# Nelson-Siegel-Svensson Model Documentation

***Release 0.5.0***

**luphord**

**Nov 13, 2022**

# Contents:

# CHAPTER 1

# Nelson-Siegel-Svensson Model

Implementation of the Nelson-Siegel-Svensson interest rate curve model in Python.

```
from nelson_siegel_svensson import NelsonSiegelSvenssonCurve
import numpy as np
from matplotlib.pyplot import plot

y = NelsonSiegelSvenssonCurve(0.028, -0.03, -0.04, -0.015, 1.1, 4.0)
t = np.linspace(0, 20, 100)
plot(t, y(t))
```


- Free software: MIT license
- Python 3.7 or later supported
- Documentation: https://nelson-siegel-svensson.readthedocs.io.

## 1.1 Features

- Python implementation of the Nelson-Siegel curve (three factors)
- Python implementation of the Nelson-Siegel-Svensson curve (four factors)

- Methods for zero and forward rates (as vectorized functions of time points)
- Methods for the factors (as vectorized function of time points)
- Calibration based on ordinary least squares (OLS) for betas and nonlinear optimization for taus
- Simple command line interface (CLI) for evaluating, calibrating and plotting curves

## 1.2 Calibration

In order to calibrate a curve to given data you can use the *calibrate_ns_ols* and *calibrate_nss_ols* functions in the *calibrate* module:

```
import numpy as np
from nelson_siegel_svensson.calibrate import calibrate_ns_ols

t = np.array([0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0])
y = np.array([0.01, 0.011, 0.013, 0.016, 0.019, 0.021, 0.026, 0.03, 0.035, 0.037, 0.
↪038, 0.04])

curve, status = calibrate_ns_ols(t, y, tau0=1.0)  # starting value of 1.0 for the␣
↪optimization of tau
assert status.success
print(curve)
```

which gives the following output:

```
NelsonSiegelCurve(beta0=0.04201739383636799, beta1=-0.031829031569430594, beta2=-0.
↪026797319779108236, tau=1.7170972656534174)
```

docs/_static/calibrated_nelson-siegel-curve.png

## 1.3 Command Line interface

*nelson_siegel_svensson* provides basic functionality using a command line interface (CLI):

```
Usage: nelson_siegel_svensson [OPTIONS] COMMAND [ARGS]...

Commandline interface for nelson_siegel_svensson.

Options:
--help  Show this message and exit.

Commands:
calibrate  Calibrate a curve to the given data points.
evaluate   Evaluate a curve at given points.
plot       Plot a curve at given points.
```

In order to calibrate a curve to given data points on the command line, try

```
nelson_siegel_svensson calibrate -t '[0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0, 10.0, 15.0,␣
↪20.0, 25.0, 30.0]' -y '[0.01, 0.011, 0.013, 0.016, 0.019, 0.021, 0.026, 0.03, 0.035,
↪ 0.037, 0.038, 0.04]' --nelson-siegel --initial-tau1 1.0
```

which gives

```
{"beta0": 0.042017393764903765, "beta1": -0.03182903146166806, "beta2": -0.
↪026797320316066128, "tau": 1.717097232403383}
```

This curve can then be evaluated on the command line using

```
nelson_siegel_svensson evaluate -c '{"beta0": 0.042017393764903765, "beta1": -0.
↪03182903146166806, "beta2": -0.026797320316066128, "tau": 1.717097232403383}' -t
↪'[0, 1, 2, 3]'
```

resulting in

```
[0.010188362303235707, 0.012547870204470839, 0.01574855552855885, 0.01897955804146046]
```

And finally, the curve can be plotted with

```
nelson_siegel_svensson plot -o cli_plot_example.png -c '{"beta0": 0.
↪042017393764903765, "beta1": -0.03182903146166806, "beta2": -0.026797320316066128,
↪"tau": 1.717097232403383}'
```

docs/_static/cli_plot_example.png

Note that the quoting in the above commands prevents *bash* from evalutating the JSON-based parameters. Depending on your shell, you may require a different quoting mechanism.

## 1.4 Credits

Main developer is luphord.

This package was prepared with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

# Installation

## 2.1 Stable release

To install Nelson-Siegel-Svensson Model, run this command in your terminal:

```
$ pip install nelson_siegel_svensson
```

This is the preferred method to install Nelson-Siegel-Svensson Model, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

## 2.2 From sources

The sources for Nelson-Siegel-Svensson Model can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/luphord/nelson_siegel_svensson
```

Or download the tarball:

```
$ curl  -OL https://github.com/luphord/nelson_siegel_svensson/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

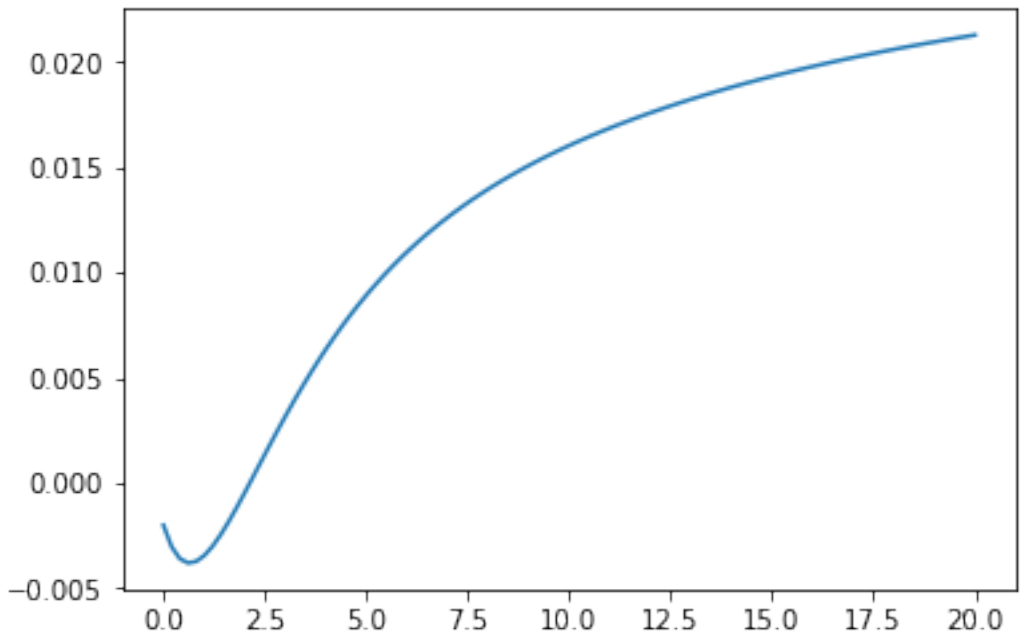
# CHAPTER 3

# Usage

To instantiate and evaluate a Nelson-Siegel-Svensson-Curve simply import the curve class, call the constructor with the desired parameters and then call the resulting curve instance with a numpy array of time points:

```
from nelson_siegel_svensson import NelsonSiegelSvenssonCurve
import numpy as np
from matplotlib.pyplot import plot

y = NelsonSiegelSvenssonCurve(0.028, -0.03, -0.04, -0.015, 1.1, 4.0)
t = np.linspace(0, 20, 100)
plot(t, y(t))
```



In order to calibrate a curve to given data you can use the *calibrate_ns_ols* and *calibrate_nss_ols* functions in the *calibrate* module:

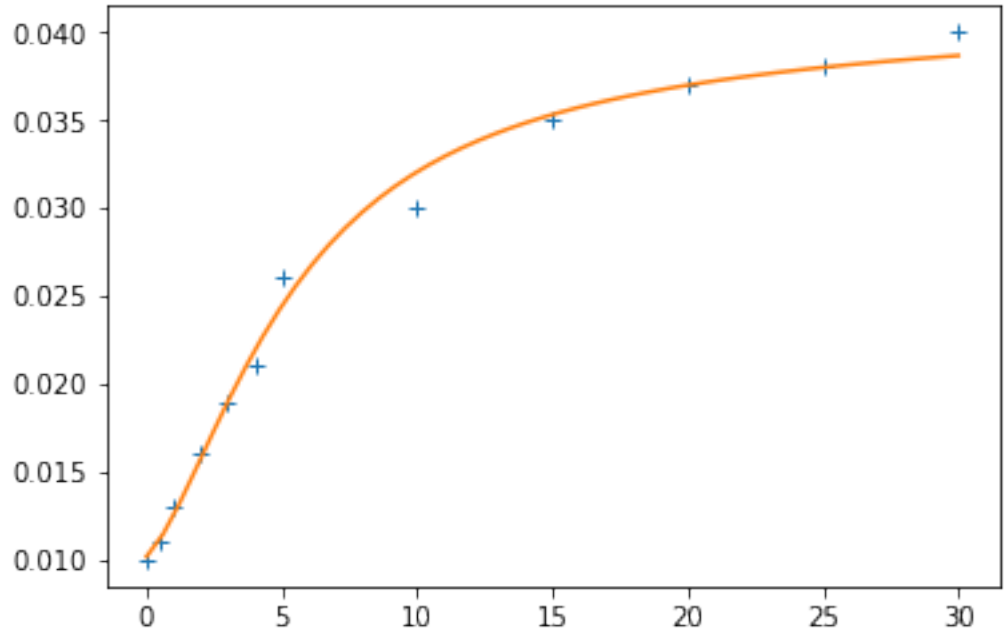
```
import numpy as np
from nelson_siegel_svensson.calibrate import calibrate_ns_ols

t = np.array([0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0])
y = np.array([0.01, 0.011, 0.013, 0.016, 0.019, 0.021, 0.026, 0.03, 0.035, 0.037, 0.
↪038, 0.04])

curve, status = calibrate_ns_ols(t, y, tau0=1.0)  # starting value of 1.0 for the␣
↪optimization of tau
assert status.success
print(curve)
```

which gives the following output:

```
NelsonSiegelCurve(beta0=0.04201739383636799, beta1=-0.031829031569430594, beta2=-0.
↪026797319779108236, tau=1.7170972656534174)
```



CHAPTER 4

# nelson_siegel_svensson

## 4.1 nelson_siegel_svensson package

### 4.1.1 Submodules

### 4.1.2 nelson_siegel_svensson.calibrate module

Calibration methods for Nelson-Siegel(-Svensson) Models. See *calibrate_ns_ols* and *calibrate_nss_ols* for ordinary least squares (OLS) based methods.

nelson_siegel_svensson.calibrate.**betas_ns_ols**(*tau: float*, *t: numpy.ndarray*, *y: numpy.ndarray*) → Tuple[nelson_siegel_svensson.ns.NelsonSiegelCurve, Any]

Calculate the best-fitting beta-values given tau for time-value pairs t and y and return a corresponding Nelson-Siegel curve instance.

nelson_siegel_svensson.calibrate.**betas_nss_ols**(*tau: Tuple[float, float]*, *t: numpy.ndarray*, *y: numpy.ndarray*) → Tuple[nelson_siegel_svensson.nss.NelsonSiegelSvenssonCurve, Any]

Calculate the best-fitting beta-values given tau (= array of tau1 and tau2) for time-value pairs t and y and return a corresponding Nelson-Siegel-Svensson curve instance.

nelson_siegel_svensson.calibrate.**calibrate_ns_ols**(*t: numpy.ndarray*, *y: numpy.ndarray*, *tau0: float = 2.0*) → Tuple[nelson_siegel_svensson.ns.NelsonSiegelCurve, Any]

Calibrate a Nelson-Siegel curve to time-value pairs t and y, by optimizing tau and chosing all betas using ordinary least squares.

nelson_siegel_svensson.calibrate.**calibrate_nss_ols**(*t: numpy.ndarray*, *y: numpy.ndarray*, *tau0: Tuple[float, float] = (2.0, 5.0)*) → Tuple[nelson_siegel_svensson.nss.NelsonSiegelSvenssonCurv
Any]

Calibrate a Nelson-Siegel-Svensson curve to time-value pairs t and y, by optimizing tau1 and tau2 and chosing all betas using ordinary least squares. This method does not work well regarding the recovery of true parameters.

nelson_siegel_svensson.calibrate.**empirical_factors**(*y_3m: float*, *y_2y: float*, *y_10y: float*) → Tuple[float, float, float]

Calculate the empirical factors according to Diebold and Li (2006).

nelson_siegel_svensson.calibrate.**errorfn_ns_ols**(*tau: float*, *t: numpy.ndarray*, *y: numpy.ndarray*) → float

Sum of squares error function for a Nelson-Siegel model and time-value pairs t and y. All betas are obtained by ordinary least squares given tau.

nelson_siegel_svensson.calibrate.**errorfn_nss_ols**(*tau: Tuple[float, float], t: numpy.ndarray, y: numpy.ndarray*) → float

Sum of squares error function for a Nelson-Siegel-Svensson model and time-value pairs t and y. All betas are obtained by ordinary least squares given tau (= array of tau1 and tau2).

### 4.1.3 nelson_siegel_svensson.cli module

Console script for nelson_siegel_svensson.

**class** nelson_siegel_svensson.cli.**Curve**

Bases: click.types.ParamType

Parameter type representing a curve (either Nelson-Siegel or Nelson-Siegel-Svensson)

**convert**(*value*, *param*, *ctx*)

Convert the value to the correct type. This is not called if the value is None (the missing value).

This must accept string values from the command line, as well as values that are already the correct type. It may also convert other compatible types.

The param and ctx arguments may be None in certain situations, such as when converting prompt input.

If the value cannot be converted, call fail() with a descriptive message.

**Parameters**

- **value** – The value to convert.
- **param** – The parameter that is using this type to convert its value. May be None.
- **ctx** – The current context that arrived at this value. May be None.

**name = 'curve'**

**class** nelson_siegel_svensson.cli.**FloatArray**

Bases: click.types.ParamType

Parameter type representing an array of floats

**convert**(*value*, *param*, *ctx*)

Convert the value to the correct type. This is not called if the value is None (the missing value).

This must accept string values from the command line, as well as values that are already the correct type. It may also convert other compatible types.

The `param` and `ctx` arguments may be `None` in certain situations, such as when converting prompt input.

If the value cannot be converted, call `fail()` with a descriptive message.

**Parameters**

- **value** – The value to convert.
- **param** – The parameter that is using this type to convert its value. May be `None`.
- **ctx** – The current context that arrived at this value. May be `None`.

**name = 'floats'**

### 4.1.4 nelson_siegel_svensson.ns module

Implementation of a Nelson-Siegel interest rate curve model. See *NelsonSiegelCurve* class for details.

**class** nelson_siegel_svensson.ns.**NelsonSiegelCurve**(*beta0: float*, *beta1: float*, *beta2: float*, *tau: float*)

Bases: `object`

Implementation of a Nelson-Siegel interest rate curve model. This curve can be interpreted as a factor model with three factors (including a constant).

**factor_matrix**(*T: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]
Factor loadings for time(s) T as matrix columns, including constant column (=1.0).

**factors**(*T: Union[float, numpy.ndarray]*) → Union[Tuple[float, float], Tuple[numpy.ndarray, numpy.ndarray]]
Factor loadings for time(s) T, excluding constant.

**forward**(*T: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]
Instantaneous forward rate(s) of this curve at time(s) T.

**zero**(*T: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]
Zero rate(s) of this curve at time(s) T.

### 4.1.5 nelson_siegel_svensson.nss module

Implementation of a Nelson-Siegel-Svensson interest rate curve model. See *NelsonSiegelSvenssonCurve* class for details.

**class** nelson_siegel_svensson.nss.**NelsonSiegelSvenssonCurve**(*beta0: float*, *beta1: float*, *beta2: float*, *beta3: float*, *tau1: float*, *tau2: float*)

Bases: `object`

Implementation of a Nelson-Siegel-Svensson interest rate curve model. This curve can be interpreted as a factor model with four factors (including a constant).

**factor_matrix**(*T: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]
Factor loadings for time(s) T as matrix columns, including constant column (=1.0).

**factors**(*T: Union[float, numpy.ndarray]*) → Union[Tuple[float, float, float], Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]]
Factor loadings for time(s) T, excluding constant.

**forward**(*T: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]
Instantaneous forward rate(s) of this curve at time(s) T.

**zero**(*T: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]
Zero rate(s) of this curve at time(s) T.

### 4.1.6 Module contents

Implementation of the Nelson-Siegel-Svensson interest rate curve model. For details, see classes *NelsonSiegelCurve* and *NelsonSiegelSvenssonCurve*.

**class** nelson_siegel_svensson.**NelsonSiegelCurve**(*beta0: float*, *beta1: float*, *beta2: float*, *tau: float*)

Bases: object

Implementation of a Nelson-Siegel interest rate curve model. This curve can be interpreted as a factor model with three factors (including a constant).

**factor_matrix**(*T: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]
Factor loadings for time(s) T as matrix columns, including constant column (=1.0).

**factors**(*T: Union[float, numpy.ndarray]*) → Union[Tuple[float, float], Tuple[numpy.ndarray, numpy.ndarray]]
Factor loadings for time(s) T, excluding constant.

**forward**(*T: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]
Instantaneous forward rate(s) of this curve at time(s) T.

**zero**(*T: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]
Zero rate(s) of this curve at time(s) T.

**class** nelson_siegel_svensson.**NelsonSiegelSvenssonCurve**(*beta0: float*, *beta1: float*, *beta2: float*, *beta3: float*, *tau1: float*, *tau2: float*)

Bases: object

Implementation of a Nelson-Siegel-Svensson interest rate curve model. This curve can be interpreted as a factor model with four factors (including a constant).

**factor_matrix**(*T: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]
Factor loadings for time(s) T as matrix columns, including constant column (=1.0).

**factors**(*T: Union[float, numpy.ndarray]*) → Union[Tuple[float, float, float], Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray]]
Factor loadings for time(s) T, excluding constant.

**forward**(*T: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]
Instantaneous forward rate(s) of this curve at time(s) T.

**zero**(*T: Union[float, numpy.ndarray]*) → Union[float, numpy.ndarray]
Zero rate(s) of this curve at time(s) T.

CHAPTER 5

# Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

## 5.1 Types of Contributions

### 5.1.1 Report Bugs

Report bugs at https://github.com/luphord/nelson_siegel_svensson/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### 5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

### 5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

### 5.1.4 Write Documentation

Nelson-Siegel-Svensson Model could always use more documentation, whether as part of the official Nelson-Siegel-Svensson Model docs, in docstrings, or even on the web in blog posts, articles, and such.

### 5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/luphord/nelson_siegel_svensson/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

## 5.2 Get Started!

Ready to contribute? Here's how to set up *nelson_siegel_svensson* for local development.

1. Fork the *nelson_siegel_svensson* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/nelson_siegel_svensson.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv nelson_siegel_svensson
$ cd nelson_siegel_svensson/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 nelson_siegel_svensson tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

## 5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.7, 3.8 and 3.6, and for PyPy. Check https://travis-ci.com/github/luphord/nelson_siegel_svensson/pull_requests and make sure that the tests pass for all supported Python versions.

## 5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_nelson_siegel_svensson
```

## 5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

# Credits

## 6.1 Development Lead

- luphord <luphord@protonmail.com>

## 6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

# History

## 7.1 0.5.0 (2022-11-13)

- Drop support for Python 3.7
- Add support Python 3.9 and 3.10
- Reformat code with black
- Upgrade dependencies
- Upgrade development status to beta

## 7.2 0.4.3 (2022-03-15)

- Migrate to GitHub Actions

## 7.3 0.4.2 (2020-02-04)

- Support Python 3.8
- Upgrade flake8 (test requirement) as prior version did not support Python 3.8

## 7.4 0.4.1 (2019-11-14)

- Added return type annotations for core modules
- Added some example notebooks

## 7.5 0.4.0 (2019-07-08)

- Simple command line interface (CLI) supporting curve evaluation, calibration and plotting
- Added more documentation

## 7.6 0.3.0 (2019-03-17)

- Added type annotations

## 7.7 0.2.0 (2019-02-20)

- Ordinary least squares based calibration of Nelson-Siegel-Svensson
- Ordinary least squares based calibration of Nelson-Siegel
- A little bit of usage documentation

## 7.8 0.1.0 (2019-02-13)

- First release on PyPI.

CHAPTER 8

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## n

# Index