

---

# **LifeFit Documentation**

***Release 1.0.4***

**Fabio Steffen**

**Feb 20, 2022**



---

## Contents:

---

<b>1</b>	<b>Webserver</b>	<b>3</b>
<b>2</b>	<b>Installation</b>	<b>7</b>
2.1	Conda . . . . .	7
2.2	PyPI . . . . .	7
2.3	Install from source . . . . .	7
<b>3</b>	<b>Tutorial</b>	<b>9</b>
<b>4</b>	<b>Bug reports</b>	<b>11</b>
4.1	Lifefit Tutorial . . . . .	11
4.2	Module description . . . . .	13
<b>5</b>	<b>Reference</b>	<b>21</b>
	<b>Python Module Index</b>	<b>23</b>
	<b>Index</b>	<b>25</b>



*LifeFit* is a Python package to analyze **time-correlated single-photon counting (TCSPC)** data sets, namely **fluorescence lifetime** and **time-resolve anisotropy** decays.




# CHAPTER 1


---

## Webserver

---

You can run LifeFit directly in your browser: <https://tcspc-lifefit.herokuapp.com/>





Type of dataset

Fluorescence Lifetime▼

Mode

LifeFit Demo

Analyze your own data

Fileformat

Horiba▼



---

**Note:** Initial startup of the webserver might take a few seconds, please be patient.

---



There are different options how to install LifeFit.

### 2.1 Conda

Install the package into your conda environment

```
conda install -c fdsteffen lifefit
```

### 2.2 PyPI

Alternatively, you can install the latest release of with pip

```
pip install lifefit
```

### 2.3 Install from source

Finally, you can also get the latest development version directly from Github

```
pip install git+https://github.com/fdsteffen/Lifefit.git
```



## CHAPTER 3

---

### Tutorial

---

For an introduction into the functionality of LifeFit visit the [tutorial](#). The Jupyter Notebook can be downloaded [here](#).



---

Please report any *bugs* via the [issue tracker](#) on Github.

## 4.1 Lifefit Tutorial

```
[1]: # import modules
import lifefit as lf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

# seaborn settings
sns.set_style('white')
sns.set_context("notebook")
sns.set(font='Arial')

# plot settings
def set_ticksStyle(x_size=4, y_size=4, x_dir='in', y_dir='in'):
    sns.set_style('ticks', {'xtick.major.size': x_size, 'ytick.major.size': y_size,
    ↪ 'xtick.direction': x_dir, 'ytick.direction': y_dir})
```

### 4.1.1 Lifetime

First, define the path to the data

```
[23]: atto550_dna_path = lf._DATA_DIR+'/lifetime/Atto550_DNA.txt'
irf_path = lf._DATA_DIR+'/IRF/irf.txt'
```

Next, we read in our datafile for the fluorescence decay and the instrument response function (IRF). Instead of using the `lf.read_decay()` function we can define a custom import function that outputs a two-column array containing numbered channels and intensity counts.

```
[24]: atto550_dna, timestep_ns = lf.tcspc.read_decay(atto550_dna_path)
      irf, _ = lf.tcspc.read_decay(irf_path)
```

Next we instantiate a `Lifetime` object by providing the data arrays of the fluorescence decay and the IRF along with the timestep between two channels

```
[25]: atto550_dna_life = lf.tcspc.Lifetime(atto550_dna, timestep_ns, irf)
```

Fit the fluorescence decay by iterative reconvolution with the IRF

```
[26]: atto550_dna_life.reconvolution_fit([1,5])
```

```
=====
Reconvolution fit with experimental IRF
tau0: 1.01 ± 0.01 ns (29%)
tau1: 3.89 ± 0.01 ns (71%)
mean tau: 3.61 ± 0.01 ns

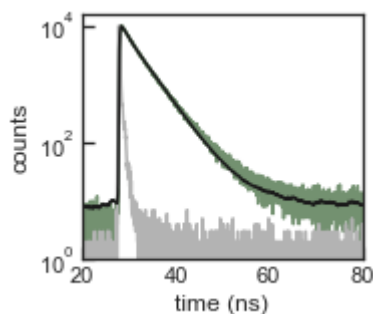
irf shift: 0.11 ns
offset: 1 counts
=====
```

Plot the IRF, the fluorescence decay and the fit

```
[27]: with sns.axes_style('ticks'):
      set_ticksStyle()
      f, ax = plt.subplots(nrows=1, ncols=1, figsize=(2.5,2.25), sharex=False,
      ↪sharey=True, squeeze=False)

      ax[0,0].semilogy(atto550_dna_life.fluor[:,0], atto550_dna_life.fluor[:,2],
      ↪color=[0.45, 0.57, 0.44])
      ax[0,0].semilogy(atto550_dna_life.irf[:,0], atto550_dna_life.irf[:,2], color=[0.7,
      ↪0.7, 0.7])
      ax[0,0].semilogy(atto550_dna_life.fluor[:,0], atto550_dna_life.fit_y, color='k')

      ax[0,0].set_ylabel('counts')
      ax[0,0].set_xlabel('time (ns)')
      ax[0,0].set_xlim((20,80))
      ax[0,0].set_ylim(bottom=1)
```



## 4.1.2 Anisotropy

Read the four different fluorescence decays and generate a `lifetime` object from each channel



```
[29]: atto550_dna_path = {}
atto550_dna = {}
atto550_dna_life = {}
for c in ['VV', 'VH', 'HV', 'HH']:
    atto550_dna_path[c] = lf._DATA_DIR+'/anisotropy/{}.txt'.format(c)
    atto550_dna[c], fluor_nsperchan = lf.tcspc.read_decay(atto550_dna_path[c])
    atto550_dna_life[c] = lf.tcspc.Lifetime(atto550_dna[c], fluor_nsperchan, irf)
```

Compute an `anisotropy` object from the lifetime objects and fit a two-rotator model to the anisotropy decay

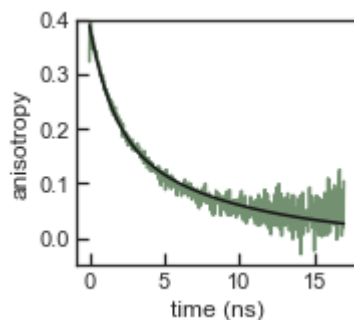
```
[30]: atto550_dna_aniso = lf.tcspc.Anisotropy(atto550_dna_life['VV'], atto550_dna_life['VH']
↪), atto550_dna_life['HV'],atto550_dna_life['HH'])
atto550_dna_aniso.rotation_fit(p0=[0.4, 1, 10,1], model='two_rotations')
```

```
=====
Anisotropy fit
model: two_rotations
r0: 0.19 ± 0.01 ns
b: 0.00 ± 0.02 ns
tau: 8.50 ± 0.45 ns
tau2: 1.57 ± 0.14 ns
=====
```

Plot the anisotropy decay with the fit

```
[31]: with sns.axes_style('ticks'):
    set_ticksStyle()
    f, ax = plt.subplots(nrows=1, ncols=1, figsize=(2.5,2.25), sharex=False,
↪sharey=True, squeeze=False)

    ax[0,0].plot(atto550_dna_aniso.time, atto550_dna_aniso.r, color=[0.45, 0.57, 0.
↪44])
    ax[0,0].plot(atto550_dna_aniso.time, atto550_dna_aniso.fit_r, color='k')
    ax[0,0].set_ylim((-0.05,0.4))
    ax[0,0].set_xlabel('time (ns)')
    ax[0,0].set_ylabel('anisotropy')
```



## 4.2 Module description

Fit lifetime decays

**class** lifefit.tcspc.**Anisotropy** (*VV, VH, HV, HH*)

Bases: object

Create an Anisotropy object with four polarization resolved lifetime decays

**Parameters**

- **VV** (*ndarray*) – vertical excitation - vertical emission
- **VH** (*ndarray*) – vertical excitation - horizontal emission
- **HV** (*ndarray*) – horizontal excitation - vertical emission
- **HH** (*ndarray*) – horizontal excitation - horizontal emission

**Example**

```
>>> lf.tccspc.Anisotropy(decay['VV'], decay['VH'], decay['HV'], decay['HH'])
```

**static G\_factor** (*HV, HH*)

Compute G-factor to correct for differences in transmission efficiency of the horizontal and vertical polarized light

**Parameters**

- **HV** (*ndarray*) – horizontal excitation - vertical emission
- **HH** (*ndarray*) – horizontal excitation - horizontal emission

**Returns** **G** (*float*) – G-factor

**Notes**

The G-factor is defined as follows:

$$G = \frac{\int HV}{\int HH}$$

**static aniso\_decay** (*VV, VH, G*)**Parameters**

- **VV** (*ndarray*) – vertical excitation - vertical emission
- **VH** (*ndarray*) – vertical excitation - horizontal emission
- **G** (*float*) – G-factor

**Returns** **r** (*ndarray*) – anisotropy decay

**Notes**

The anisotropy decay is calculated from the parallel and perpendicular lifetime decays as follows:

$$r(t) = \frac{I_{VV} - GI_{VH}}{I_{VV} + 2GI_{VH}}$$

**export** (*filename*)

Export the data and the fit parameters to a json file

**Parameters** **filename** (*str*) –

**static hindered\_rotation** (*time, r0, tau\_r, r\_inf*)

Hindered rotation in-a-cone model

**Parameters**

- **time** (*array\_like*) – time bins
- **r0** (*float*) – fundamental anisotropy
- **tau\_r** (*float*) – rotational correlation time
- **r\_inf** (*float*) – residual anisotropy at time->inf

**Returns** *ndarray* – hindered rotation anisotropy decay

**static local\_global\_rotation** (*time, r0, tau\_rloc, r\_inf, tau\_rglob*)

Local-global rotation in-a-cone model

**Parameters**

- **time** (*array\_like*) – time bins
- **r0** (*float*) – fundamental anisotropy
- **tau\_rloc** (*float*) – local rotational correlation time
- **r\_inf** (*float*) – residual anisotropy at time->inf
- **tau\_rglob** (*float*) – global rotational correlation time

**Returns** *ndarray* – local-global rotation anisotropy decay

**static one\_rotation** (*time, r0, tau*)

Single rotator model

**Parameters**

- **time** (*array\_like*) – time bins
- **r0** (*float*) – fundamental anisotropy
- **tau\_r** (*float*) – rotational correlation time

**Returns** *ndarray* – two-rotation anisotropy decay

**rotation\_fit** (*p0=[0.4, 1], model='one\_rotation', manual\_interval=None, bounds=(0, inf), verbose=True, ns\_before\_VVmax=1, signal\_percentage=0.01*)

Fit rotation model to anisotropy decay.

**Parameters**

- **p0** (*array\_like*) – start values of the chosen anisotropy fit model
- **model** (*str*) – one of the following anisotropy models: {'one\_rotation', 'two\_rotations', 'hindered\_rotation', 'local\_global\_rotation'}
- **manual\_interval** (*2-tuple of float, optional*) –
- **bounds** (*2-tuple of float or array\_like*) – lower and upper bounds for each parameter in p0. Can be either a tuple of two scalars (same bound for all parameters) or a tuple of array\_like with the same length as p0. To deactivate parameter bounds set: *bounds=(-np.inf, np.inf)*
- **verbose** (*bool*) – print anisotropy fit result
- **ns\_before\_VVmax** (*float, optional*) – how many nanoseconds before the maximum of the VV decay should the search for r0 start
- **signal\_percentage** (*float, optional*) – percentage of the VV decay serving as a threshold to define the end of the anisotropy fit interval

## Example

```
>>> obj.rotation_fit(p0=[0.4, 1, 10, 1], model='two_rotations')
```

**static two\_rotations** (*time*, *r0*, *b*, *tau\_r1*, *tau\_r2*)  
Two-rotator model

### Parameters

- **time** (*array\_like*) – time bins
- **r0** (*float*) – fundamental anisotropy
- **b** (*float*) – amplitude of second decay
- **tau\_r1** (*float*) – first rotational correlation time
- **tau\_r2** (*float*) – second rotational correlation time

**Returns** *ndarray* – two-rotation anisotropy decay

**class** lifefit.tcspc.Lifetime (*fluor\_decay*, *fluor\_ns\_per\_chan*, *irf\_decay=None*,  
*gauss\_sigma=None*, *gauss\_amp=None*, *shift\_time=False*)

Bases: object

Create lifetime class

### Parameters

- **fluor\_decay** (*ndarray*) – *n* x 2 array containing numbered channels and intensity counts of the fluorescence decay
- **fluor\_ns\_per\_chan** (*float*) – nanoseconds per channel
- **irf\_decay** (*ndarray*, *optional*) – *n* x 2 array containing numbered channels and intensity counts for instrument response function (IRF). If *None*, then IRF is approximated by a Gaussian

### Variables

- **ns\_per\_chan** (*float*) – nanoseconds per channel
- **fluor** (*ndarray*) – *n* x 4 array containing time, channel number, intensity counts and associated Poissonian weights of the fluorescence decay
- **irf** (*ndarray*) – *n* x 3 array containing time, channel number and intensity counts of the IRF
- **irf\_type** (*str*) – type of IRF: {'Gaussian', 'experimental'}
- **fit\_param** (*ndarray*) –
- **fit\_param\_std** (*ndarray*) –

## Example

```
>>> fluor, fluor_nsperchan = lf.tcspc.read_decay(pathToFluorDecay)
>>> irf, irf_nsperchan = lf.tcspc.read_decay(pathToIRF)
>>> lf.tcspc.Lifetime(fluor, fluor_nsperchan, irf)
```

**static average\_lifetime** (*a*, *tau\_val*, *tau\_std*)  
Calculate average lifetime according to<sup>1</sup>

<sup>1</sup>

**Parameters**

- **a** (*array\_like*) – weighting factors of tau
- **tau\_val** (*array\_like*) – fluorescence lifetimes
- **tau\_std** (*array\_like*) – standard deviation of the fluorescence lifetimes

**Returns** **av\_lt** (*tuple*) – average lifetime and associated standard deviation

**References**

**static convolution** (*irf, sgl\_exp*)

Compute convolution of irf with a single exponential decay

**Parameters**

- **irf** (*array\_like*) – intensity counts of the instrument reponse function (experimental of Gaussian shaped)
- **sgl\_exp** (*array\_like*) – single-exponential decay

**Returns** **convolved** (*ndarray*) – convoluted signal of IRF and exponential decay

**static exp\_decay** (*time, tau*)

Create a single-exponential decay

**Parameters**

- **time** (*array\_like*) – time bins
- **tau** (*float*) – fluorescence lifetime

**Returns** **sgl\_exp** (*array\_like*) – single-exponential decay

**export** (*filename*)

**classmethod from\_filenames** (*fluor\_file, irf\_file=None, fileformat='HORIBA', gauss\_sigma=None, gauss\_amp=None, shift\_time=False*)

Alternative constructor for the Lifetime class by reading in filename for the fluorophore and IRF decay

**Parameters**

- **fluor\_file** (*str*) – filename of the fluorophore decay
- **irf\_file** (*str*) – filename of the IRF decay
- **fileformat** (*str, optional*) – currently implemented formats: {'HORIBA'}

**Example**

```
>>> lf.tcspc.Lifetime.from_filenames(pathToFluorDecay, pathToIRFDecay)
```

**static gauss\_irf** (*time, mu, sigma=0.01, A=10000*)

Calculate a Gaussian-shaped instrument response function (IRF)

**Parameters**

- **time** (*ndarray*) – time bins
- **mu** (*float*) – mean of the Gaussian distribution

---

J. Lakowicz, *Principles of Fluorescence*, 3rd ed., Springer, 2010.

- **sigma** (*float, optional*) – standard deviation of the Gaussian distribution
- **A** (*float, optional*) – amplitude of the Gaussian distribution

**Returns** **irf** (*ndarray*) – Gaussian shaped instrument response function (IRF)

**nls\_convolve\_irfexp** (*x\_data, p0*)

Solve non-negative least squares for series of IRF-convolved single-exponential decays. First, the IRF is shifted, then convolved with each exponential decay individually (decays 1,...,n), merged into an  $m \times n$  array (=A) and finally plugged into `scipy.optimize.nls(A, experimental y-data)` to compute  $\text{argmin}_x \|Ax - y\|_2$ . This optimizes the relative weight of the exponential decays whereas the `curve_fit` function optimizes the decay parameters (`tau1`, `taus2`, etc.)

#### Parameters

- **x\_data** (*array\_like*) – array of the independent variable
- **p0** (*array\_like*) – start values for the fit model

#### Returns

- **A** (*ndarray*) – matrix containing irf-convoluted single-exponential decays in the first n columns and ones in the last column (background counts)
- **x** (*ndarray*) – vector that minimizes  $\|Ax - y\|_2$
- **y** (*ndarray*) – fit vector computed as  $y = Ax$

**reconvolution\_fit** (*tau0=[1], tau\_bounds=(0, inf), irf\_shift=0, sigma=None, verbose=True*)

Fit the experimental lifetime decay to a series of exponentials via iterative reconvolution with the instrument response function (IRF).

#### Parameters

- **tau0** (*int or array\_like*) – start value(s) of the fluorescence lifetime(s)
- **tau\_bounds** (*2-tuple of float or 2-tuple of array\_like, optional*) – lower and upper bounds for each parameter in `tau0`. Can be either a tuple of two scalars (same bound for all parameters) or a tuple of `array_like` with the same length as `tau0`. To deactivate parameter bounds set: `bounds=(-np.inf, np.inf)`
- **irf\_shift** (*int, optional*) – shift of the IRF on the time axis (in channel units)
- **sigma** (*array\_like, optional*) – uncertainty of the decay (same length as `y_data`)
- **verbose** (*bool, optional*) – print lifetime fit result

### Example

```
>>> obj.reconvolution_fit([1, 5])
```

`lifefit.tcspec.fit` (*fun, x\_data, y\_data, p0, bounds=[0, 0, 0], [inf, inf, inf], sigma=None*)

Wrapper for the `curve_fit` function of the `scipy.optimize` module. The `curve_fit` optimizes the decay parameters (`tau1`, `tau2`, etc.) while the `nls` weights the exponential decays.

#### Parameters

- **fun** (*callable*) – The model function  $f(x, \dots)$  taking `x` values as a first argument followed by the function parameters
- **x\_data** (*array\_like*) – array of the independent variable

- **y\_data** (*array\_like*) – array of the dependent variable
- **p0** (*array\_like*) – start values for the fit model
- **bounds** (*2-tuple of float or 2-tuple of array\_like, optional*) – lower and upper bounds for each parameter in p0. Can be either a tuple of two scalars (same bound for all parameters) or a tuple of *array\_like* with the same length as p0. To deactivate parameter bounds set: *bounds=(-np.inf, np.inf)*
- **sigma** (*array\_like, optional*) – uncertainty of the decay (same length as y\_data)

**Returns**

- **p** (*ndarray*) – optimized fit parameters
- **p\_std** (*ndarray*) – standard deviation of optimized fit parameters

`lifefit.tcspc.parseCmd()`

Parse the command line to get the experimental decay and instrument reponse function (IRF) file.

**Returns**

- **fluor\_file** (*str*) – filename of the fluorescence decay
- **irf\_file** (*str*) – filename of the IRF (if None then the IRF is approximated by a Gaussian)

`lifefit.tcspc.parse_file(decay_file, fileformat='Horiba')`

Parse the decay file

**Parameters**

- **decay\_file** (*StringIO*) –
- **fileformat** (*str, optional*) – currently implemented formats: {'HORIBA'}

**Returns**

- **decay\_data** (*ndarray*) –  $n \times 2$  decay containing numbered channels and intensity counts for instrument reponse function (IRF)
- **ns\_per\_chan** (*float*)

`lifefit.tcspc.read_decay(filepath_or_buffer, fileformat='Horiba')`

Read TCSPC decay file from HORIBA or another data format

**Parameters**

- **filepath\_or\_buffer** (*str, os.PathLike, StringIO*) – filename of the decay or StringIO object
- **fileformat** (*str, optional*) – currently implemented formats: {'HORIBA'}

**Returns**

- **decay\_data** (*ndarray*) –  $n \times 2$  decay containing numbered channels and intensity counts for instrument response function (IRF)
- **ns\_per\_chan** (*float*)





## CHAPTER 5

---

### Reference

---

To cite LifeFit, please refer to the following paper:

F.D. Steffen, R.K.O. Sigel, R. Börner, *Phys. Chem. Chem. Phys.* **2016**, *18*, 29045-29055.



|

`lifefit.tcspc`, 13



## A

`aniso_decay()` (*lifefit.tcspc.Anisotropy static method*), 14

*Anisotropy* (class in *lifefit.tcspc*), 13

`average_lifetime()` (*lifefit.tcspc.Lifetime static method*), 16

## C

`convolution()` (*lifefit.tcspc.Lifetime static method*), 17

## E

`exp_decay()` (*lifefit.tcspc.Lifetime static method*), 17

`export()` (*lifefit.tcspc.Anisotropy method*), 14

`export()` (*lifefit.tcspc.Lifetime method*), 17

## F

`fit()` (*in module lifefit.tcspc*), 18

`from_filenames()` (*lifefit.tcspc.Lifetime class method*), 17

## G

`G_factor()` (*lifefit.tcspc.Anisotropy static method*), 14

`gauss_irf()` (*lifefit.tcspc.Lifetime static method*), 17

## H

`hindered_rotation()` (*lifefit.tcspc.Anisotropy static method*), 14

## L

*lifefit.tcspc* (module), 13

*Lifetime* (class in *lifefit.tcspc*), 16

`local_global_rotation()` (*lifefit.tcspc.Anisotropy static method*), 15

## N

`nls_convolve_irfexp()` (*lifefit.tcspc.Lifetime method*), 18

## O

`one_rotation()` (*lifefit.tcspc.Anisotropy static method*), 15

## P

`parse_file()` (*in module lifefit.tcspc*), 19

`parseCmd()` (*in module lifefit.tcspc*), 19

## R

`read_decay()` (*in module lifefit.tcspc*), 19

`reconvolution_fit()` (*lifefit.tcspc.Lifetime method*), 18

`rotation_fit()` (*lifefit.tcspc.Anisotropy method*), 15

## T

`two_rotations()` (*lifefit.tcspc.Anisotropy static method*), 16