PyFDAP: automated analysis of Fluorescence Decay After Photoconversion (FDAP) experiments

User Guide

Alexander Bläßle and Patrick Müller

Friedrich Miescher Laboratory of the Max Planck Society
Spemannstraße 39
72076 Tübingen
Germany

E-Mail: alexander.blaessle@tuebingen.mpg.de, patrick.mueller@tuebingen.mpg.de
Website: http://people.tuebingen.mpg.de/mueller-lab/

CONTENTS

1 INTRODUCTION

Fluorescence Decay After Photoconversion (FDAP) is a microscopy-based technique for measuring protein half-lives (Rogers *et al.*, 2014). In FDAP experiments, a protein of interest is tagged with a photoconvertible fluorescent protein and expressed *in vivo*. The fluorescent fusion protein is then photoconverted, and the decrease in fluorescence intensity over time is monitored. The resulting intensity data is fitted with a decay function, and half-lives can be calculated from the fits.

Both intracellular and extracellular protein half-lives can be determined using FDAP. A static intracellular signal (e.g. Alexa488-dextran) can be used to create an intracellular mask, such that only intracellular pixels are considered when calculating intracellular intensity. The mask can be inverted to calculate extracellular intensities.

Here, we provide a standardized computational framework to analyze FDAP datasets. Our software PyFDAP features (i) a comprehensive data format for handling, sorting, and annotating large FDAP datasets, (ii) the capability to separate fluorescence intensities in FDAP datasets into intra- and extracellular compartments based on counter-labeling, (iii) established fitting algorithms, and (iv) a user-friendly environment that allows researchers from a non-computational background to easily evaluate FDAP datasets.

2 INSTALLATION

PyFDAP was developed as an open source graphical user interface (GUI) in Python with PyQT and SciPy in order to make it accessible and extendable across the most frequently used operating systems Ubuntu Linux, Mac OS X, and Microsoft Windows. Over the past two decades, Python has become a widely used scientific programming language and provides PyFDAP users with enormous resources and easily addable software packages (Millman and Aivazis, 2011).

All software packages needed to run PyFDAP are freely available. PyFDAP can be installed using stand-alone executables (see Section 2.1). Alternatively, users can run the PyFDAP packages from source (see Section 2.2), which offers the possibility to edit the PyFDAP code and to import new modules.

2.1 Running PyFDAP using stand-alone executables

Download the executable that fits your system from http://people.tuebingen.mpg.de/mueller-lab/. This is suitable for users who want to analyze FDAP experiments and do not need to customize the PyFDAP code. A list of currently available binary files and systems on which the binaries have been tested can be found in Table 1. If there is no executable available for your system, we recommend using the Anaconda installation approach explained in Section 2.2.1.

OS	Version	32-bit	64-bit	Executable	Test System
Linux	3.13.0-36-generic		×	pyfdap_v1.0_Linux_64bit	Thinkpad x230
Mac OS X	10.9.2		×	pyfdap_v1.0_OSX_64bit.app	MacMini6,1
Mac OS X	10.9.2		×	pyfdap_v1.0_OSX_64bit.app	MacBookPro10,2
Mac OS X	10.9.5		×	pyfdap_v1.0_OSX_64bit.app	MacBookPro8,1
Mac OS X	10.9.5		×	pyfdap_v1.0_OSX_64bit.app	MacBookPro8,2
Mac OS X	10.9.5		×	pyfdap_v1.0_OSX_64bit.app	MacBookPro10,2
Windows	7	×		pyfdap_v1.0_Win_32bit.exe	Samsung N150
Windows	8		×	pyfdap_v1.0_Win_64bit.exe	Dell OPTIPLEX 9010

Table 1: List of systems on which the currently available PyFDAP executables have been tested. The executables might also run on systems not listed here.

2.2 Running PyFDAP from source

In order to be able to edit the PyFDAP code and to import new modules, it is necessary to download and install all necessary Python packages and to run PyFDAP from source. There are two ways to do this:

1. Download and install the Anaconda Python distribution (see Section 2.2.1).
2. Download and install all Python packages manually (see Section 2.2.2).

2.2.1 Running PyFDAP using the Anaconda distribution

Anaconda is a bundle of Python packages and includes all packages needed to run PyFDAP. To install Anaconda, follow these steps:

- Go to http://continuum.io/downloads and download the current Python 2.7.x release of Anaconda for your operating system
- Launch the installer by double-clicking (Mac OS X and Windows) or
  - Open a Terminal
  - Go to the directory containing the installer by typing

    ```
    cd path/to/installer
    ```

    and execute the installer with

    ```
    ./installer
    ```

- Follow the instructions of the installer
- Launch PyFDAP by double-clicking `pyfdap_app.py` in the PyFDAP source directory (Windows) or
  - Open a Terminal
  - Go to the directory containing the PyFDAP source files

    ```
    cd path/to/PyFDAP
    ```

  - Launch PyFDAP by typing

    ```
    python pyfdap_app.py
    ```

2.2.2 Running PyFDAP using a manual Python installation

In this section, we explain how to manually install all necessary Python packages on Linux, Mac OS X, and Windows in order to run PyFDAP. The manual installation allows for customizability as well as debugging options. The instructions provided here describe the installation process for computers that currently do not have Python installed. For computers on which Python is already installed, the installation of PyFDAP will differ from the instructions provided below. We recommend running PyFDAP using a Debian-based Linux distribution such as Ubuntu since installing Python packages is more straightforward using such operating systems.

Manual installation under Linux

Here we explain how to manually install and run PyFDAP on Linux operating systems. The following instructions are only suitable for Debian-based Linux distributions and have been tested on Ubuntu Linux 12.04, 13.10, and 14.04 (64-bit). Installation steps may vary between different versions and distributions of Linux (e.g. RedHat-based Linux distributions such as Fedora or Suse).

- Open a Terminal
- In your Terminal, type (you will need sudo rights)

  ```
  sudo apt-get install python-numpy
  sudo apt-get install python-scipy
  sudo apt-get install python-matplotlib
  sudo apt-get install python-qt4
  sudo apt-get install python-skimage
  ```

  Note: On Ubuntu versions older than 12.10, python-skimage needs to be installed from `http://neuro.debian.net/pkgs/python-skimage.html`.

- Download and unpack the current version of PyFDAP from `http://people.tuebingen.mpg.de/mueller-lab`
- Go to your PyFDAP folder by typing

  ```
  cd path/to/PyFDAP/
  ```

  and launch PyFDAP by typing

  ```
  python pyfdp_app.py
  ```

  If PyFDAP does not launch, open a Python Terminal and try to import all necessary packages by typing

  ```
  import numpy
  import scipy
  import matplotlib
  import matplotlib.image
  import PyQt4
  import code
  ```

  If you receive an error message while importing any of these modules, try to re-install the packages or visit the development website of the problematic package.

Manual installation under Mac OS X

Here we explain how to manually install and run PyFDAP on Mac OS X. The following instructions have only been tested on Mac OS X Snow Leopard 10.6.8 (64-bit) and Mac OS X Maverick 10.9.2, 10.9.4, 10.9.5 (64-bit). Installation steps may vary between different versions of OS X.

- Installing Python packages requires the C++ compiler gcc. gcc can be obtained by downloading XCode from the Apple AppStore.

- Launch a Terminal in Applications → Utilities → Terminal
- Type

  ```
  gcc
  ```

  You should see a pop-up window asking you to install Command Line Tools. Follow the instructions in the pop-up window.
- Homebrew is a package manager for Mac OS X that facilitates installing packages under OS X. Download Homebrew by typing

  ```
  ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/
      go/install)"
  ```

- Check the Homebrew installation by typing

  ```
  brew update
  brew doctor
  ```

  If the ouput returns any problems, visit the Homebrew website (http://brew.sh/) for further instructions.
- Install Python by typing into the Terminal

  ```
  brew install python
  ```

  Note that Mac OS X comes with a native Python installation. If you want to use the native Python installation, you can install all packages separately by using the Python Package Index (pip), or you can use Homebrew to install all packages and then link them using the `site` package from https://docs.python.org/2/library/site.html. However, we recommend using the Python installation of Homebrew.
- Link the new Homebrew installation by typing into the Terminal

  ```
  brew link python
  brew linkapps
  ```

- Link the new Python installation into .bash_profile by launching the text editor `nano`

  ```
  nano ~/.bash_profile
  ```

  and add the following lines

  ```
  PATH="/usr/local/bin:${PATH}"
  export PATH
  export PYTHONPATH=/usr/local/lib/python2.7/site-packages/:
  ```

  Press `Ctrl+O` and `Ctrl+X` to save the new .bash_profile and exit. Restart the Terminal and type

  ```
  which python
  ```

  The output should be

```
/usr/local/bin/python
```

If not, ensure that you have set the Python path properly and use the appropriate Homebrew installation prefix. If everything went correctly, you will now use the Homebrew Python installation when you call `python` in the Terminal.

- Download and install PyQT4 and SIP by typing into the Terminal

```
brew install sip
brew install pyqt
brew linkapps
```

- Download and install Nose and NumPy by typing into the Terminal

```
pip install nose
brew install numpy
brew link numpy
```

Sometimes NumPy can also be found by typing into the Terminal

```
brew install homebrew/python/numpy
brew link numpy
```

- Download and install SciPy by typing into the Terminal

```
pip install scipy
```

or

```
brew install scipy
```

- Download and install scikit-image by typing into the Terminal

```
pip install cython
pip install scikit-image
```

- Download and install Matplotlib by typing into the Terminal

```
pip install python-dateutil
pip install pyparsing
brew install matplotlib
```

- Download and install PIL by typing into the Terminal

```
brew install Homebrew/python/pillow
```

- Download and unpack the current version of PyFDAP from `http://people.tuebingen.mpg.de/mueller-lab/`
- Go to your PyFDAP folder by typing into the Terminal

```
cd path/to/PyFDAP/
```

and launch PyFDAP by typing into the Terminal

```
python pyfdp_app.py
```

If PyFDAP does not launch, open a Python Terminal and try to import all necessary packages by typing into the Terminal

```
import numpy
import scipy
import matplotlib
import matplotlib.image
import PyQt4
import code
```

If you receive an error message while importing any of these modules, try to re-install the packages or visit the development website of the problematic package.

Manual installation under Microsoft Windows

Here we explain how to manually install and start PyFDAP on Microsoft Windows. The following instructions have only been tested for Microsoft Windows 8 (64-bit) and may differ for other versions.

- Download and install the current version of Python 2.7x from `https://www.python.org/download`.
- Download and install the current version of PyQt4 from http://www.riverbankcomputing.co.uk/. The Windows installer will also install the required package SIP and all necessary QT libraries.
- Download and install the current version of SciPy Stack from http://www.lfd.uci.edu/~gohlke/pythonlibs. SciPy Stack includes important Python packages such as Nose, NumPy, SciPy, and Matplotlib. We recommend using SciPy Stack, but if you need to install the packages separately because there is no suitable installation binary of SciPy Stack available, you can use the following links:
  - NumPy: `http://sourceforge.net/projects/numpy/files/NumPy/` if you are running a 32-bit system, on a 64-bit system go to `http://www.kfd.uci.edu/~gohlke/pythonlibs/`
  - SciPy: `http://sourceforge.net/projects/scipy/files/scipy/`
  - Matplotlib: `http://matplotlib.org/downloads.html`
  - Nose: `https://nose.readthedocs.org/en/latest/`
  - IPython: `https://github.com/ipython/ipython/releases`
- Download and install the current version of scikit-image from http://www.lfd.uci.edu/~gohlke/pythonlibs/.
- Download and unpack the current version of PyFDAP from `http://people.tuebingen.mpg.de/mueller-lab`.
- Go to your PyFDAP folder and launch `pyfdp_app.py`. If PyFDAP does not launch, open a Python Terminal and try to import all necessary packages by typing

```
import numpy
import scipy
import matplotlib
import matplotlib.image
import PyQt4
import code
```

If you receive an error message while importing any of these modules, try to re-install the packages or visit the development website of the problematic package.

2.3 Enabling video output for PyFDAP

PyFDAP can convert image series into video files for presentation purposes (see also Section 3.4.1). This requires the installation of MEncoder:

Under Linux, open a Terminal and type

```
sudo apt-get install mencoder
```

If you have followed the manual installation instructions for OS X (see Section 2.2.2), open a Terminal and type

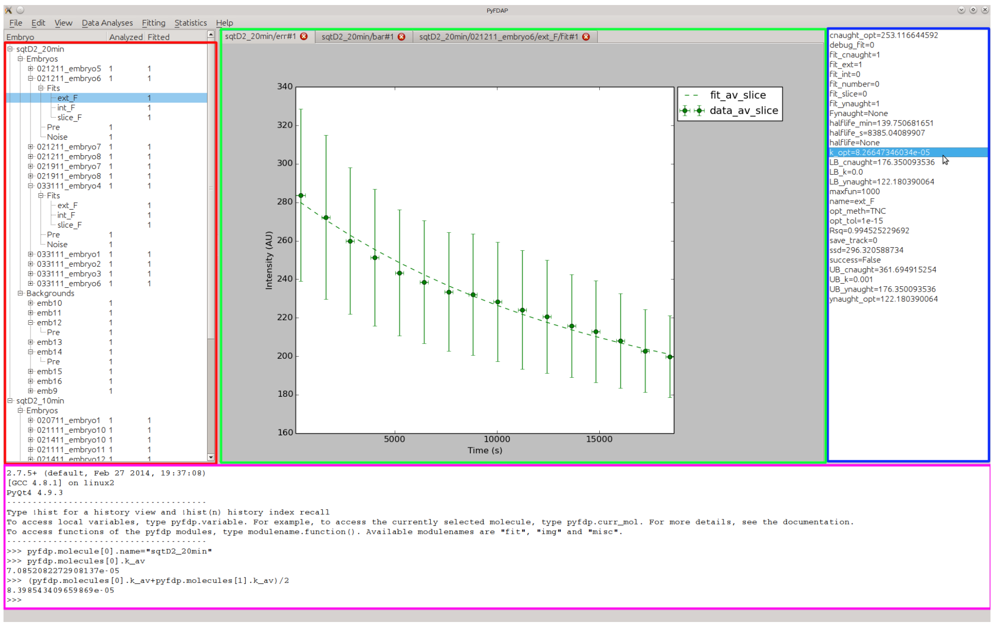
```
brew install mplayer
```

More information about data output in PyFDAP can be found in Section 3.4.

3 WORKING WITH PYFDAP

3.1 The PyFDAP main window

The PyFDAP main window consists of four major compartments: The object list on the left-hand side (red), the property list on the right-hand side (blue), the plot tab in the center (green), and the console at the bottom (magenta).



```
Embryo                 Analyzed Fitted
⊟ sqtD2_20min
  ⊟ Embryos
    ⊞ 021211_embryo5   1        1
    ⊟ 021211_embryo6   1        1
      ⊟ Fits
          ext_F                 1
          int_F                 1
          slice_F               1
        Pre            1
        Noise          1
    ⊞ 021211_embryo7   1        1
    ⊞ 021211_embryo8   1        1
    ⊞ 021911_embryo7   1        1
    ⊞ 021911_embryo8   1        1
    ⊟ 033111_embryo4   1        1
      ⊟ Fits
          ext_F                 1
          int_F                 1
          slice_F               1
        Pre            1
        Noise          1
    ⊞ 033111_embryo1   1        1
    ⊞ 033111_embryo2   1        1
    ⊞ 033111_embryo3   1        1
    ⊞ 033111_embryo6   1        1
  ⊟ Backgrounds
    ⊞ emb10            1
    ⊞ emb11            1
    ⊟ emb12            1
        Pre            1
    ⊞ emb13            1
    ⊟ emb14            1
        Pre            1
    ⊞ emb15            1
    ⊞ emb16            1
    ⊞ emb9             1
⊟ sqtD2_10min
  ⊟ Embryos
    ⊞ 020711_embryo1   1        1
    ⊞ 021111_embryo10  1        1
    ⊞ 021411_embryo10  1        1
    ⊞ 021111_embryo11  1        1
    ⊞ 021411_embryo12  1        1
```

```
cnaught_opt=253.116644592
debug_fit=0
fit_cnaught=1
fit_ext=1
fit_int=0
fit_number=0
fit_slice=0
fit_ynaught=1
Fynaught=None
halflife_min=139.750681651
halflife_s=8385.04089907
halflife=None
k_opt=8.26647346034e-05
LB_cnaught=176.350093536
LB_k=0.0
LB_ynaught=122.180390064
maxfun=1000
name=ext_F
opt_meth=TNC
opt_tol=1e-15
Rsq=0.994525229692
save_track=0
ssd=296.320588734
success=False
UB_cnaught=361.694915254
UB_k=0.001
UB_ynaught=176.350093536
ynaught_opt=122.180390064
```

```
2.7.5+ (default, Feb 27 2014, 19:37:08)
[GCC 4.8.1] on linux2
PyQt4 4.9.3
---------------------------------------
Type !hist for a history view and !hist(n) history index recall
To access local variables, type pyfdp.variable. For example, to access the currently selected molecule, type pyfdp.curr_mol. For more details, see the documentation.
To access functions of the pyfdp modules, type modulename.function(). Available modulenames are "fit", "img" and "misc".
---------------------------------------
>>> pyfdp.molecule[0].name="sqtD2_20min"
>>> pyfdp.molecules[0].k_av
7.08520822729081 37e-05
>>> (pyfdp.molecules[0].k_av+pyfdp.molecules[1].k_av)/2
8.39854340965986 9e-05
>>>
```

After creating a new molecule, FDAP, background dataset, or fit, the newly created object is shown in the object list according to its hierarchical structure (see Section 4). To inspect the object properties, double-click on the object of choice. The object properties are then listed in the property list on the right-hand side. Many functions in PyFDAP will require you to select the right type of object and will return an error message if not done so.

PyFDAP provides the user with several plotting options. Each plot opens in a new tab with a name according to the currently selected object and the plot type. You can easily switch between plots by clicking on the open tabs.

PyFDAP also comes with an internal Python console. NumPy and the three main PyFDAP modules *img, fit, misc* are automatically imported. You can use the console to manipulate all PyFDAP objects such as molecules and embryos (FDAP datasets), call other Python functions or simply let PyFDAP return molecule or embryo properties such as longer vectors that are not shown in the property list. PyFDAP also uses the console for debugging outputs, so having a look at the console is often useful.

All major PyFDAP functions can be found in the menu bar at the top of the PyFDAP window. The menus are sorted according to the normal workflow of FDAP experiment analysis.

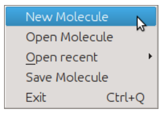
3.2 First steps with PyFDAP

We provide a fully analyzed FDAP dataset on our website. If you wish to try out PyFDAP using this test dataset, go to `http://people.tuebingen.mpg.de/mueller-lab`, download the test dataset *TestDataset.zip*, and unzip it to your PyFDAP folder. If you wish to put it somewhere else, you need to adjust some paths in the molecule file in PyFDAP later. You can now analyze the raw images of the test dataset or your own data (see Section 3.2.1), or you can load a pre-analyzed dataset (see Section 3.2.2).

3.2.1 Analyzing an FDAP dataset

The following section guides you through the major steps of how to use PyFDAP to analyze and fit FDAP datasets if you wish to perform your own FDAP analysis.

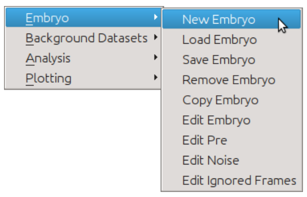
1. Create a new molecule project by clicking on *File* → *New Molecule*.



2. Change the name of the molecule project by clicking on *Edit* → *Edit Molecule*.

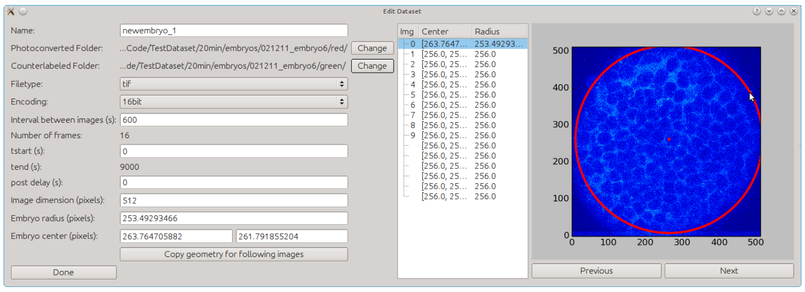
3. Add a new embryo object (FDAP measurement):

   (a) Go to *Data Analyses* → *Embryo* → *New Embryo*



   (b) Choose the photoconverted folder (images of photoconverted proteins) and counter-labeled folder (images of cell-tracing molecules, e.g. Alexa488-Dextran). For the test dataset, these can be found in the folder *TestDataset/squint-dendra2_20min-interval/embryo6/post*; the photoconverted folder is called *red*, and the counter-labeled folder is called *green*.

(c) Enter the dataset-specific properties such as intervals between images (20 min = 1200 s for the test dataset), post-delay (delay between first and second post-conversion pictures resulting from re-adjustment), and center and radius for each image. You can easily select the center and the radius for each image by clicking on the picture. The first click will define the center, the second the radius, and the third click will delete both selections. If you wish to copy the selected radius and center for all following images, click on *Copy geometry for following images*. When you are done defining the dataset, click on *Done*.



Name:	newembryo_1
Photoconverted Folder:	...Code/TestDataset/20min/embryos/021211_embryo6/red/
Counterlabeled Folder:	...de/TestDataset/20min/embryos/021211_embryo6/green/
Filetype:	tif
Encoding:	16bit
Interval between images (s):	600
Number of frames:	16
tstart (s):	0
tend (s):	9000
post delay (s):	0
Image dimension (pixels):	512
Embryo radius (pixels):	253.49293466
Embryo center (pixels):	263.764705882 261.791855204

Img	Center	Radius
0	[263.7647...	253.49293...
1	[256.0, 25...	256.0
2	[256.0, 25...	256.0
3	[256.0, 25...	256.0
4	[256.0, 25...	256.0
5	[256.0, 25...	256.0
6	[256.0, 25...	256.0
7	[256.0, 25...	256.0
8	[256.0, 25...	256.0
9	[256.0, 25...	256.0
	[256.0, 25...	256.0
	[256.0, 25...	256.0
	[256.0, 25...	256.0
	[256.0, 25...	256.0
	[256.0, 25...	256.0
	[256.0, 25...	256.0

(d) The next pop-up window will allow you to set the "photoconverted" folder, counter-labeled folder, and specific properties of the pre-conversion images similar to the post-conversion dataset in steps (b) and (c). For the test dataset, these can be found in *TestDataset/squint-dendra2_20min-interval/embryo6/pre*; the "photoconverted" folder is called *red*, and the counter-labeled folder is called *green*.

(e) The third pop-up window will allow you to define the method of noise calculation. You can choose between three methods:

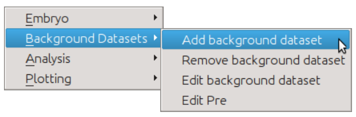
- *Outside* will average intensities outside of the selected radius for each image defined in (c) and then average over all of the calculated averages.
- *Predefined* gives you the possibility to enter a value for the noise level yourself.
- *Separate Dataset* lets you analyze a separate dataset taken to calculate noise levels. These images are generally taken before or after the experiments without a sample.

After clicking *Done*, all important settings for the embryo object are entered.

(f) You can add additional embryo objects (FDAP measurements) to the molecule by repeating steps (a) - (e).

4. Add a new background object:

(a) Go to *Data Analyses* → *Background Datasets* → *Add background dataset*.



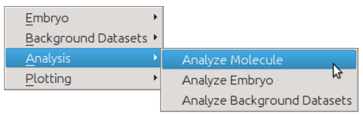
(b) Choose the "photoconverted" folder (images of "photoconverted" proteins) and counter-labeled folder (images of cell-tracing molecules). For the test dataset, these can be found in *TestDataset/squint-background_20min-interval/embryo10/post*; the "photoconverted" folder is called *red*, and the counter-labeled folder is called *green*.

(c) Similar to the embryo object, select parameters specific to the dataset by using the given text fields or by clicking on the image.

(d) The next pop-up window will allow you to set the folders and properties of the pre-conversion images of the background dataset similar to the post-conversion dataset. For the test dataset, these can be found in *TestDataset/squint-background_20min-interval/embryo10/pre*; the "photoconverted" folder is called *red*, and the counter-labeled folder is called *green*.

(e) After clicking *Done*, all important settings for the background object are entered. You can add additional background objects to the molecule by repeating steps (a) - (d).

5. Analyze the molecule project by going to *Data Analyses* → *Analysis* → *Analyze Molecule*. This can take several minutes depending on the amount of datasets added to the molecule project (see Section 5).



The image analysis progress will be printed into the PyFDAP console.

6. Double-click on the embryo object you want to analyze and add a new fit object:

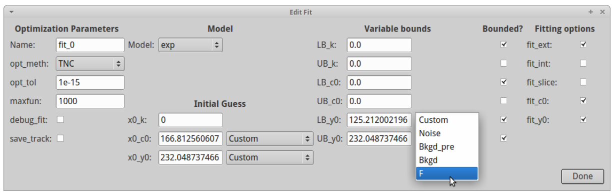
(a) Go to *Fitting* → *Fits* → *New fit*.

(b) Enter the parameters of the fit. The most important are:

- *opt_meth* is the optimization method (see Table 3 for details) used for finding the minimum of the SSD (sum of squared differences).
- *opt_tol* is the level of tolerance (i.e how good the fit needs to be) given to the optimization algorithm.
- *maxfun* is the maximum number of iterations used by the optimizer.
- *Model* is the underlying decay model used for the fit. See Section 6.1 for more information.
- *x0_k, x0_c0, x0_y0* are the initial guesses for the three parameters k, c_0, and y_0.
- *LB_k, UB_k, LB_c0, UB_c0, LB_y0, UB_y0* are the lower and upper bounds for the three parameters k, c_0, and y_0 given to the optimizer. You can use the checkboxes to set each variable bounded or unbounded from below and above. For the lower bound of y_0, PyFDAP offers several presets:
  - *Custom* allows you to enter a value yourself.
  - *Noise* takes the level of noise as the lower bound for y_0.
  - *Bkgd_pre* takes the level of the background pre-conversion images as the lower bound for y_0.
  - *Bkgd* takes the average background level as the lower bound for y_0.
  - *F* takes the weighting function given in Müller *et al.* (2012) as the lower bound for y_0.

  More details on the estimation of initial guesses and variable bounds can be found in Section 6.2. Note that not all optimization algorithms offer bounded optimization (see Section 6.3 for more details).
- *fit_ext, fit_int, fit_slice* define which regions of the images need to be fitted. You can only select one of the three regions intracelluar, extracelluar, and slice (i.e. total imaged domain) to be fitted during one particular fit.

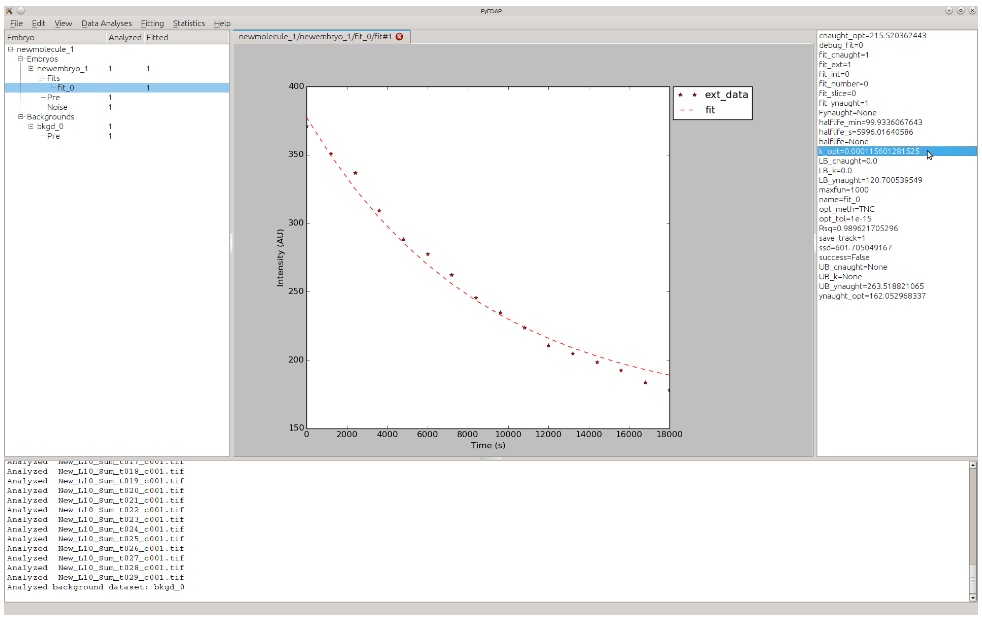
- *fit_c0*, *fit_y0* are flags on which parameters are kept fixed and which are free. If a parameter is unchecked, the optimization algorithm will keep this parameter at its initial guess value.



Optimization Parameters		Model			Variable bounds		Bounded?	Fitting options	
Name:	fit_0	Model:	exp		LB_k:	0.0	✓	fit_ext:	✓
opt_meth:	TNC				UB_k:	0.0		fit_int:	
opt_tol	1e-15				LB_c0:	0.0	✓	fit_slice:	
maxfun:	1000	Initial Guess			UB_c0:	0.0		fit_c0:	✓
debug_fit:		x0_k:	0		LB_y0:	125.212002196	✓	fit_y0:	✓
save_track:		x0_c0:	166.812560607	Custom	UB_y0:	232.048737466	✓		
		x0_y0:	232.048737466	Custom					

Custom
Noise
Bkgd_pre
Bkgd
F

- After clicking *Done*, all important settings for the fit object are entered. The fit is performed instantly, and you will see the fitted data. To inspect the optimal parameters resulting from the fit, double-click on the current fit and look in the property list on the right-hand side for *k_opt* (the decay rate constant) and *halflife_min* (the half-life in minutes).



Embryo	Analyzed	Fitted
newmolecule_1		
Embryos		
newembryo_1	1	1
Fits		
fit_0		1
Pre	1	
Noise	1	
Backgrounds		
bkgd_0	1	
Pre	1	

```
cnaught_opt=215.520362443
debug_fit=0
fit_cnaught=1
fit_ext=1
fit_int=0
fit_number=0
fit_slice=0
fit_ynaught=1
Fynaught=None
halflife_min=99.9336067643
halflife_s=5996.01640586
halflife=None
k_opt=0.000115601281525
LB_cnaught=0.0
LB_k=0.0
LB_ynaught=120.700539549
maxfun=1000
name=fit_0
opt_meth=TNC
opt_tol=1e-15
Rsq=0.989621705296
save_track=1
ssd=601.705049167
success=False
UB_cnaught=None
UB_k=None
UB_ynaught=263.518821065
ynaught_opt=162.052968337
```

```
Analyzed  New_L10_Sum_t017_c001.tif
Analyzed  New_L10_Sum_t018_c001.tif
Analyzed  New_L10_Sum_t019_c001.tif
Analyzed  New_L10_Sum_t020_c001.tif
Analyzed  New_L10_Sum_t021_c001.tif
Analyzed  New_L10_Sum_t022_c001.tif
Analyzed  New_L10_Sum_t023_c001.tif
Analyzed  New_L10_Sum_t024_c001.tif
Analyzed  New_L10_Sum_t025_c001.tif
Analyzed  New_L10_Sum_t026_c001.tif
Analyzed  New_L10_Sum_t027_c001.tif
Analyzed  New_L10_Sum_t028_c001.tif
Analyzed  New_L10_Sum_t029_c001.tif
Analyzed background dataset: bkgd_0
```

(c) You can add additional fit objects to the molecule project by repeating steps (a) and (b) described above.

7. If you changed any settings of a fit (by selecting *Fitting* → *Fits* → *Edit fit*) and want it to be performed again, select the fit in the left column and go to *Fitting* → *Perform Fits* → *Perform fit*.

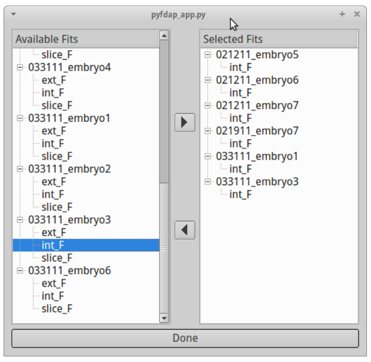
8. If you have added and fitted multiple embryo objects (FDAP measurements) and wish to find the average fit over all embryo objects, go to *Statistics* → *Plotting* → *Plot average fit* (see Section 3.3 for details).

3.2.2 Loading a pre-analyzed dataset

Launch PyFDAP, go to *File* → *Open Molecule*, and select the file *TestDataset/results/TestDataset_20min.pk*. You have now successfully loaded a molecule project including one embryo object (FDAP dataset) and one background dataset. You can now try out all features of PyFDAP including all plotting functions.

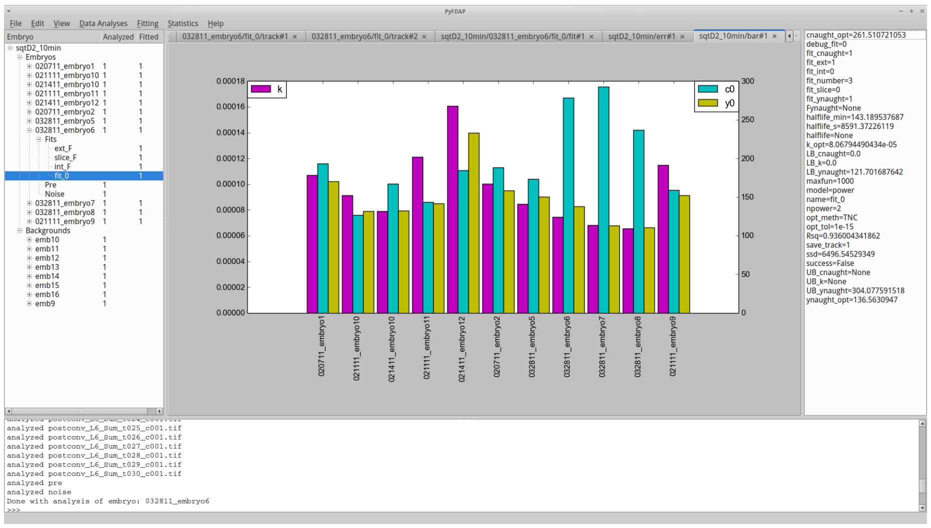
3.3 Making use of statistical functions in PyFDAP

PyFDAP comes with a few statistical tools for data averaging and analysis. To average the fits from multiple embryo objects (FDAP measurements), go to *Statistics → Average Molecule*. A pop-up window will ask you to select fits from different embryo objects:



You can add the fits that you want to be considered for averaging to the selection on the right-hand side by double-clicking on the particular fit or by using the arrow buttons on the screen. You can also remove fits from the selection by double-clicking or by using the arrow buttons. Note that for averaging to work, you can only select fits of the same region, e.g. you cannot average a fit for the extracellular region with one for the intracellular region. It is also not possible to let two fits of the same embryo object contribute to the averaged fit.

After selecting the fits that you want to include for averaging, press *Done*. PyFDAP will automatically compute averages of all important fitting parameters and display them in the property list on the right-hand side. Details on how these averages are computed can be found in Section 6.4. After averaging a selection of fits, you can use PyFDAP's bar plot functions to compare fitting results from different embryos. Go to *Statistics → Plotting* and choose between *Plot ks by fit*, *Plot y0s by fit*, *Plot c0s by fit* to plot each of the parameters by fit in a bar plot, or choose *Plot all parameters by fit* to plot all three optimal parameters by fit.

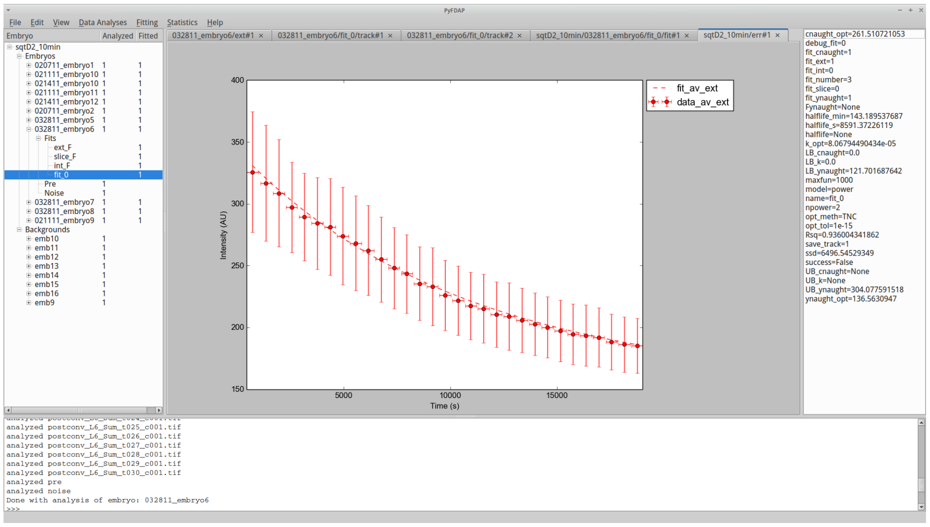


```
cnaught_opt=261.510721053
debug_fit=0
fit_cnaught=1
fit_ext=1
fit_int=0
fit_number=3
fit_slice=0
fit_ynaught=1
Fynaught=None
halflife_min=143.189537687
halflife_s=8591.37226119
halflife=None
k_opt=8.06794490434e-05
LB_cnaught=0.0
LB_k=0.0
LB_ynaught=121.701687642
maxfun=1000
model=power
name=fit_0
npower=2
opt_meth=TNC
opt_tol=1e-15
Rsq=0.936004341862
save_track=1
ssd=6496.54529349
success=False
UB_cnaught=None
UB_k=None
UB_ynaught=304.077591518
ynaught_opt=136.5630947
```

```
analyzed postconv_L6_Sum_t025_c001.tif
analyzed postconv_L6_Sum_t026_c001.tif
analyzed postconv_L6_Sum_t027_c001.tif
analyzed postconv_L6_Sum_t028_c001.tif
analyzed postconv_L6_Sum_t029_c001.tif
analyzed postconv_L6_Sum_t030_c001.tif
analyzed pre
analyzed noise
Done with analysis of embryo: 032811_embryo6
>>>
```

This plot allows you to identify fits that produce parameters strongly deviating from the mean. You can then go back to those fits and adjust the fitting parameters to optimize your final result.

You can also plot the averaged time-dependent fluorescence decay data as error bar plots for unnormalized data or for data normalized between values of 0 and 1. To generate these plots, go to *Statistics* → *Plotting* → *Plot average fit* or *Statistics* → *Plotting* → *Plot normed average fit*.



Embryo	Analyzed	Fitted
sqtD2_10min		
Embryos		
020711_embryo1	1	1
021111_embryo10	1	1
021411_embryo10	1	1
021111_embryo11	1	1
021411_embryo12	1	1
020711_embryo2	1	1
032811_embryo5	1	1
032811_embryo6	1	1
Fits		
ext_F		1
slice_F		1
int_F		1
fit_0		1
Pre	1	
Noise	1	
032811_embryo7	1	1
032811_embryo8	1	1
021111_embryo9	1	1
Backgrounds		
emb10	1	
emb11	1	
emb12	1	
emb13	1	
emb14	1	
emb15	1	
emb16	1	
emb9	1	

```
cnaught_opt=261.510721053
debug_fit=0
fit_cnaught=1
fit_ext=1
fit_int=0
fit_number=3
fit_slice=0
fit_ynaught=1
Fynaught=None
halflife_min=143.189537687
halflife_s=8591.37226119
halflife=None
k_opt=8.06794490434e-05
LB_cnaught=0.0
LB_k=0.0
LB_ynaught=121.701687642
maxfun=1000
model=power
name=fit_0
npower=2
opt_meth=TNC
opt_tol=1e-15
Rsq=0.936004341862
save_track=1
ssd=6496.54529349
success=False
UB_cnaught=None
UB_k=None
UB_ynaught=304.077591518
ynaught_opt=136.5630947
```

```
analyzed postconv_L6_Sum_t025_c001.tif
analyzed postconv_L6_Sum_t026_c001.tif
analyzed postconv_L6_Sum_t027_c001.tif
analyzed postconv_L6_Sum_t028_c001.tif
analyzed postconv_L6_Sum_t029_c001.tif
analyzed postconv_L6_Sum_t030_c001.tif
analyzed pre
analyzed noise
Done with analysis of embryo: 032811_embryo6
>>>
```

Mathematical details for error bar computation and data normalization can be found in Section 6.4.

Since Version 1.1 PyFDAP also includes statistical tests for comparison between different samples and a normality test. To compare two samples, load two different molecule files, then select the test you want to perform via *Statistics* → *Tests*. Select the two molecules you want to compare and click *Done*. The test statistic will then be displayed in the terminal window. Note: Both molecules need to be averaged before via *Statistics* → *Average molecule*. More details about the implemented statistical tests can be found in Section 6.4.1

3.4 Saving results from PyFDAP

PyFDAP offers multiple ways to save and share FDAP project data and details such as plots, videos, analysis settings, and whole molecule projects.

3.4.1 Saving figures and movies

In *Data Analysis* → *Plotting*, users can find plotting commands for

- Data and background images for the whole region (slice) as well as for extra- and intracellular domains
- Masked images for the whole region (slice) as well as for extra- and intracellular domains
- Masks for the whole region (slice) as well as for extra- and intracellular domains
- Analysis results for all three regions including background values

Moreover, users can plot fitting results and the fitting progress under *Fitting* → *Plotting*. Single plot frames can be saved as *.png, *.pdf, *.eps, *.jpg, *.pgf, *.ps, *.rgba, *.svg, or *.tif. In order to edit the plots using a vector graphics software, we recommend saving images as *.pdf or *.eps files.

PyFDAP also allows users to export image series (such as the fitting progress) as *.mpg or *.avi movies for presentation purposes. Note that PyFDAP does not automatically provide the necessary package for the conversion of image files to movie files; more information about the installation process to enable video output can be found in Section 2.3.

3.4.2 Saving molecule and embryo files

Users can save their molecule sessions to JavaScript Object Notation (JSON) object files. These object files follow the logical hierarchical structure explained in Section 4 and contain all of the data used for the FDAP analysis as well as the fitting results. The molecule and embryo files can be re-loaded into PyFDAP to enable researchers to continue working on a session and to facilitate collaboration among researchers in different locations.

3.4.3 Saving plots and results as .csv files

Plots as well as molecule and embryo objects can also be saved as comma-separated value files that can then be read into other plotting or analysis software such as Excel or Matlab. The molecule and embryo *.csv files follow the hierarchical system of the JSON files (see above). Note that image data will not be exported to *.csv files.

4 DATA STRUCTURE

PyFDAP provides a hierarchical object structure to organize the datasets obtained from FDAP experiments and to facilitate data navigation (Figure 1).

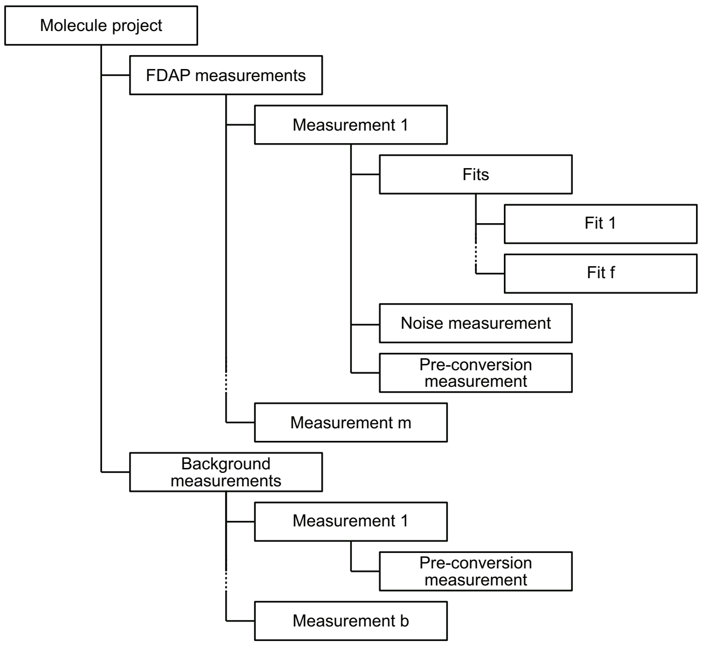


Figure 1: Hierarchical PyFDAP data structure for FDAP experiments. Experiments are grouped into a main molecule project and divided into FDAP (1 to m) and background (1 to b) pre- and post-conversion measurements. Each FDAP measurement can have multiple fits (1 to f) with different fitting options.

Molecule projects: Replicate experiments with the same protein are grouped into a main molecule project. PyFDAP can handle multiple molecule projects in one session.

FDAP measurements: Replicate experiments are divided into FDAP and background pre- and post-conversion measurements. Intra- and extracellular protein stability can be different, and PyFDAP can import a second dataset that counter-labels intra- or extracellular space. The separation of fluorescence intensities into intra- or extracellular masks is performed using the Otsu binarization algorithm (Otsu, 1979). The masks and corresponding datasets can be investigated inside the PyFDAP GUI by clicking on *Data Analysis* → *Plotting* → *Background Dataset*. The masks are applied to the images of the photoconverted signal, and the average intensities in the intra- and extracellular domains and in the entire image are calculated. Each PyFDAP embryo dataset (FDAP measurement) can have multiple fits for various regions, using different fitting parameters and different data points to allow maximum flexibility. The fits are automatically included in the PyFDAP data structure.

Noise measurements: Noise measurements can be imported for each embryo dataset and can be used to calculate estimates for the baseline of the fit (see Section 6.2).

Pre-conversion measurements: Pre-conversion intensity measurements provide information about the levels of autofluorescence and can be used to calculate estimates for the baseline of the fit (see Section 6.2).

Background measurements: Background measurements provide information about the levels of autofluorescence after mock-photoconversion in the presence of unlabeled variants of the protein of interest and can be used to calculate estimates for the baseline of the fit (see Section 6.2).

5 PERFORMANCE

We tested PyFDAP on various system configurations and ran a test script measuring the total operation time. The test script contained the following operations:

1. Open a test molecule file
2. Analyze a single FDAP dataset with all necessary additional data
3. Analyze a background dataset with all necessary additional data
4. Perform three fits for the intra- and extracellular and slice data

The dataset used for this performance test is freely available from `http://people.tuebingen.mpg.de/mueller-lab/`, and the results of our performance tests are listed in Table 2.

System	**OS**	**CPU**	**Memory**	**Operational Time**
Thinkpad x230	Xubuntu 14.04	Intel(R) Core(TM) i7-3520M, 2.90 GHz	8 GB	55 s
MacBookPro7.1	Mac OS X 10.9.4	Intel(R) Core(TM) 2 Duo-P8600, 2.40 GHz	4 GB	88 s
MacBookPro8.1	Mac OS X 10.9.5	Intel(R) Core(TM) i5-2410M , 2.30 GHz	4 GB	55 s

Table 2: Performance test results of PyFDAP.

6 MATHEMATICAL BACKGROUND

6.1 Decay models

PyFDAP supports two different decay models: Linear- and non-linear decay. Linear decay is given by the ordinary differential equation (ODE)

$$\frac{\mathrm{d}c}{\mathrm{d}t} = -kc$$

where c is the concentration of a molecule and k is the rate constant of the decay. Since we assume that the level of fluorescence is proportional to the molecule concentration, we can substitute the concentration with fluorescence intensity. Solving this ODE results in

$$c(t) = c_0 e^{-kt} + y_0$$

where $c(t)$ is the concentration of a molecule at time t, $c(0) = c_0$ is the concentration at time $t = 0$, and y_0 is the baseline fluorescence intensity to which the population of decaying molecules converges. In terms of fluorescence intensity, y_0 resembles the baseline level of noise and autofluorescence. From k we can then compute the molecule's half-life τ by

$$\tau = \frac{\ln(2)}{k}.$$

Some molecules are proposed to decay non-linearly (Eldar *et al.*, 2003), and we have

$$\frac{\mathrm{d}c}{\mathrm{d}t} = -kc^n$$

where $n > 1$ is the degree of non-linearity and k is the decay rate constant of the molecule. We can solve this ODE and obtain the power-law solution

$$c(t) = \left(c_0^{1-n} - kt(1-n)\right)^{\frac{1}{1-n}} + y_0.$$

For the case of a non-linear decay model, we compute the molecule's half-life by

$$\tau = \frac{(2^{n-1} - 1)c_0^{1-n}}{k(n-1)}.$$

6.2 Estimation of initial guesses and bounds for variables

PyFDAP offers multiple options to calculate initial guesses and bounds for variables that are used by the fitting algorithms to obtain biologically reasonable estimates based on noise, pre-conversion, and background measurements (see Section 4).

Initial guess for the estimation of c_0: A good estimate for c_0 is the difference between the pre-conversion and the first post-conversion image, i.e.

$$I_{\text{post}}(t_{\text{start}}) - I_{\text{pre}},$$

where I_{post} and I_{pre} are the fluorescence intensities after and before photoconversion, respectively, and t_{start} is the time at which the first image was taken.

Initial guess for the estimation of the baseline y_0: PyFDAP offers the two presets $I_{\text{post}}(t_{\text{start}})$ and $I_{\text{post}}(t_{\text{end}})$, where t_{end} is the time at which the last image was taken and where protein decay should be almost complete. Our tests showed that the optimization algorithms worked well if $y_{0,\text{opt}}$ is approached from above using $I_{\text{post}}(t_{\text{start}})$ as the initial guess for y_0.

Estimation of the lower bound for the baseline y_0: This estimate is a crucial part of the fitting process. PyFDAP offers several algorithms to perform this estimation based on the amount and quality of the data available.

- The simplest estimate of the lower bound of y_0 is the average background noise of the measurements $\bar{N}$. Due to autofluorescence of the samples, this estimate is generally too low, but it serves as the lower bound of the lower bounds of y_0.
- Alternatively, the lower bound of the baseline y_0 can be estimated from the average level of autofluorescence represented by

  $$\bar{B}_{\mathrm{pre}_r} = \frac{\sum_{j=1}^{b} B_{\mathrm{pre}_{j,r}}}{b},$$

  where $r \in \{\text{intracellular, extracellular, entire domain}\}$ is the investigated region, and $j \in \{1, ..., b\}$ are the indices of background pre-conversion datasets with intensities $B_{\mathrm{pre}_{j,r}}$.
- PyFDAP also offers the possibility to use the average background intensity as the lower bound of the baseline y_0:

  $$\bar{B}_r = \frac{\sum_{j=1}^{b} \bar{B}_{j,r}}{b},$$

  where $\bar{B}_{j,r}$ is the mean intensity in region r of a background dataset over all data points given by

  $$\bar{B}_{j,r} = \frac{\sum_{l=1}^{T} B(t_l)_{j,r} + B_{\mathrm{pre}_{j,r}}}{T+1}.$$

  Here, t_l with $l \in \{1, ..., T\}$ is the time when the l-th image was taken and T is the number of post-conversion images.
- PyFDAP includes a special weighting function F (Müller *et al.*, 2012) given by

  $$F_{i,r} = \frac{1}{b} \sum_{j=1}^{b} \min_t \left(\frac{B_{j,r}(t) - N_i}{B_{\mathrm{pre}_{j,r}} - N_i} \right),$$

  where i is the current FDAP measurement, r is the investigated region, and j is the index of background datasets with intensities $B(t)$ at time t. Here, the noise measurement of measurement i is given by N_i. Using the function F, users can compute the lower bound of the baseline $y_{0_{i,r}}$ for measurement i and region r by

  $$y_{0_{i,r}} \geq F_{i,r} \cdot (I_{\mathrm{pre}_{i,r}} - N_i) + N_i,$$

  where $I_{\mathrm{pre}_{i,r}}$ denotes the pre-conversion intensity of the FDAP measurement i in region r.

Method	**Name in PyFDAP**	**Type**	**Reference**
Bounded methods			
Limited-memory BFGS	L-BFGS-B	quasi-Newton	Byrd *et al.* (1995)
Truncated Newton Conjugate	TNC	Newton conjugate	Nash (1984)
Sequential Least Squares Programming	SLSQP	sequential quadratic	Kraft (1988)
Brute force	brute	brute force	SciPy Reference Guide
Unbounded methods			
Nelder-Mead	Nelder-Mead	simplex	Nelder and Mead (1965)
Broyden-Fletcher-Goldfarb-Shanno	BFGS	quasi-Newton	Broyden (1970); Goldfarb (1970); Fletcher (1970); Shanno (1970)
Nonlinear Conjugate Gradient	CG	Newton conjugate	Polak and Ribière (1969)

Table 3: List of optimization algorithms in PyFDAP.

6.3 Optimization algorithms

PyFDAP comes with a wide selection of optimization algorithms taken from the SciPy optimize package (http://docs.scipy.org/doc/scipy/reference/optimize.html) (Nelder and Mead (1965); Polak and Ribière (1969); Broyden (1970); Goldfarb (1970); Fletcher (1970); Shanno (1970); Nash (1984); Kraft (1988); Byrd *et al.* (1995); Nocedal and Wright (2006)). A list of all optimization algorithms available in PyFDAP can be found in Table 3.

6.4 Statistics

PyFDAP can average over multiple fits from different embryo objects (FDAP measurements). Details of how to select fits for averaging are described in Section 3.3.

PyFDAP averages the optimal parameters for k, y_0, c_0, and protein half-lives τ through an arithmetic mean. For example, the average decay rate constant $\bar{k}$ is obtained by

$$\bar{k} = \frac{\sum\limits_{i=1}^{\tilde{m}} k_i}{\tilde{m}},$$

where $\tilde{m}$ is the number of fits to be averaged. The average half-life $\bar{\tau}$ can be computed in two ways resulting in different average half-lives. PyFDAP computes the average half-life $\bar{\tau}$ through the arithmetic mean given by

$$\bar{\tau} = \frac{\sum\limits_{i=1}^{\tilde{m}} \tau_i}{\tilde{m}}.$$

For the linear decay model, this yields

$$\bar{\tau} = \frac{1}{\tilde{m}} \sum_{i=1}^{\tilde{m}} \frac{\ln(2)}{k_i}, \tag{1}$$

and in the case of the non-linear decay model we obtain

$$\bar{\tau} = \frac{1}{\tilde{m}} \sum_{i=1}^{\tilde{m}} \frac{(2^{n-1}-1)c_{0,i}^{1-n}}{k_i(n-1)}. \tag{2}$$

However, computing the average half-life $\bar{\tau}$ directly from the average decay rate $\bar{k}$ yields

$$\bar{\tau} = \frac{\ln(2)}{\frac{1}{\tilde{m}} \sum\limits_{i=1}^{\tilde{m}} k_i}, \tag{3}$$

for the linear decay model and

$$\bar{\tau} = \frac{(2^{n-1}-1)\frac{1}{\tilde{m}} \sum\limits_{i=1}^{\tilde{m}} c_{0,i}^{1-n}}{\frac{1}{\tilde{m}} \sum\limits_{i=1}^{\tilde{m}} k_i(n-1)}. \tag{4}$$

in case of the non-linear decay model. It is obvious that equations 1 and 3 as well as equations 2 and 4 do not produce the same half-lives, and the user needs to decide which way of half-life computation is appropriate for the application.

PyFDAP can produce different error bar plots for each averaged region. Clicking on *Statistics → Plotting → Plot average fit* will result in a plot in which each average data point $\bar{c}(t_j)$ is computed as the arithmetic mean

$$\bar{c}(t_j) = \frac{1}{\tilde{m}} \sum_{i=1}^{\tilde{m}} c_i(t_j).$$

Error bars are computed as the standard deviation for each time t_j. Clicking on *Statistics → Plotting → Plot normed average fit* returns a plot in which all data points are normalized between values of 0 and 1. The normalization is performed by subtracting the baseline value $y_{0,i}$ from each data point and dividing the result by $c_{0,i}$, i.e.

$$\tilde{c_i}(t_j) = \frac{c_i(t_j) - y_{0,i}}{c_{0,i}},$$

where $\tilde{c_i}(t_j)$ is the normalized data point at time t_j. This normalization facilitates the comparison of decay curve shapes, but it substantially changes the meaning of the error bars. Since all data series are pinned to a value of 1 at their first time point, the standard deviation vanishes for this data point. The following data points will generally produce increasing error bars since the decay curves generally diverge. The length of the normalized error bars can be interpreted as the extent to which the decay curves diverge throughout the experiments.

6.4.1 Statistical tests

PyFDAP offers multiple statistical tests both to

- test the normality of the distribution of degradation constants,
- and compare two different molecule files and determine of the resulting degradation constants are significantly different.

Method	**Requires normality**	**Reference**
Standard t-test	Yes	Student (1908)
Welch's t-test	Yes	Welch (1947)
Mann-Whitney U Test	No	Mann and Whitney (1947)
Wilcoxon	No	Wilcoxon (1945)

Table 4: List of statistical comparison methods in PyFDAP.

A multitude of statistical tests such as the Students t-test (Student, 1908) require normally distributed samples. One way to test this is using the Shapiro-Wilk test (Shapiro and Wilk, 1965), which was recently found to have the best sensibility compared to other common normality tests (Razali and Wah, 2011).

PyFDAP offers four methods to compare the degradation of two molecules. All methods are summarized in table 4.

7 ACKNOWLEDGMENTS

We thank Katherine Rogers and Gary Soh for useful suggestions and for testing PyFDAP. We thank Katherine Rogers and Alex Schier for providing the test dataset.

8 LIST OF FREQUENTLY USED ABBREVIATIONS AND VARIABLES

Name	Description
Abbreviations	
LB	Lower bound
UB	Upper bound
ODE	Ordinary differential equation
SSD	Sum of squared differences
Variables	
b	Number of background measurements
B	Background intensity
c	Molecule concentration
c_0	Initial molecule concentration
$\tilde{c}$	Normalized molecule concentration
f	Number of fits
F	Weighting function for the estimation of the lower bound of y_0
i	Control variable
I	Fluorescence intensity
j	Control variable
k	Molecule decay rate constant
l	Control variable
m	Number of measurements
$\tilde{m}$	Number of selected fits for averaging
n	Degree of non-linearity of molecule decay
N	Noise intensity
r	Region of measurement
t	Time
T	Number of post-conversion frames
τ	Molecule half-life
x_0	Initial parameter guess
y_0	Molecule decay baseline
Variable subscripts	
ext	Extracellular
int	Intracellular
post	After photoconversion
pre	Before photoconversion
slice	Entire optical slice

9 REFERENCES

Broyden, C. G. (1970). The convergence of a class of double-rank minimization algorithms. 1. General considerations. *IMA Journal of Applied Mathematics*, **6**(1), 76–90.

Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. (1995). A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, **16**(5), 1190–1208.

Eldar, A., Rosin, D., Shilo, B. Z., and Barkai, N. (2003). Self-enhanced ligand degradation underlies robustness of morphogen gradients. *Dev. Cell*, **5**(4), 635–646.

Fletcher, R. (1970). A new approach to variable metric algorithms. *The Computer Journal*, **13**(3), 317–322.

Goldfarb, D. (1970). A family of variable-metric methods derived by variational means. *Mathematics of Computation*, **24**, 23–26.

Kraft, D. (1988). A software package for sequential quadratic programming. *DFVLR-FB 88-28, Köln, Germany*.

Mann, H. and Whitney, D. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*.

Millman, K. J. and Aivazis, M. (2011). Python for scientists and engineers. *Computing in Science & Engineering*, **13**(2), 9–12.

Müller, P., Rogers, K. W., Jordan, B. M., Lee, J. S., Robson, D., Ramanathan, S., and Schier, A. F. (2012). Differential diffusivity of Nodal and Lefty underlies a reaction-diffusion patterning system. *Science*, **336**(6082), 721–724.

Nash, S. G. (1984). Newton-type minimization via the Lanczos method. *SIAM Journal on Numerical Analysis*, **21**(4), 770–788.

Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, **7**(4), 308–313.

Nocedal, J. and Wright, S. J. (2006). *Numerical optimization*. Springer series in operations research and financial engineering. Springer, New York, 2. ed. edition.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Trans Syst., Man, Cybern.*, **9**(1), 62–66.

Polak, E. and Ribière, G. (1969). Note sur la convergence de méthodes de directions conjuguées. *Revue Française d'Informatique et de Recherche Opérationnelle*, **16**, 35–43.

Razali, N. M. and Wah, Y. B. (2011). Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests. **2**, 21–33.

Rogers, K. W., Bläßle, A., Schier, A. F., and Müller, P. (2014). Measuring protein stability in living zebrafish embryos using Fluorescence Decay After Photoconversion (FDAP). *Journal of Visualized Experiments, doi:10.3791/52266*.

Shanno, D. F. (1970). Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, **24**(111), 647–656.

Shapiro, S. and Wilk, M. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, **52**, 591–611.

Student (1908). The probable error of a mean. *Biometrika*, **6**, 1–25.

Welch, B. (1947). The generalization ofstudent's' problem when several different population variances are involved. *Biometrika*, (2).

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, **1**, 80–83.