                payments += coupon[m] * np.exp(-rates[i] * maturity[i])
        rate = - np.log((price[m] - payments)/(principle + coupon[m])) / maturity[m]
        rates[m] = rate

    spl = make_interp_spline(maturity, rates, k=3) #BSpline object
    return spl(FIVE_YEARS), np.array(FIVE_YEARS)

def one_year_forward_rates(spot_rates):
    """Generate 1 year forward rates from zero rates from 2 to 5 years out."""
    m = len(spot_rates)
```

```
        forward_rates = [0] * (m-1)
        for i in range(1, m):
            riti = spot_rates[i]*FIVE_YEARS[i]
            r0t0 = spot_rates[0]*FIVE_YEARS[0]
            forward_rates[i-1] = (riti - r0t0)/(FIVE_YEARS[i]-FIVE_YEARS[0])
        return np.array(forward_rates)

    def cov(rates):
        """Generate covariance matrix of time series"""
        df = pd.DataFrame.from_dict(rates)
        n = df.shape[0] # number of assets
        m = df.shape[1] # number of features

        df.columns = np.arange(1, m+1)
        df.index = np.arange(1, n+1)

        df = df.shift(periods=1, axis='columns') / df
        df = df.drop(columns = 1)

        cov_matrix = np.cov(df)

        return cov_matrix, np.linalg.eig(cov_matrix)
```

<yield_curve.py>

```
from bootstrapping import bootstrap as bs
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import make_interp_spline



FILENAME = "raw_bond_data.xls"

WEEK_ONE_START = '2019-01-07'
WEEK_TWO_START = '2019-01-14'

JANFIRST = '2019-01-01'
FEBFIRST = '2019-02-01'

FIVE_YEARS = [i for i in range(1,6)]
```

```
if __name__ == "__main__":

# ================================================================================
#     Pre-Processing
# ================================================================================

    bonds_dict = {}

    for i in range(10):
        if i < 5:
            date = str((pd.to_datetime(WEEK_ONE_START) \
                + pd.Timedelta(i, unit='d')).date())
        else:
            date = str((pd.to_datetime(WEEK_TWO_START) \
                + pd.Timedelta(i-5, unit='d')).date())

        raw_bonds = pd.read_excel(FILENAME, sheet_name = date)

        # Round today's date to the start of next month if necessary for easier calculation
        if int(date[-2:]) < 15:
            bonds = bs.preprocess(raw_bonds, JANFIRST)
        else:
            bonds = bs.preprocess(raw_bonds, FEBFIRST)

        bonds_dict[date] = bonds


# ================================================================================
#     Yield Curve
# ================================================================================

    ### all ytm
    yields = {}
    legend = [0]*20
    for i in range(10):
        if i < 5:
            date = str((pd.to_datetime(WEEK_ONE_START) \
                + pd.Timedelta(i, unit='d')).date())
        else:
            date = str((pd.to_datetime(WEEK_TWO_START) \
                + pd.Timedelta(i-5, unit='d')).date())

        bonds = bonds_dict[date]
        ytm, maturity = bs.ytm(bonds)
        legend[2*i] = date
```

```
        xnew = np.linspace(maturity.min(),maturity.max(),300)
        spl = make_interp_spline(maturity, ytm, k=3) #BSpline object
        ytm_smooth = spl(xnew)
        plt.plot(xnew, ytm_smooth, '-', linewidth=2)

        ytm = spl(FIVE_YEARS)
        yields[date] = ytm
        plt.plot(FIVE_YEARS, ytm, 'ro', markersize=2, label='_nolegend_')

    plt.title('Yield To Maturity')
    plt.ylabel('Yield (%)')
    plt.xlabel('Year')
    plt.legend(legend[::2], prop={'size': 5})
    plt.show()

# ============================================================================
#     Spot Curve
# ============================================================================

    spot_rates = {}
    legend = [0]*20
    for i in range(10):
        if i < 5:
            date = str((pd.to_datetime(WEEK_ONE_START) \
                + pd.Timedelta(i, unit='d')).date())
        else:
            date = str((pd.to_datetime(WEEK_TWO_START) \
                + pd.Timedelta(i-5, unit='d')).date())

        bonds = bonds_dict[date]
        spot_rate, maturity = bs.spot_rates(bonds)
        legend[2*i] = date

        xnew = np.linspace(maturity.min(),maturity.max(),300)
        spl_spot = make_interp_spline(maturity, spot_rate, k=3) #BSpline object
        spot_rate_smooth = spl_spot(xnew)
        plt.plot(xnew, spot_rate_smooth, '-', linewidth=0.5)

        spot_rate = spl_spot(FIVE_YEARS)
        plt.plot(FIVE_YEARS, spot_rate, 'ro', markersize=2, label='_nolegend_')
        spot_rates[date] = spot_rate

    plt.title('Spot Curve')
    plt.ylabel('Spot Rate')
    plt.xlabel('Year')
    plt.legend(legend[::2], prop={'size': 5}, loc = 'lower right')
```

```
    plt.show()


# ============================================================================
#     Forward Rates
# ============================================================================

    forward_rates = {}
    legend = [0]*10
    for i in range(10):
        if i < 5:
            date = str((pd.to_datetime(WEEK_ONE_START) \
                + pd.Timedelta(i, unit='d')).date())
        else:
            date = str((pd.to_datetime(WEEK_TWO_START) \
               + pd.Timedelta(i-5, unit='d')).date())

        bonds = bonds_dict[date]
        spot_rates, maturity = bs.spot_rates(bonds)
        forward_rate = bs.one_year_forward_rates(spot_rates)
        forward_rates[date] = forward_rate
        legend[i] = date
        plt.plot(FIVE_YEARS[1:], forward_rate*100, 'o-', linewidth=0.5, markersize=2)

    plt.title('Forward Curve')
    plt.ylabel('Forward Rate')
    plt.xlabel('Year')
    plt.legend(legend, prop={'size': 5})
    plt.show()


# ============================================================================
#     Covariance Matrix
# ============================================================================

    ## Yield Rates

    cov_yield, (eigenvalue, eigenvectors) = bs.cov(yields)

    print('Covariance Matrix: \n')
    print(cov_yield)

    df_ev_yield = pd.DataFrame(eigenvectors)
    df_ev_yield.columns  = eigenvalue

    fig = plt.figure()
```

```
plt.plot(FIVE_YEARS, eigenvalue/sum(eigenvalue)*100, 'o-', \
    linewidth=0.5, markersize=2)

plt.title('Yield Curve Precentage Variance Explained')
plt.ylabel('Precentage')
plt.xlabel('Factor Number')
plt.show()

print('Eigenvalues: \n')
print(eigenvalue)
print('Eigenvectors (By Columns): \n')
print(eigenvectors)

PC = pd.DataFrame(eigenvectors)
PC.columns  = FIVE_YEARS
PC.index = range(1,6)
PC.iloc[:, :3].plot()

## Forward Rates

cov_forward, (eigenvalue, eigenvectors) = bs.cov(forward_rates)

print('Covariance Matrix: \n')
print(cov_forward)

df_ev_forward = pd.DataFrame(eigenvectors)
df_ev_forward.columns  = eigenvalue

fig = plt.figure()

plt.plot(FIVE_YEARS[1:], eigenvalue/sum(eigenvalue)*100, \
    'o-', linewidth=0.5, markersize=2)

plt.title('Forward Curve Percentage Variance Explained')
plt.ylabel('Percentage')
plt.xlabel('Factor Number')
plt.show()

print('Eigenvalues: \n')
print(eigenvalue)
print('Eigenvectors (By Columns): \n')
print(eigenvectors)
```

References

[1] **Investopedia.** *On-The-Run Treasuries, https://www.investopedia.com/terms/o/on-the-runtreasuries.asp*

[2] **Rod Pienaar, Moorad Choudhry.** *Fitting the term structure of interest rates: the practical implementation of cubic spline methodology*

[3] **Andrew Hecht.** The Price of Crude Oil and Equity Prices *https://www.thebalance.com/the-price-of-crude-oil-and-equity-prices-808866*

[4] **Gary Kennedy.** Principle Component Analysis of the swap curve: an introduction *https://www.clarusft.com/principal-component-analysis-of-the-swap-curve-an-introduction/*