https://medium.com/@polanitzer/nelson-siegel-svensson-in-python-estimating-the-spot-rate-curve-using-the-nelson-siegel-svensson-4753969e61c8

Nelson-Siegel-Svensson in Python; Estimating the Spot Rate Curve using the Nelson-Siegel-Svensson (1994)

Roi Polanitzer · Follow
12 min read · Mar 24

3 1

The Nelson-Siegel-Svensson (1994) model modifies the Nelson-Siegel (1987) model by adding 2 additional curve estimation parameters. Virtualy any yield curve shape can be interpolated using these two models, which are widely used at banks around the world.

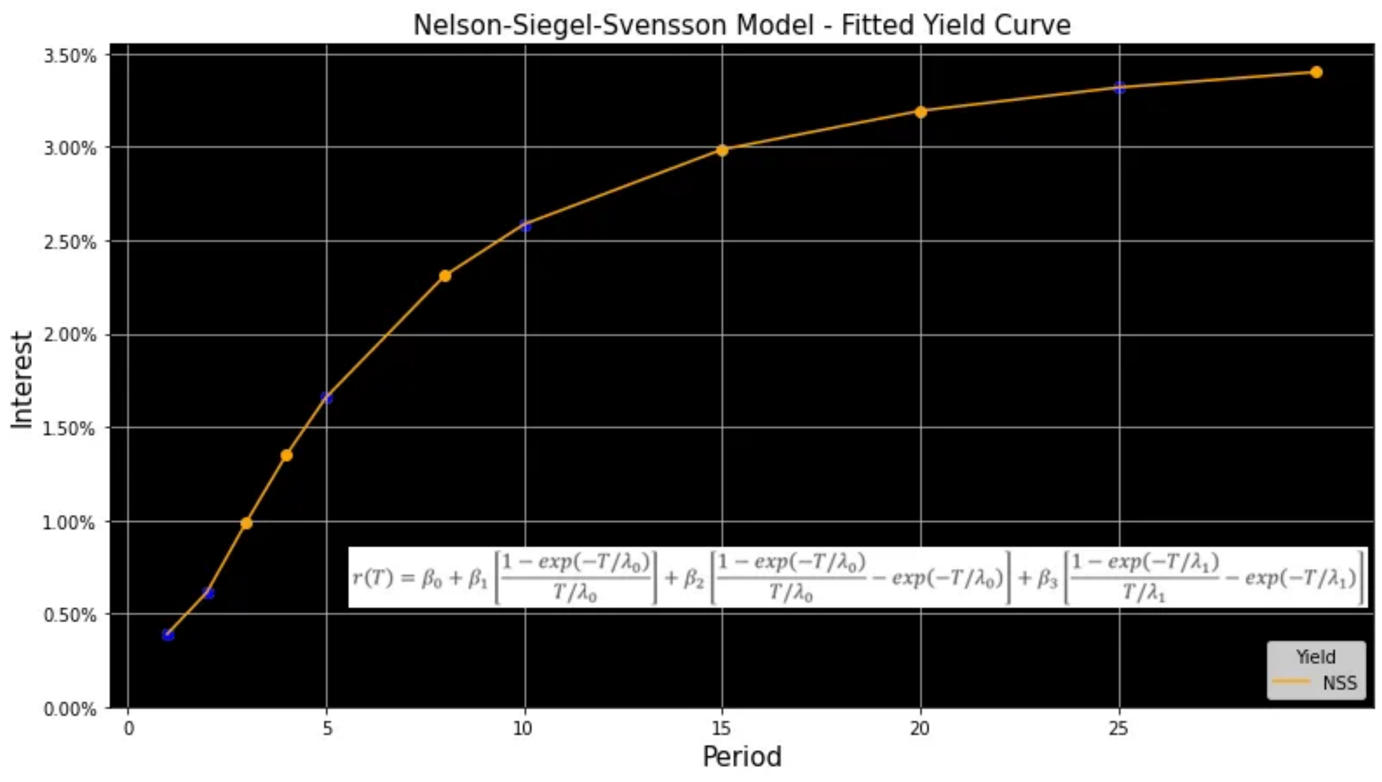


Figure 1

Interpolation and Extrapolation

Sometimes there are missing values in a time-series data set. For instance, interest rates for years 1 to 3 may exist, followed by years 5 to 8, and then year 10. Nelson-Siegel model can be used to also be used to forecast or extrapolate values of future time periods beyond the time period of available data.

The known X values represent the values on axis are the values that are known in advance such as time or years) and the known Y values represent the values on the y-axis (in our case, the known Interest Rates). The y-axis variable is typically the variable you wish to interpolate

Nelson-Siegel-Svensson (1994) Model

The Nelson-Siegel-Svensson (1994) model is used for generating the term structure of interest rates and yield curve estimation. Econometric modeling techniques are required to calibrate the values of several inputs in this model.

The Nelson-Siegel-Svensson (1994) model modifies the Nelson-Siegel (1987) model by adding additional generalized parameters. Virtualy any yield curve shape can be interpolated using these two models, which are widely used at banks around the world.

The Nelson-Siegel-Svensson (1994) model is run with 6 curve estimation parameters, while the Nelson-Siegel (1987) model is run with 4 curve estimation parameters. If properly modeled, the Nelson-Siegel-Svensson (1994) model can be made to fit almost any yield curve shape.

Calibrating the inputs in the Nelson-Siegel-Svensson (1994) model requires facility with econometric modeling and error optimization techniques. The Nelson-Siegel-Svensson (1994) model is used to “fill in the gaps” of missing spot yields and term structure of interest rates whereby the model can be used to both interpolate missing data points within a time series of interest rates (as well as other macroeconomic variables such as inflation rates and commodity prices or market returns) and also used to extrapolate outside of the given or known range, useful for forecasting purposes.

Nelson-Siegel-Svensson Yield Curve Fit Method

The Nelson-Siegel method is famous for its simplicity, but it may fail to match the observed zero yields for all maturities in a stressed market environment.

$$r(T) = \beta_0 + \beta_1 \left[\frac{1 - exp(-T/\lambda)}{T/\lambda}\right] + \beta_2 \left[\frac{1 - exp(-T/\lambda)}{T/\lambda} - exp(-T/\lambda)\right]$$

Figure 2

In 1994 Svensson tried to create a more flexible version by adding an additional term to the existing Nelson-Siegel formula that contained two extra parameters. His nice and simple idea was to create the additional term by reusing the last term of his predecessors' equation.

This is the final formula:

$$r(T) = \beta_0 + \beta_1 \left[\frac{1 - exp(-T/\lambda_0)}{T/\lambda_0}\right] + \beta_2 \left[\frac{1 - exp(-T/\lambda_0)}{T/\lambda_0} - exp(-T/\lambda_0)\right] + \beta_3 \left[\frac{1 - exp(-T/\lambda_1)}{T/\lambda_1} - exp(-T/\lambda_1)\right]$$

Figure 3

Where ***β0, β1, β2, β3,* λ0** and **λ1** are the constant parameters and ***T*** is the time to maturity in annual units.

Before we get into any python let's just describe our problem.

The Underlying Problem

We live in a country called Israel (Israel is somewhere in the middle east and it's capital city called Jerusalem). In Jerusalem is located Israel's central bank called the bank of Israel.

The bank of Israel provides the standard riskless yield curve and is the bank which has the lowest amount of risk in the country (therefore, it provides the standard yield curve. Let's say, for example, that the bank of Israel only has a certain number of bonds.

Let's see what bonds that the government of Israel has outstanding at the moment. So it has:

- 1-year bond and that's currently at a yield of 0.39%.
- 2-year bond and that's currently at a yield of 0.61%.
- 5-year bond and that's currently at a yield of 1.66%.
- 10-year bond and that's currently at a yield of 2.58%.
- 25-year bond and that's currently at a yield of 3.32%.

Well,that's fine the only problem with that, though is that if we built up a yield curve from those figures the problem is we don't have any figures in between. We would like to get a nice yield curve.

Why do we want to get a full nice curve? say, for example, that the reason why is because we have a client who has contacted us in Israel, and in order to get 10,000,000 Israeli shekels (ILS) profit from this particular client we need to have a really good estimate for two things:

- 20-year yield for the bank of Israel.
- 30-year yield for the bank of Israel.

Unfortunately, of course as we can see from our list, we don't have those figures we have got the 10-year and we have got the 25-year but we haven't got the 20-year and we haven't got the 30-year. So, are we going to wave bye-bye to the ILS 10,000,000 profit from the client? or are we going to have a really good educated attempt to try to create those missing bits and pieces on a particular yield curve?

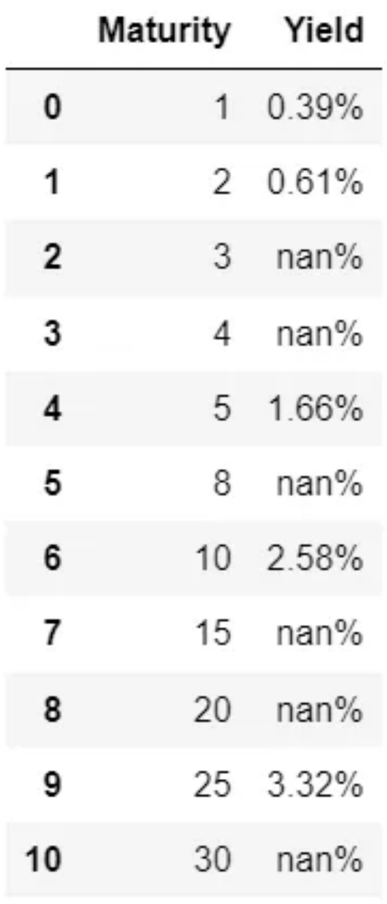
Well, we can. We can use a thing called the Nelson-Siegel-Svensson method to generate really good educated guesses as to what those bits in between should be and what those points extrapolated should be.

The Jupyter Notebook

Let's create a new jupyter notebook.

```
from scipy.optimize import fmin
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dd = pd.read_csv('ns.csv')
df = dd.copy()
df.style.format({'Maturity': '{:,.0f}'.format,'Yield': '{:,.2%}'})
```



	Maturity	Yield
0	1	0.39%
1	2	0.61%
2	3	nan%
3	4	nan%
4	5	1.66%
5	8	nan%
6	10	2.58%
7	15	nan%
8	20	nan%
9	25	3.32%
10	30	nan%

Figure 4

You will notice we don't have the 20-year which we would like to interpolate, we don't have the 30-year which we would like to extrapolate because if we can extrapolate those things then we are going to get those figures calculated. We are going to be able to make the ILS 10,000,000 profit and that is what we would like to do.

```
sf = df.copy()
sf = sf.dropna()
sf1 = sf.copy()
sf1['Y'] = round(sf['Yield']*100,4)
sf = sf.style.format({'Maturity': '{:,.2f}'.format,'Yield': '{:,.4%}'})
import matplotlib.pyplot as plt
import matplotlib.markers as mk
import matplotlib.ticker as mtick
fontsize=15
```

```
    fig = plt.figure(figsize=(13,7))
    plt.title("Nelson-Siegel-Svensson Model - Unfitted Yield Curve",fontsize=fon
    ax = plt.axes()
    ax.set_facecolor("black")
    fig.patch.set_facecolor('white')
    X = sf1["Maturity"]
    Y = sf1["Y"]
    plt.scatter(X, Y, marker="o", c="blue")
    plt.xlabel('Period',fontsize=fontsize)
    plt.ylabel('Interest',fontsize=fontsize)
    ax.yaxis.set_major_formatter(mtick.PercentFormatter())
    ax.xaxis.set_ticks(np.arange(0, 30, 5))
    ax.yaxis.set_ticks(np.arange(0, 4, 0.5))
    ax.legend(loc="lower right", title="Yield")
    plt.grid()
    plt.show()
```

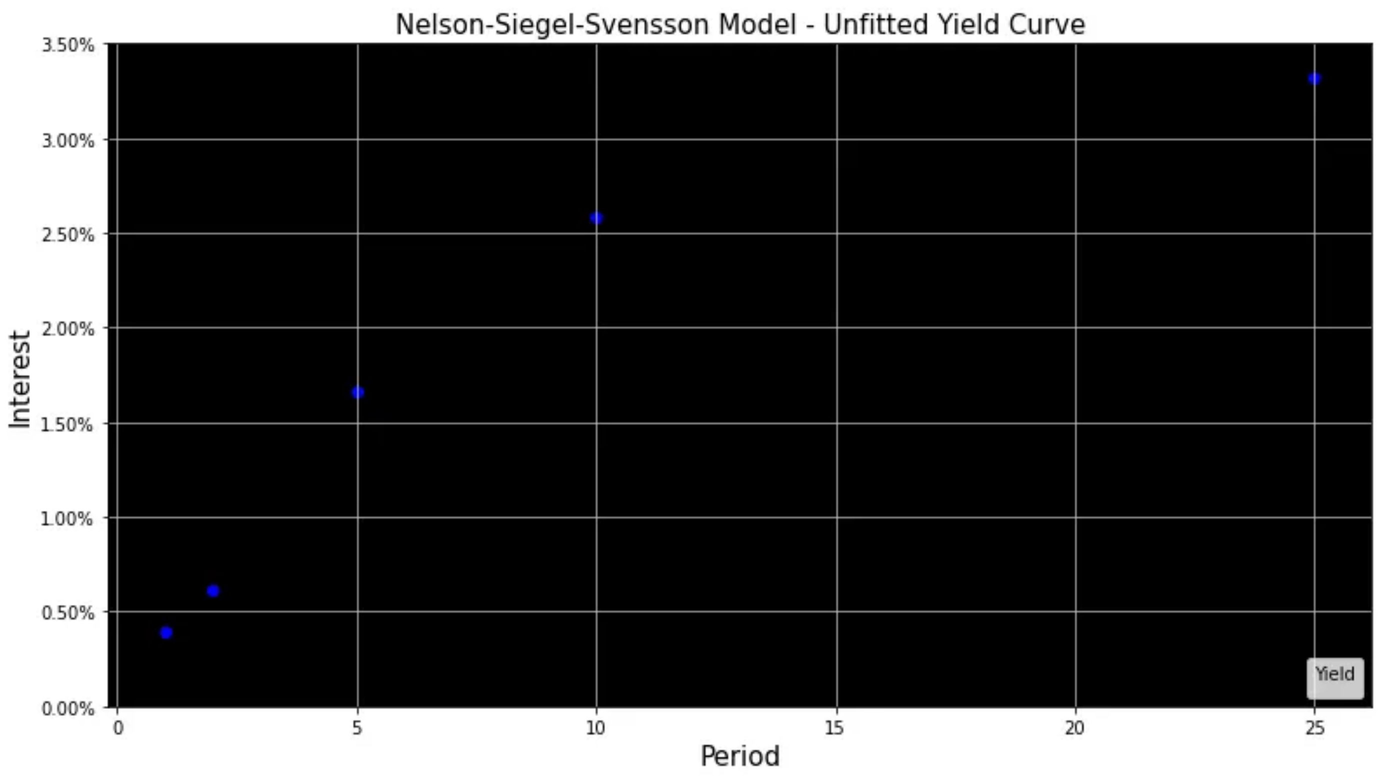


Figure 5

So, you can see we have got this kind of raggedy yield curve here. We need to know the 20-year and the 30-year. Let’s set see how we are going to get this Nelson-Siegel-Svensson method.

The first thing we need to do is to put in 3 calibration figures. If you search for the Nelson-Siegel-Svensson Model on the internet you will find these calibration figures you can put in.

I would suggest that these are pretty good guesses to get going with and so for the 4 beta figures put 0.01 in for the 2 lambda radioactive decay style figures I would suggest that 1.0 is a good starting value. Now these are going to be changed by a function called scipy.optimize.fmin.

```
β0 = 0.01
β1 = 0.01
β2 = 0.01
β3 = 0.01
λ0 = 1.00
λ1 = 1.00
```

What we are going to do now is to build a curve which won't be any good but which will be changed later by special piece of scipy.optimize magic called the fmin function. I need to build up another curve in a new column on my dataframe called "NSS" and when I build up another curve in this column that magic piece of technological solve will move these around to make the yield curve (that is the "Yield" column on my dataframe) fit the above figures.

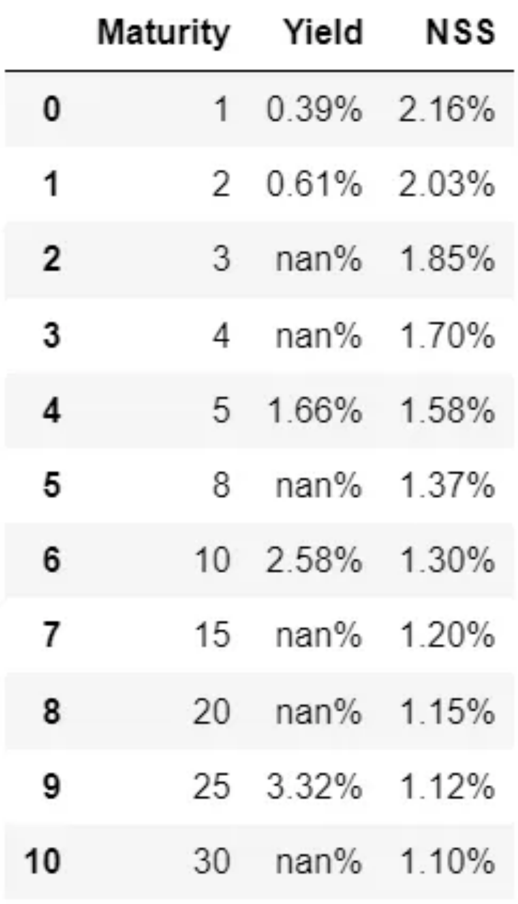
Let's put the Nelson-Siegel-Svensson formula in here that we are going to be working now

$$r(T) = \beta_0 + \beta_1 \left[\frac{1 - exp(-T/\lambda_0)}{T/\lambda_0}\right] + \beta_2 \left[\frac{1 - exp(-T/\lambda_0)}{T/\lambda_0} - exp(-T/\lambda_0)\right] + \beta_3 \left[\frac{1 - exp(-T/\lambda_1)}{T/\lambda_1} - exp(-T/\lambda_1)\right]$$

Figure 5

As you can see it's a bit of a monster.

```
df['NSS'] = (β0)+(β1*((1-np.exp(-df['Maturity']/λ0))/(df['Maturity']/λ0)))+(
df.style.format({'Maturity': '{:,.0f}'.format,'Yield': '{:,.2%}','NSS': '{:,
```



	Maturity	Yield	NSS
0	1	0.39%	2.16%
1	2	0.61%	2.03%
2	3	nan%	1.85%
3	4	nan%	1.70%
4	5	1.66%	1.58%
5	8	nan%	1.37%
6	10	2.58%	1.30%
7	15	nan%	1.20%
8	20	nan%	1.15%
9	25	3.32%	1.12%
10	30	nan%	1.10%

Figure 6

Let's visualize our unfitted yield curve

```
df1 = df.copy()
df['Y'] = round(df['Yield']*100,4)
df['NSS'] =(β0)+(β1*((1-np.exp(-df['Maturity']/λ0))/(df['Maturity']/λ0)))+(β
df['N'] = round(df['NSS']*100,4)
df2 = df.copy()
df2 = df2.style.format({'Maturity': '{:,.2f}'.format,'Y': '{:,.2%}', 'N': '{
import matplotlib.pyplot as plt
import matplotlib.markers as mk
import matplotlib.ticker as mtick
fontsize=15
fig = plt.figure(figsize=(13,7))
plt.title("Nelson-Siegel-Svensson Model - Unfitted Yield Curve",fontsize=fon
ax = plt.axes()
ax.set_facecolor("black")
fig.patch.set_facecolor('white')
X = df["Maturity"]
Y = df["Y"]
x = df["Maturity"]
y = df["N"]
ax.plot(x, y, color="orange", label="NSS")
plt.scatter(x, y, marker="o", c="orange")
plt.scatter(X, Y, marker="o", c="blue")
plt.xlabel('Period',fontsize=fontsize)
plt.ylabel('Interest',fontsize=fontsize)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.xaxis.set_ticks(np.arange(0, 30, 5))
ax.yaxis.set_ticks(np.arange(0, 4, 0.5))
ax.legend(loc="lower right", title="Yield")
plt.grid()
plt.show()
```

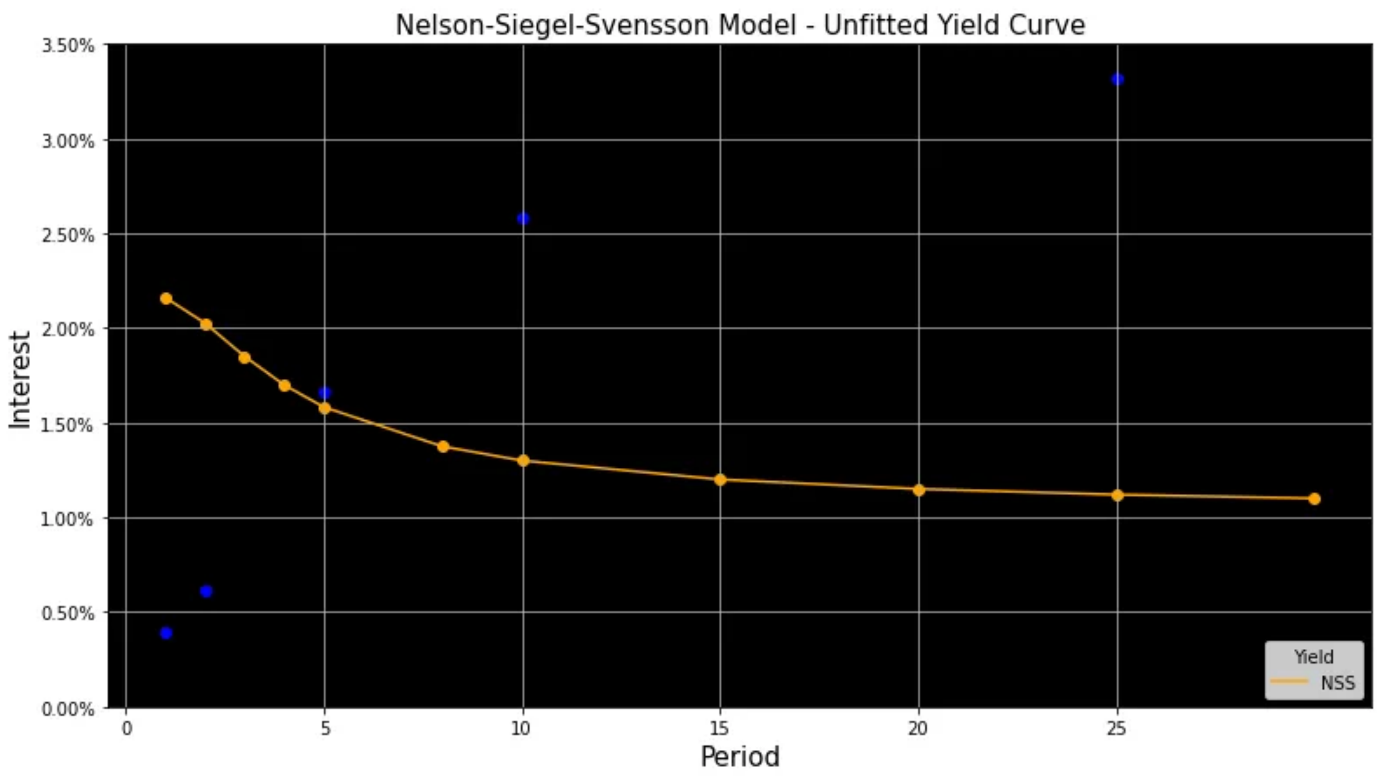


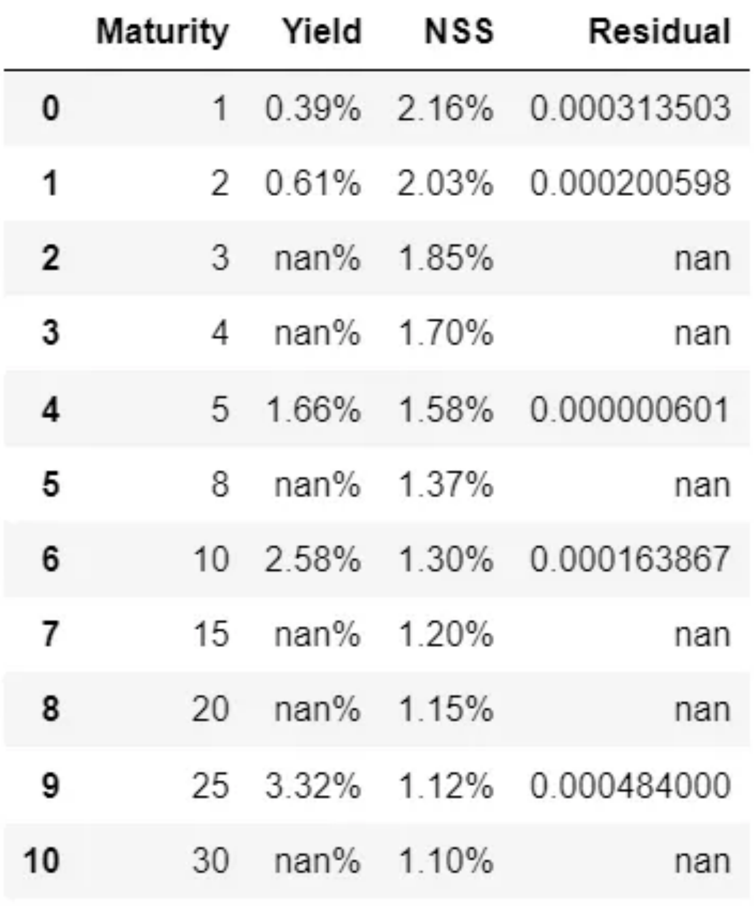
Figure 7

What we have now got is we have now gone unfitted curve this is like the material from a suit before we make the suit, we are going to do some tailoring with these calibration statistics and we are going to make that suit fit the blue points and we're going to use the special fmin function to be able to do that. Before we can do that, I need to build up another curve in a new column on my dataframe called "Residual".

We take a thing called special residuals we are going to take some errors between the blue points and the unfitted curve. We are going to take these errors and then we are going to square those errors to get rid of any negative signs.

```
df['Residual'] =  (df['Yield'] - df['NSS'])**2
df22 = df[['Maturity','Yield','NSS','Residual']]
```

```
df22.style.format({'Maturity': '{:,.0f}'.format,'Yield': '{:,.2%}','NSS': '{
```



	Maturity	Yield	NSS	Residual
0	1	0.39%	2.16%	0.000313503
1	2	0.61%	2.03%	0.000200598
2	3	nan%	1.85%	nan
3	4	nan%	1.70%	nan
4	5	1.66%	1.58%	0.000000601
5	8	nan%	1.37%	nan
6	10	2.58%	1.30%	0.000163867
7	15	nan%	1.20%	nan
8	20	nan%	1.15%	nan
9	25	3.32%	1.12%	0.000484000
10	30	nan%	1.10%	nan

Figure 8

Now we take all of these error terms which exist between the blue points and the orange points where the blue points exist so we take those terms we square them. Then what I am going to do is I am going to add them all up.

```
np.sum(df['Residual'])
```

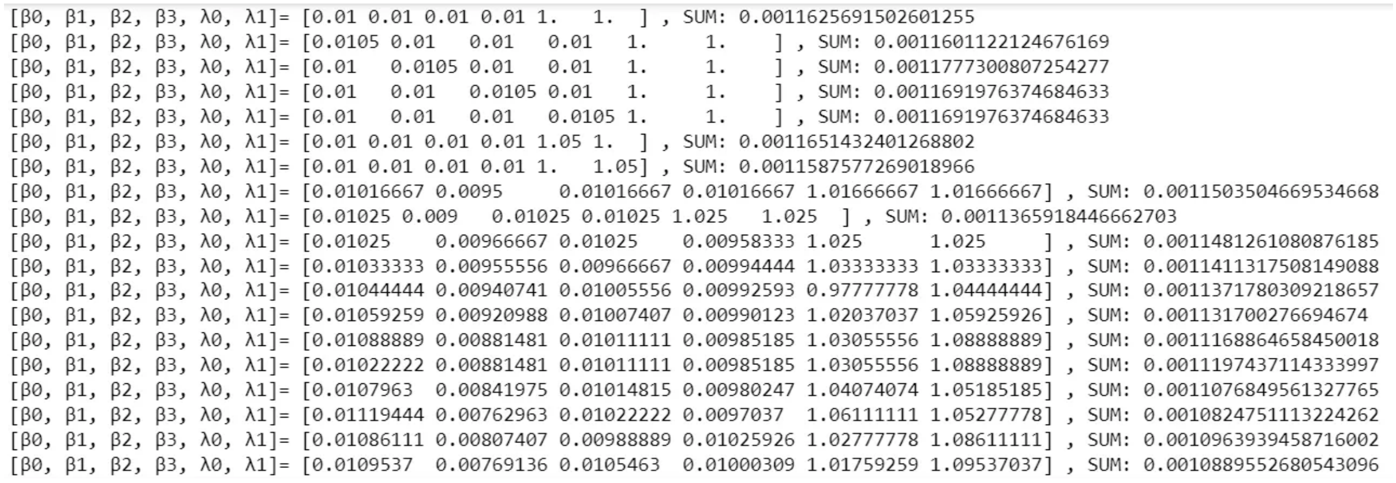
0.0011625691502601255

Now here is where the magic comes in. What I am going to do is I am going to call the fmin function and I am going to run it over the above 6 values and the solve is going to jiggle these things up and down, up and down, up and down, up and down and that is going to make the values on the "NSS" column go up and down, up and down, up and down and we are going to try to minimize the values on the "Residual" column to make them collectively add up to as low a number as possible and in doing so hopefully they will make this orange line fit this broken blue line.

We want to minimize that sum of the "Residual" column to make it as small as possible by changing the above 6 figures. Here we are okay with negative numbers no problem there just goes with the default kind of algorithm set up and then call the fmin.

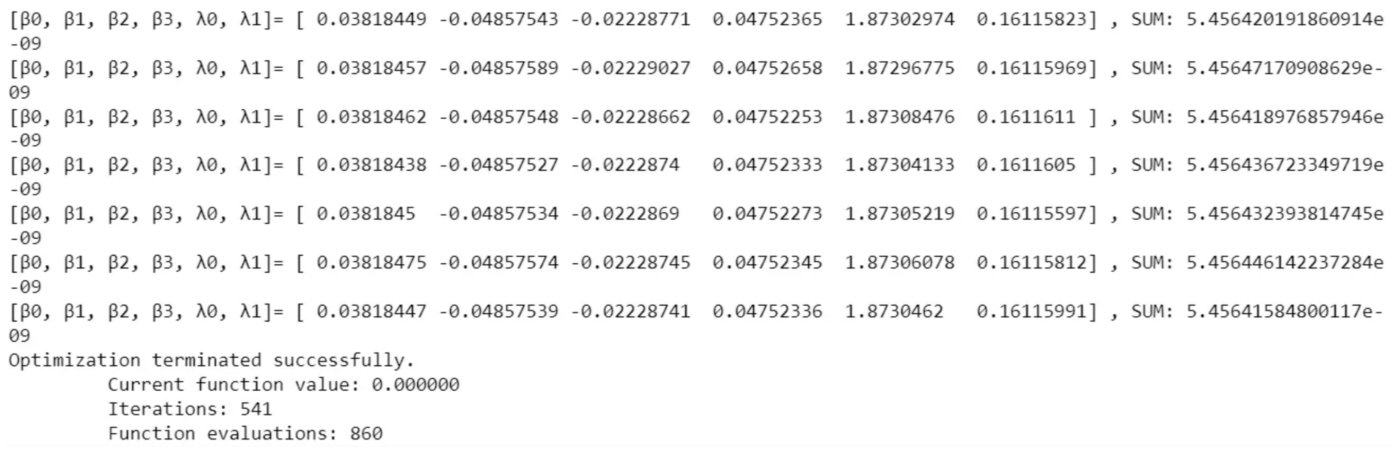
```
def myval(c):
    df = dd.copy()
    df['NSS'] =(c[0])+(c[1]*((1-np.exp(-df['Maturity']/c[4]))/(df['Maturity'
    df['Residual'] =  (df['Yield'] - df['NSS'])**2
    val = np.sum(df['Residual'])
    print("[β0, β1, β2, β3, λ0, λ1]=",c,", SUM:", val)
    return(val)

c = fmin(myval, [0.01, 0.01, 0.01, 0.01, 1.00, 1.00])
```



```
[β0, β1, β2, β3, λ0, λ1]= [0.01 0.01 0.01 0.01 1.   1.  ] , SUM: 0.0011625691502601255
[β0, β1, β2, β3, λ0, λ1]= [0.0105 0.01   0.01   0.01   1.     1.    ] , SUM: 0.0011601122124676169
[β0, β1, β2, β3, λ0, λ1]= [0.01   0.0105 0.01   0.01   1.     1.    ] , SUM: 0.0011777300807254277
[β0, β1, β2, β3, λ0, λ1]= [0.01   0.01   0.0105 0.01   1.     1.    ] , SUM: 0.0011691976374684633
[β0, β1, β2, β3, λ0, λ1]= [0.01   0.01   0.01   0.0105 1.     1.    ] , SUM: 0.0011691976374684633
[β0, β1, β2, β3, λ0, λ1]= [0.01 0.01 0.01 0.01 1.05 1.  ] , SUM: 0.0011651432401268802
[β0, β1, β2, β3, λ0, λ1]= [0.01 0.01 0.01 0.01 1.   1.05] , SUM: 0.0011587577269018966
[β0, β1, β2, β3, λ0, λ1]= [0.01016667 0.0095     0.01016667 0.01016667 1.01666667 1.01666667] , SUM: 0.0011503504669534668
[β0, β1, β2, β3, λ0, λ1]= [0.01025 0.009   0.01025 0.01025 1.025   1.025  ] , SUM: 0.0011365918446662703
[β0, β1, β2, β3, λ0, λ1]= [0.01025    0.00966667 0.01025    0.00958333 1.025      1.025     ] , SUM: 0.0011481261080876185
[β0, β1, β2, β3, λ0, λ1]= [0.01033333 0.00955556 0.00966667 0.00994444 1.03333333 1.03333333] , SUM: 0.0011411317508149088
[β0, β1, β2, β3, λ0, λ1]= [0.01044444 0.00940741 0.01005556 0.00992593 0.97777778 1.04444444] , SUM: 0.0011371780309218657
[β0, β1, β2, β3, λ0, λ1]= [0.01059259 0.00920988 0.01007407 0.00990123 1.02037037 1.05925926] , SUM: 0.001131700276694674
[β0, β1, β2, β3, λ0, λ1]= [0.01088889 0.00881481 0.01011111 0.00985185 1.03055556 1.08888889] , SUM: 0.0011168864658450018
[β0, β1, β2, β3, λ0, λ1]= [0.01022222 0.00881481 0.01011111 0.00985185 1.03055556 1.08888889] , SUM: 0.0011197437114333997
[β0, β1, β2, β3, λ0, λ1]= [0.0107963  0.00841975 0.01014815 0.00980247 1.04074074 1.05185185] , SUM: 0.0011076849561327765
[β0, β1, β2, β3, λ0, λ1]= [0.01119444 0.00762963 0.01022222 0.0097037  1.06111111 1.05277778] , SUM: 0.0010824751113224262
[β0, β1, β2, β3, λ0, λ1]= [0.01086111 0.00807407 0.00988889 0.01025926 1.02777778 1.08611111] , SUM: 0.0010963939458716002
[β0, β1, β2, β3, λ0, λ1]= [0.0109537  0.00769136 0.0105463  0.01000309 1.01759259 1.09537037] , SUM: 0.0010889552680543096
```

Figure 9



```
[β0, β1, β2, β3, λ0, λ1]= [ 0.03818449 -0.04857543 -0.02228771  0.04752365  1.87302974  0.16115823] , SUM: 5.456420191860914e
-09
[β0, β1, β2, β3, λ0, λ1]= [ 0.03818457 -0.04857589 -0.02229027  0.04752658  1.87296775  0.16115969] , SUM: 5.45647170908629e-
09
[β0, β1, β2, β3, λ0, λ1]= [ 0.03818462 -0.04857548 -0.02228662  0.04752253  1.87308476  0.1611611 ] , SUM: 5.456418976857946e
-09
[β0, β1, β2, β3, λ0, λ1]= [ 0.03818438 -0.04857527 -0.0222874   0.04752333  1.87304133  0.1611605 ] , SUM: 5.456436723349719e
-09
[β0, β1, β2, β3, λ0, λ1]= [ 0.0381845  -0.04857534 -0.0222869   0.04752273  1.87305219  0.16115597] , SUM: 5.456432393814745e
-09
[β0, β1, β2, β3, λ0, λ1]= [ 0.03818475 -0.04857574 -0.02228745  0.04752345  1.87306078  0.16115812] , SUM: 5.456446142237284e
-09
[β0, β1, β2, β3, λ0, λ1]= [ 0.03818447 -0.04857539 -0.02228741  0.04752336  1.8730462   0.16115991] , SUM: 5.45641584800117e-
09
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 541
         Function evaluations: 860
```

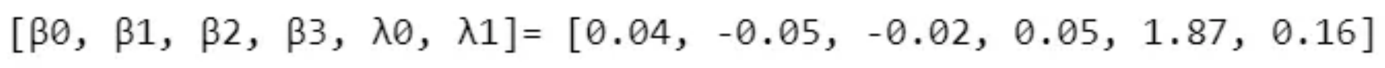
Figure 10

What's happened is that the solver jiggled the above 6 figures up and down from my original inputs and it made the orange curve fit those blue points that we saw earlier. Isn't that a fantastic piece of magic.

Now, let's get the results for our 6 figures

```
β0 = c[0]
β1 = c[1]
β2 = c[2]
β3 = c[3]
λ0 = c[4]
```

```
    λ1 = c[5]
    print("[β0, β1, β2, β3, λ0, λ1]=", [c[0].round(2), c[1].round(2), c[2].round
```



```
[β0, β1, β2, β3, λ0, λ1]= [0.04, -0.05, -0.02, 0.05, 1.87, 0.16]
```

Figure 11

Let’s visualize our fitted yield curve

```
    df = df1.copy()
    df['NSS'] =(β0)+(β1*((1-np.exp(-df['Maturity']/λ0))/(df['Maturity']/λ0)))+(β
    sf4 = df.copy()
    sf5 = sf4.copy()
    sf5['Y'] = round(sf4['Yield']*100,4)
    sf5['N'] = round(sf4['NSS']*100,4)
    sf4 = sf4.style.format({'Maturity': '{:,.2f}'.format,'Yield': '{:,.2%}', 'NS
    M0 = 0.00
    M1 = 3.50
    import matplotlib.pyplot as plt
    import matplotlib.markers as mk
    import matplotlib.ticker as mtick
    fontsize=15
    fig = plt.figure(figsize=(13,7))
    plt.title("Nelson-Siegel-Svensson Model - Fitted Yield Curve",fontsize=fonts
    ax = plt.axes()
    ax.set_facecolor("black")
    fig.patch.set_facecolor('white')
    X = sf5["Maturity"]
    Y = sf5["Y"]
    x = sf5["Maturity"]
    y = sf5["N"]
    ax.plot(x, y, color="orange", label="NSS")
    plt.scatter(x, y, marker="o", c="orange")
    plt.scatter(X, Y, marker="o", c="blue")
    plt.xlabel('Period',fontsize=fontsize)
    plt.ylabel('Interest',fontsize=fontsize)
    ax.yaxis.set_major_formatter(mtick.PercentFormatter())
    ax.xaxis.set_ticks(np.arange(0, 30, 5))
```

```
ax.yaxis.set_ticks(np.arange(0, 4, 0.5))
ax.legend(loc="lower right", title="Yield")
plt.grid()
plt.show()
```

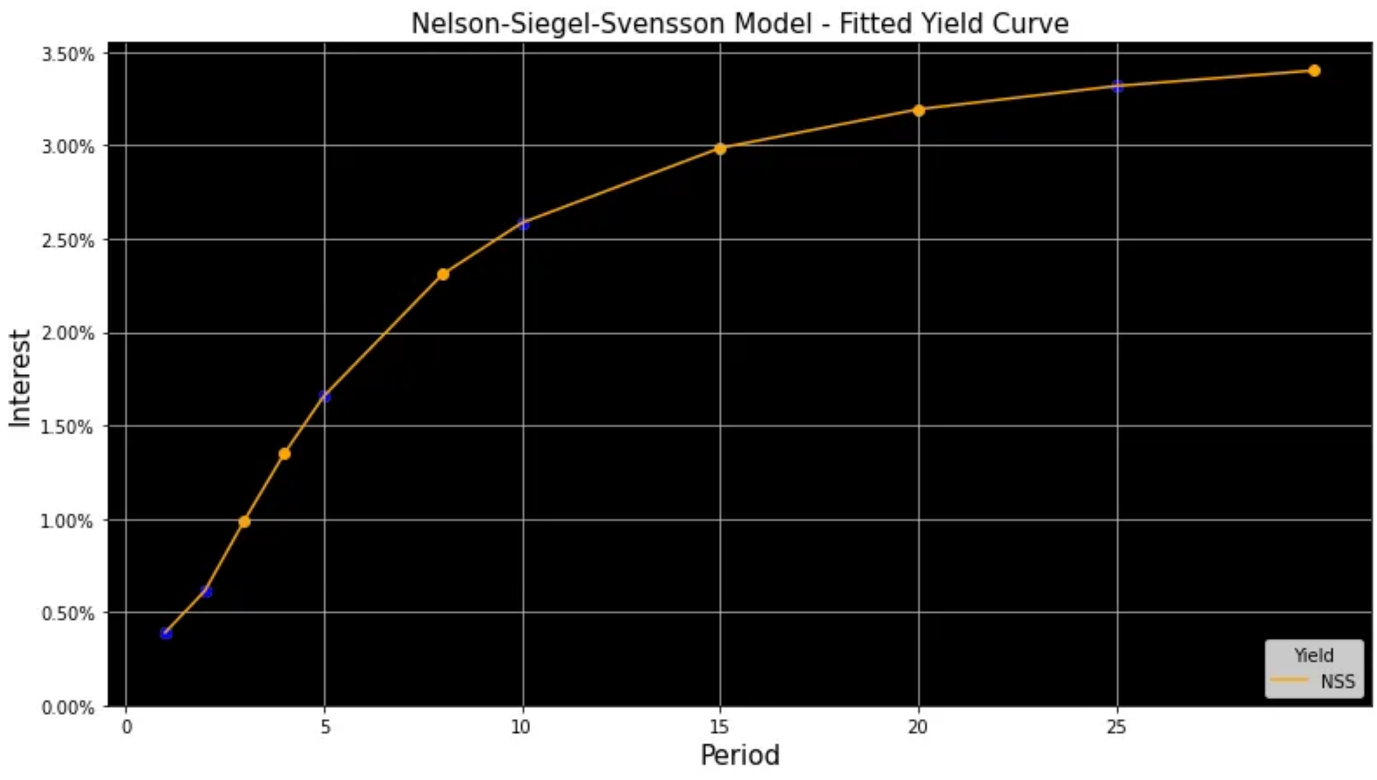
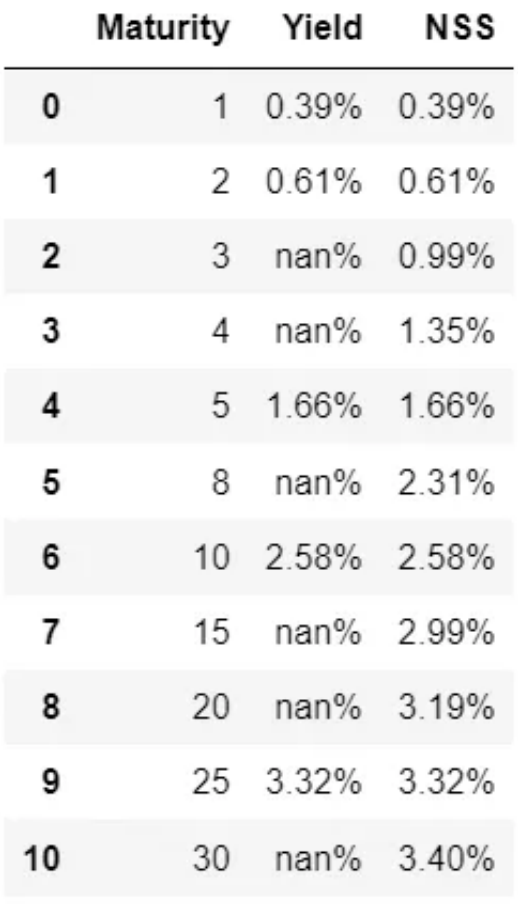


Figure 12

This Nelson-Siegel-Svensson's equation is brilliant it's probably the best one of these kinds of equations it is better than the polynomial equation for creating these non-linear lines and these curves and most central banks and software draws these kinds of lines is using this Nelson-Siegel-Svensson method to create these line.

```
df.style.format({'Maturity': '{:,.0f}'.format,'Yield': '{:,.2%}','NSS': '{:,
```



	Maturity	Yield	NSS
0	1	0.39%	0.39%
1	2	0.61%	0.61%
2	3	nan%	0.99%
3	4	nan%	1.35%
4	5	1.66%	1.66%
5	8	nan%	2.31%
6	10	2.58%	2.58%
7	15	nan%	2.99%
8	20	nan%	3.19%
9	25	3.32%	3.32%
10	30	nan%	3.40%

Figure 13

You can see how well that has done to create a full yield curve and now I can see the 20-year probable yield which is 3.2% and I have extrapolated the probable 30-year yield of 3.4%. So, I think that's pretty good what that means is we can now go for this ILS 10,000,000 profit.

This is how you implement the Nelson-Siegel-Svensson method in order to create yield curves from limited information.

Your Turn!

Hopefully, this post gives you a good idea of how to generate a yield curve with the Nelson-Siegel-Svensson method. As you can see, it's really easy, simple and fast.

Now it's time to get out there and start interpolating and extrapolating your data. Try this algorithm, and let me know how it goes.

I would be pleased to receive feedback or questions on any of the above.

About the Author

Roi Polanitzer, FRM, F.IL.A.V.F.A., QFV

Roi Polanitzer, CFV, QFV, FEM, F.IL.A.V.F.A., FRM, CRM, PDS, is a well-known authority in Israel the field of business valuation and has written hundreds of papers that articulate many of the concepts used in modern business valuation around the world. Mr. Polanitzer is the Owner and Chief Appraiser of Intrinsic Value — Independent Business Appraisers, a business valuation firm headquartered in Rishon LeZion, Israel. He is also the Owner and Chief Data Scientist of Prediction Consultants, a consulting firm that specializes in advanced analysis and model development.