Fixed Income: Yield Curve Fitting

Jeppe Johansen (32596), Jonas Philip Christensen (32704)

February 20, 2018

1 Estimation of YTM and yield curves

The yield to maturity is the rate of return that solves the price equation:

$$B(t) = \sum_{i=1}^{m} \frac{c/n}{e^{y_s*(T_m-t)}} + \frac{1}{e^{y_s*(T_m-t)}} \tag{1}$$

The YTM is a measure of realized returns if 1) the bond is held until maturity and 2) all coupons are reinvsted to the same rate of return. As we know the price, the coupon frequency and the maturity rates, the equation has one unknown and thus has a unique solution.

We solve for y_s-rates with numerical optimization in Python (see appendix) and get the yield curve profile

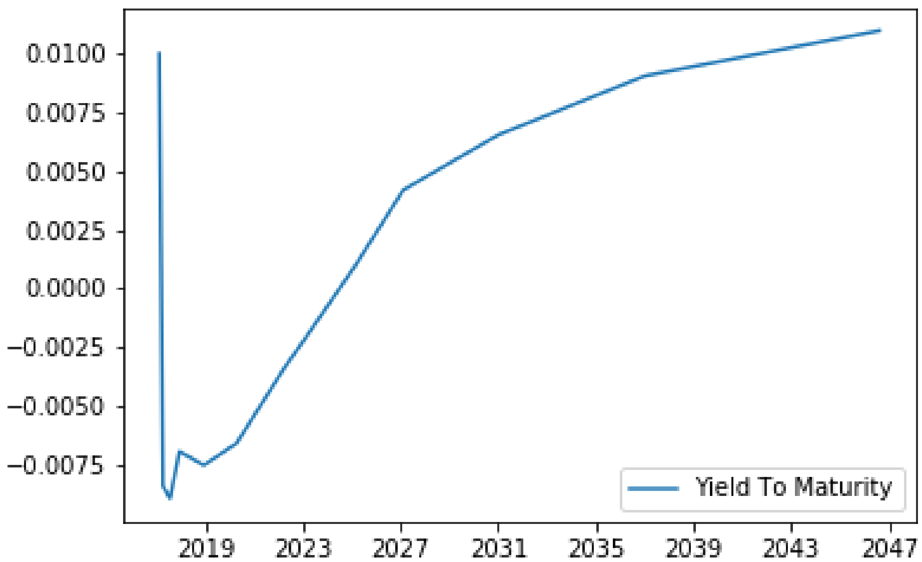


Figure 1: 3 Yield curve for German bonds

Ultra-short maturity bonds have a positive return, and so do the long-term bonds (10 years maturity and beyond), but the maturity rates in between have negative yields. This is highly unusual in a historical context, but is a result of quantitative easing and low (expected) inflation in the Euro-zone. The upward trend of the yield curve indicates, that the long-run expectations of the market is, that interest rate will increase.

2 Problem 2

We use numerical optimization to solve find the right parameter estimates, where the objective function is the mean-squared error, between the estimated prices of bonds and the market prices of bonds. To find the right parameter estimates, good initial guesses is needed, such that the optimization doesn't end up in local maxima. For the Nelson-Siegel the following was done:

Nelson-Siegel: Let β_1 be the long rate found in the market, we set that to be approximately $\beta_1 = 0.01$, that might be a tad low, however this is just initial guesses. Next we chose β_2 s.t. $\beta_1 + \beta_2 =$ short rate. This leads to an $\beta_2 = -0.1$. We set $\lambda_1 = 2$ and $\beta_3 = 0$ to be good initial guesses. Furthermore we saw that similar values was presented in the lecture notes. The Optimization has a mean squared error estimate of:

$$MSE_{Nelson-Siegel} = 0.0004302 \tag{2}$$

Svensson: Finding good estimates for $\beta_1, \beta_2, \beta_3, \lambda_1$ in the first optimization (look table 1) we use these as initial guesses in the Svensson model. The initial guesses for $\beta_4 = 0$, and $\lambda_2 = 2$. Table 1, shows the results. The mean squared error estimate of the model is:

$$MSE_{Svensson} = 0.0003997 \tag{3}$$

Table 1: Parameter Estimates of Nelson-Siegel & Svensson

	Nelson-Siegel	Svensson
β_1	0.0149	0.0151
β_2	-0.0123	-0.0162
β_3	-0.0468	0.0036
β_4		-0.0544
λ_1	2.0003	0.8651
λ_2		2.0408

Table 2: Rates for different maturities & Svensson

	Nelson-Siegel	Svensson
$r_\infty(0, 1/52)$	0.0024	-0.0012
$r_\infty(0, 0.5)$	-0.0010	-0.0022
$r_\infty(0, 1)$	-0.0032	-0.0032
$r_\infty(0, 5)$	-0.0029	-0.0027
$r_\infty(0, 10)$	0.0034	0.0034
$r_\infty(0, 30)$	0.0110	0.0110

The three figures can be seen in one plot in figure 2. The Svensson-model and the Nelson-Siegel model can be found in separate figures in figure 3 and 4.

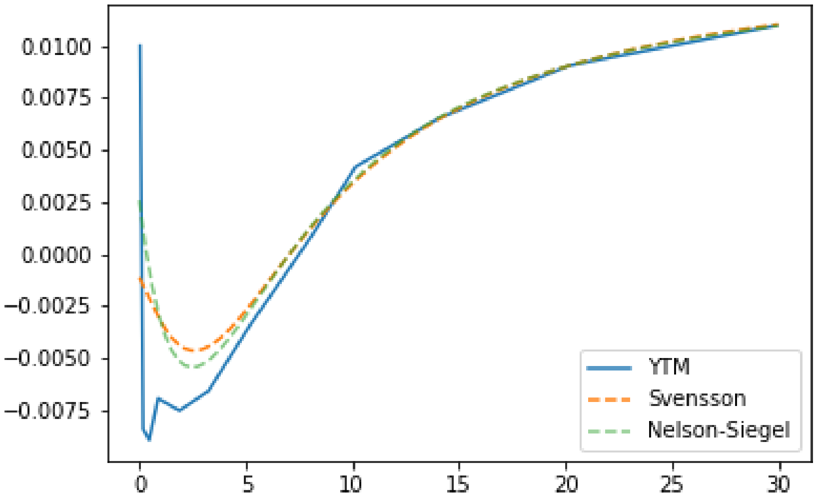


Figure 2: Nelson-Siegel, Svensson and Yield To Maturity

3 Comparison of Nelson-Siegler and Svennson

To determine the better model, one can use Mean Squared Error as a measure. The measure takes into account both the variance and the bias of the model

$$MSE = Var_{\hat{\theta}}(\hat{\theta}) + bias_{\hat{\theta}}(\hat{\theta}, \theta)^2 \tag{4}$$

Minimizing the MSE is a way to handle the trade-off between bias and variance of the model. Overfitting the model leads to a low bias, but a high degree of variance, as it captures all the noise in the data - This is usually not a good strategy, if one wants to estimate something out-of-sample.

Underfitting the model will give a smooth, well-behaved curve, and hence a low variance, but the bias will be relative big as a result

$$MSE_{Nelson-Siegler} = 0.00043 \tag{5}$$

$$MSE_{Svensson} = 0.00039 \tag{6}$$

The MSE indicates, that the Svensson-model is a better choice. However, the small difference among them suggests, that this has very limited implications in practical applications.

The Nelson-Siegler model has fewer parameters than the Svensson model, and is therefore less prone to overfitting, and better suited for few-observational fittings (due to more degrees of freedom).

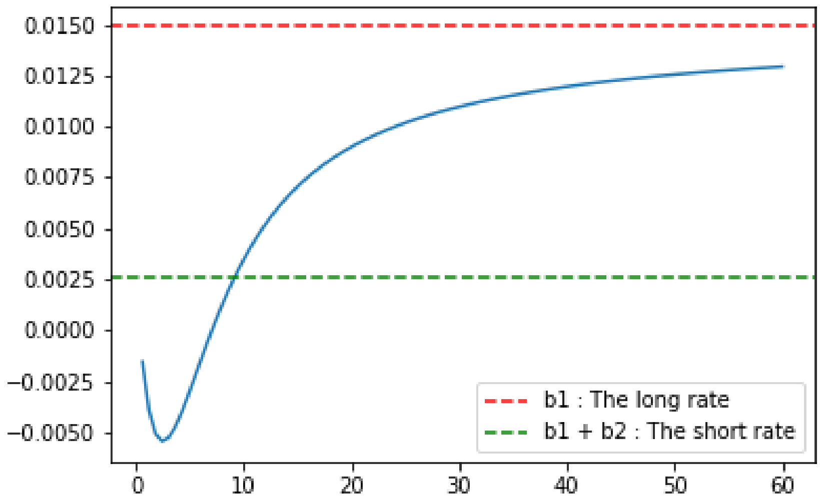


Figure 3: Nelson Siegel model

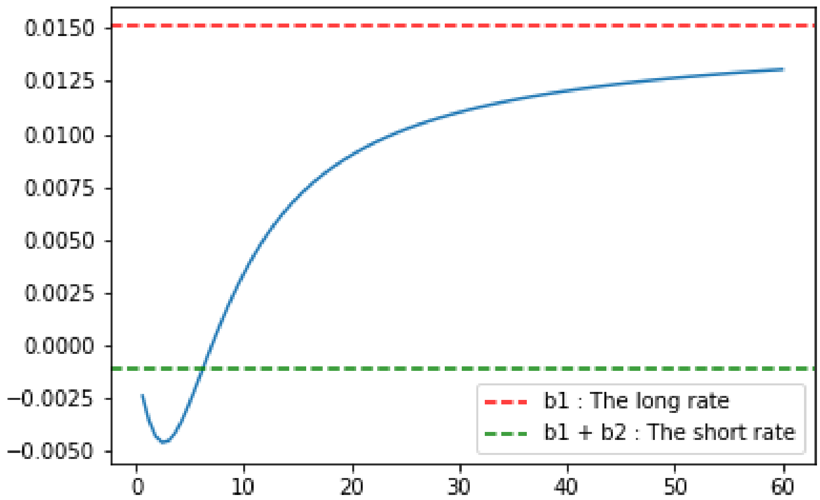


Figure 4: Svensson model

main

February 20, 2018

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from scipy.optimize import minimize
        %matplotlib inline
```

0.0.1 Loading data

```
In [2]: data = pd.read_excel('data.xlsx')
        data.columns
        data['nr_days'] = data['End_date'] - data['Start_date']
        data['price'] = data['price'] / 100
        data['coupon'] = data['coupon']/100
        data['nr_years'] = data['nr_days'].apply(lambda x: int(x / np.timedelta64(1,'D'))) / 360
        data
```

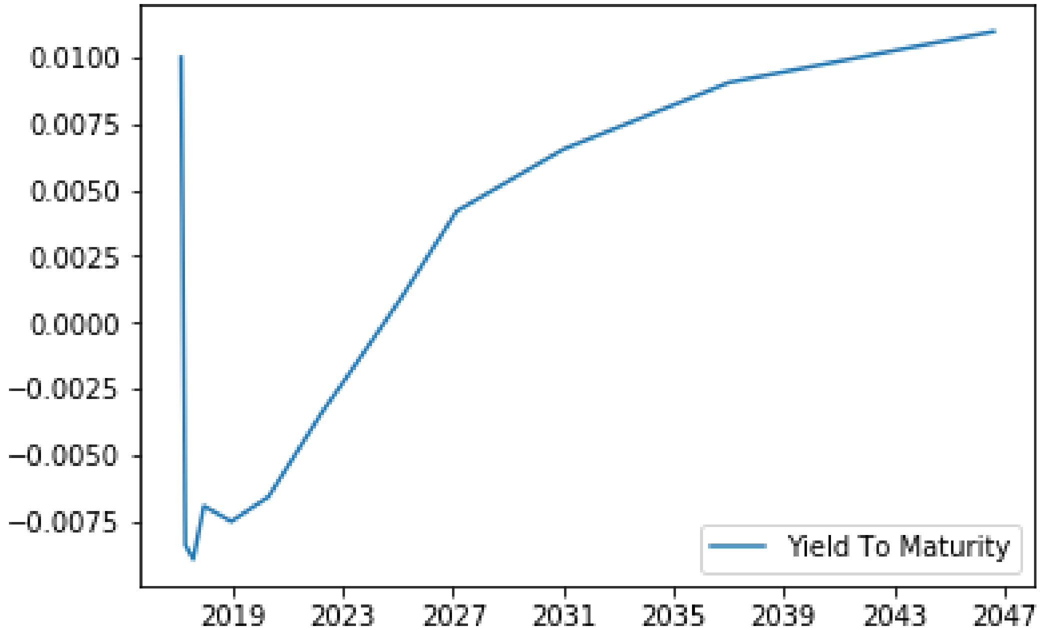
```
Out[2]:       price  month  coupon  Coupons pr year Start_date   End_date    nr_days  \
        0   1.00021      1  0.0000                1 2017-02-02 2017-02-15    13 days
        1   1.00626     12  0.0000                1 2017-02-03 2017-12-15   315 days
        2   1.01397     24  0.0000                1 2017-02-04 2018-12-14   678 days
        3   1.00153      3  0.0000                1 2017-02-05 2017-04-12    66 days
        4   1.02142     36  0.0000                1 2017-02-06 2020-04-17  1166 days
        5   1.01776     60  0.0000                1 2017-02-07 2022-04-08  1886 days
        6   1.00411      6  0.0000                1 2017-02-08 2017-07-26   168 days
        7   1.10358     72  0.0150                1 2017-02-09 2023-02-15  2197 days
        8   1.03233     96  0.0050                1 2017-02-10 2025-02-15  2927 days
        9   0.98297    120  0.0025                1 2017-02-11 2027-02-15  3656 days
        10  1.64471    180  0.0550                1 2017-02-12 2031-01-04  5074 days
        11  1.56051    240  0.0400                1 2017-02-13 2037-01-04  7265 days
        12  1.33147    360  0.0250                1 2017-02-14 2046-08-15 10774 days

            nr_years
        0   0.036111
        1   0.875000
        2   1.883333
        3   0.183333
        4   3.238889
        5   5.238889
```

```
6      0.466667
7      6.102778
8      8.130556
9     10.155556
10    14.094444
11    20.180556
12    29.927778
```

```
In [3]: def ListOfPeriods(n, T, t=0):

            last_coupon = int(np.floor(n*(T-t)))
            period_adder = n*(T-t) - last_coupon

            return [(i + period_adder)/n  for i in range(1, last_coupon + 1)]


        def BondPrice(c, n, T, ys, t=0):

            """Returns price, for given YTM"""

            LOP = ListOfPeriods(n, T, t=t)

            discounted_coupons = [(c/n) * np.exp(-(i-t)*(ys)) for i in LOP]
            FV = np.exp(-(T-t)*(ys))

            B = np.sum(discounted_coupons) + FV
            return B


        def YTM(B, c, n, T, t=0, ys_guess=0.01):

            def ObjectiveFunc(ys):

                return (B - BondPrice(c, n, T, ys, t=t))**2

            bound_ys = [(-1 , 1)]

            solution = minimize(fun= ObjectiveFunc, x0=ys_guess, method='SLSQP', bounds=bound_ys

            return solution.x[0]
```

```
In [4]: data['YTM'] = data.apply(lambda x: YTM(B = x['price'], c=x['coupon'], n= x['Coupons pr y
```

```
In [5]: f, ax = plt.subplots(1,1)
        data.sort_values('End_date', inplace=True)
        ax.plot(data['End_date'], data['YTM'], label='Yield To Maturity')
        ax.legend()
        plt.savefig('YTM.png')
```



1 Question 2

```
In [6]: class NielsonSiegelModel(object):

            def __init__(self):
                pass

            @staticmethod
            def NielsonSiegel(b1, b2, b3, lam1, T):

                return b1 + b2*((1-np.exp(-T/lam1))/(T/lam1)) + b3*(((1-np.exp(-T/lam1))/(T/lam1


            def VectorRates(self, b1, b2, b3, lam1, vector_T):

                """ Returns vector of rates"""
                return [self.NielsonSiegel(b1, b2, b3, lam1, Ti) for Ti in vector_T]


            def VectorPrices(self, vector_c, vector_n, vector_T, vector_r):

                list_of_prices = []
                for i in range(len(vector_c)):
                    c, n, T, ys = vector_c[i], vector_n[i], vector_T[i], vector_r[i]
```

```
                list_of_prices.append(BondPrice(c, n, T, ys, t=0))

            return list_of_prices

        def NSM(self, data, x_guess):

            """
            x_guess: (list)
            =======
            b1 = x[0]
            b2 = x[1]
            b3 = x[2]
            lam1 = x[3]

            =======
            returns : vector of x
            """

            vector_c, vector_n, vector_T, vector_p_market = data['coupon'], data['Coupons pr

            def ObjectiveFunc(x):
                b1 = x[0]
                b2 = x[1]
                b3 = x[2]
                lam1 = x[3]

                vector_r = self.VectorRates(b1, b2, b3, lam1, vector_T) # vector_r is endoge
                vector_p_model = self.VectorPrices(vector_c, vector_n, vector_T, vector_r)

                res = np.sum(np.array(np.array(vector_p_market) - np.array(vector_p_model))*
                return res

            bound=[(-2,2),(-2,2),(-30,30),(-30,30)]

            solution = minimize(fun= ObjectiveFunc, x0=x_guess, method='SLSQP', bounds=bound

            return solution
```

In [7]:
```
x_guess = [0.01, -0.01, 0 , 2]
NSM = NielsonSiegelModel()
sol = NSM.NSM(data, x_guess)
x_list = list(sol.x)
b1, b2, b3, lam1 = x_list[0], x_list[1], x_list[2], x_list[3]
print('b1: ',b1,'b2: ',b2,'b3: ',b3, 'lam1:', lam1)
b1_NS, b2_NS, b3_NS, lam1_NS = b1, b2, b3, lam1
```

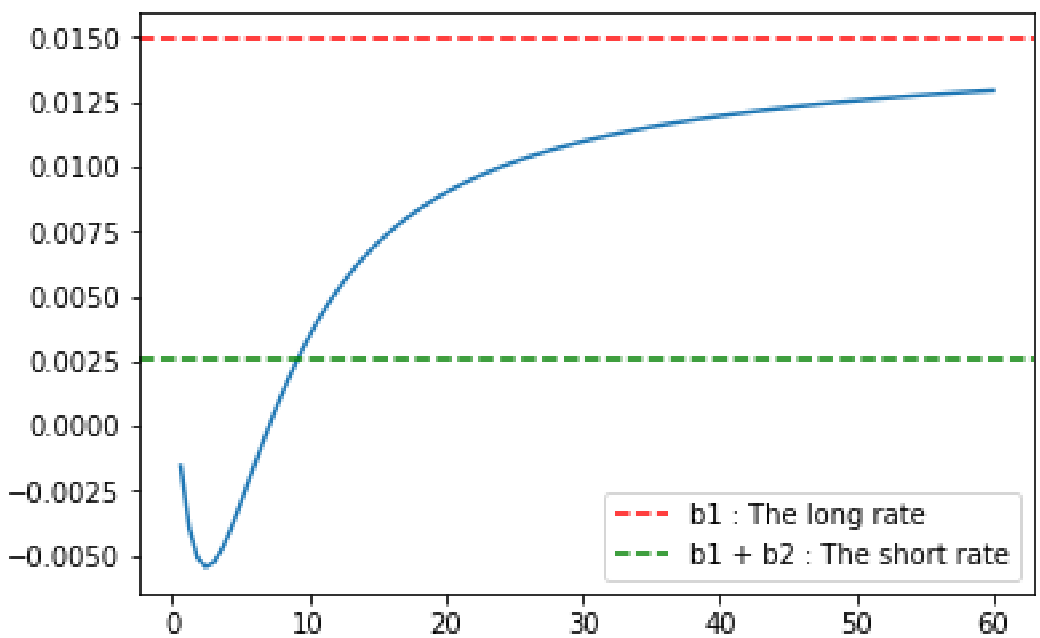
```
b1:  0.014907040259 b2:  -0.0123318899496 b3:  -0.0468055972558 lam1: 2.00034667642
```

```
In [8]: MeanSquaredErrorNelsonSiegel = sol.fun
        print(MeanSquaredErrorNelsonSiegel)
```

```
0.00043023479318226665
```

```
In [9]: x_= np.linspace(0,60,num=100)
        y_NSM = [NSM.NielsonSiegel(b1, b2, b3, lam1, i) for i in x_]

        f, ax = plt.subplots(1,1)
        ax.plot(x_, y_NSM)
        ax.axhline(b1, label= 'b1 : The long rate', color='red', ls='--')
        ax.axhline(b1+b2, label= 'b1 + b2 : The short rate', color='green', ls='--')
        ax.legend()
        plt.savefig('Nelson-Siegel.png')
```

```
/Users/Jeppe/anaconda3/lib/python3.6/site-packages/ipykernel/__main__.py:9: RuntimeWarning: inva
```




```
In [10]: class SvenssonModel(object):

             def __init__(self):
                 pass

             @staticmethod
             def Svensson(b1, b2, b3, b4, lam1, lam2, T):
```

```
        return b1 + b2*((1-np.exp(-T/lam1))/(T/lam1)) + b3*(((1-np.exp(-T/lam1))/(T/lam


    def VectorRates(self, b1, b2, b3, b4, lam1, lam2, vector_T):

        """ Returns vector of rates"""
        return [self.Svensson(b1, b2, b3, b4, lam1, lam2, Ti) for Ti in vector_T]


    def VectorPrices(self, vector_c, vector_n, vector_T, vector_r):

        list_of_prices = []
        for i in range(len(vector_c)):
            c, n, T, ys = vector_c[i], vector_n[i], vector_T[i], vector_r[i]

            list_of_prices.append(BondPrice(c, n, T, ys, t=0))

        return list_of_prices

    def SM(self, data, x_guess):

        """
        x_guess: (list)
        =======
        b1 = x[0]
        b2 = x[1]
        b3 = x[2]
        b4 = x[3]
        lam1 = x[4]
        lam2 = x[5]

        =======
        returns : vector of x
        """

        vector_c, vector_n, vector_T, vector_p_market = data['coupon'], data['Coupons p

        def ObjectiveFunc(x):
            b1 = x[0]
            b2 = x[1]
            b3 = x[2]
            b4 = x[3]
            lam1 = x[4]
            lam2 = x[5]

            vector_r = self.VectorRates(b1, b2, b3, b4, lam1, lam2, vector_T) # vector_
            vector_p_model = self.VectorPrices(vector_c, vector_n, vector_T, vector_r)
```

```
            res = np.sum(np.array(np.array(vector_p_market) - np.array(vector_p_model))
            return res

        bound=[(-2,2),(-2,2),(-2,2),(-2,2), (-10,10),(-10,10)]

        solution = minimize(fun= ObjectiveFunc, x0=x_guess, method='SLSQP', bounds=boun

        return solution
```

```
In [11]: x_guess = [0.014, -0.012, -0.468, 0, 2, 2]
         SM = SvenssonModel()
         sol = SM.SM(data, x_guess)
         x_list = list(sol.x)
         b1, b2, b3, b4, lam1, lam2 = x_list[0], x_list[1], x_list[2], x_list[3], x_list[4], x_l
         b1_SM, b2_SM, b3_SM, b4_SM, lam1_SM, lam2_SM = b1, b2, b3, b4, lam1, lam2
```

```
/Users/Jeppe/anaconda3/lib/python3.6/site-packages/ipykernel/__main__.py:57: RuntimeWarning: ove
/Users/Jeppe/anaconda3/lib/python3.6/site-packages/ipykernel/__main__.py:15: RuntimeWarning: ove
/Users/Jeppe/anaconda3/lib/python3.6/site-packages/ipykernel/__main__.py:16: RuntimeWarning: ove
```

```
In [12]: print('b1 :', b1, 'b2 :', b2, 'b3 :', b3,'b4 :', b4, 'lam1 :', lam1, 'lam2', lam2)
```

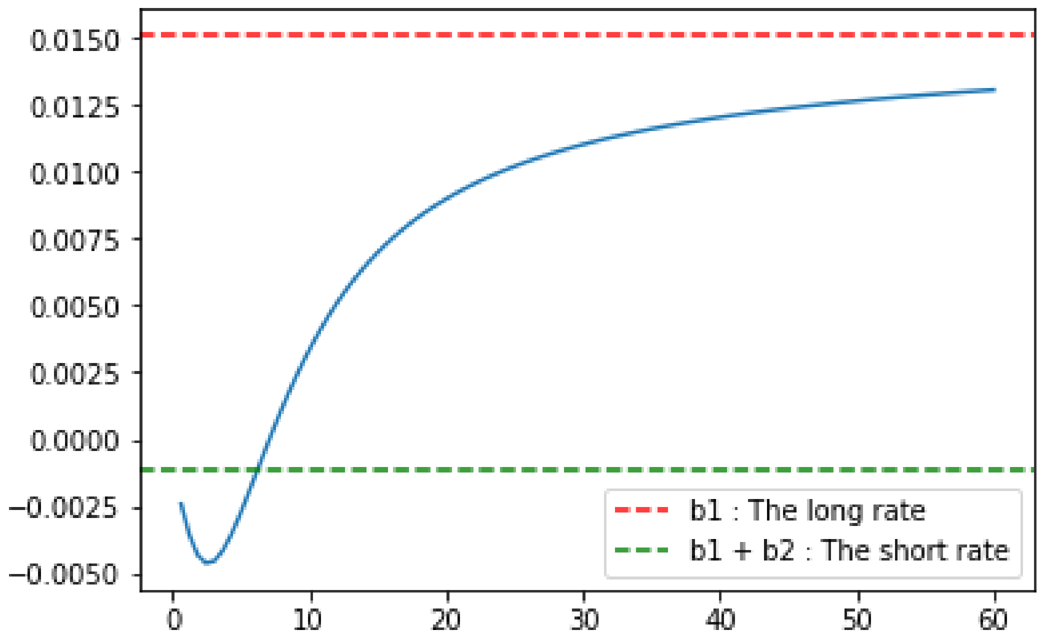
```
b1 : 0.0150950624961 b2 : -0.0162137997828 b3 : 0.00360089378392 b4 : -0.0544871532228 lam1 : 0.
```

```
In [13]: MeanSquaredErrorSvensson = sol.fun
         print(MeanSquaredErrorSvensson)
```

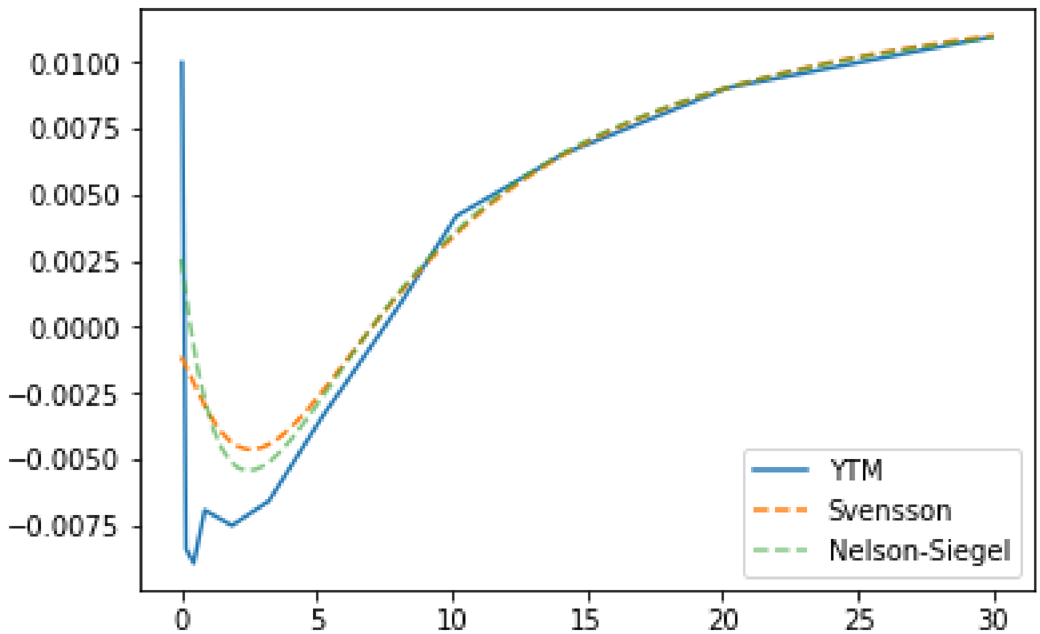
```
0.0003997039553169198
```

```
In [14]: x_= np.linspace(0,60,num=100)
         y_SM = [SvenssonModel.Svensson(b1, b2, b3, b4, lam1, lam2, i) for i in x_]

         f, ax = plt.subplots(1,1)
         ax.plot(x_, y_SM)
         ax.axhline(b1, label= 'b1 : The long rate', color='red', ls='--')
         ax.axhline(b1+b2, label= 'b1 + b2 : The short rate', color='green', ls='--')
         ax.legend()
         plt.savefig('Svensson.png')
```

```
/Users/Jeppe/anaconda3/lib/python3.6/site-packages/ipykernel/__main__.py:9: RuntimeWarning: inva
```




```
In [15]: x_compare = [1/52, 0.5, 1, 5, 10, 30]
         SM_rates = [SvenssonModel.Svensson(b1_SM, b2_SM, b3_SM, b4_SM, lam1_SM, lam2_SM, i) for
         NS_rates = [NSM.NielsonSiegel(b1_NS, b2_NS, b3_NS, lam1_NS, i) for i in x_compare]

In [16]: print(NS_rates)

[0.0024106873470304082, -0.00096489435833784724, -0.0032408950357681043, -0.0029654541346972024,


In [22]: x_= np.linspace(0.00001,30,num=100)
         SM_rates = [SvenssonModel.Svensson(b1_SM, b2_SM, b3_SM, b4_SM, lam1_SM, lam2_SM, i) for
         NS_rates = [NSM.NielsonSiegel(b1_NS, b2_NS, b3_NS, lam1_NS, i) for i in x_]

         f, ax = plt.subplots(1,1)
         ax.plot(data['nr_years'], data['YTM'])
         ax.plot(x_, SM_rates, label = 'Svensson', ls='--')
         ax.plot(x_, NS_rates, label = 'Nelson-Siegel', ls ='--', alpha=0.6)

         ax.legend()
         plt.savefig('ThreeFigures.png')
```


0.0100
0.0075
0.0050
0.0025
0.0000
−0.0025
−0.0050
−0.0075
0
5
10
15
20
25
30
YTM
Svensson
Nelson-Siegel