Program TwoFocus2FCS.m

Syntax:

```
[res head] = TwoFocus2FCS(name, maxtime, cutoff, photons);
```

Description:

`TwoFocus2FCS` calculates auto- and cross-correlation curves from t3r, pt3 or ht3 raw data taken in dual-focus mode into. By default, raw data are processed in bunches of 10^6 photons, and for each bunch, correlation functions are calculated and stored separately.

Input variables:

`name` – full name of the data file including directory.

`maxtime` – maximum lag time until which correlation functions are calculated. Default value is 3 s.

`para` – only applicable for ht3-file: **`para(1)`** is minimum lag time, **`para(2)`** bin width of TCSPC curve, both values in ns. Default is **`para = [50, 0.002]`**.

`cutoff` – allows to cut out the first **`cutoff`** ns in each time window used for correlation function calculation.

`photons` – number of photons in which raw data are divided for successive processing; default is 10^6

Output variables:

`res` – structure containing:
`res.bin` and **`res.tcspcdata`** are the calculated TCSPC histograms, where **`bin`** is a vector of TCSPC channels and **`tcspcdata(:,1)`** is the TCSPC curve from the first and **`tcspcdata(:,2)`** from the second detector;
`res.bin` and **`res.tcspc`** are again the TCSPC data, but now divided into for curves so that **`tcspc(:,1)`** corresponds to the first laser and first detector, **`tcspc(:,2)`** to the second laser and first detector, **`tcspc(:,3)`** to the first laser and second detector, and **`tcspc(:,4)`** to the second laser and second detector; **`tau`** is given in ns;
`res.autotime` is the lag-time axis of the correlation functions, and **`res.auto`** is a multidimenisonal array containing the calculated correlation functions, whereas **`auto(j,p,q,s)`** is s^{th} packet of

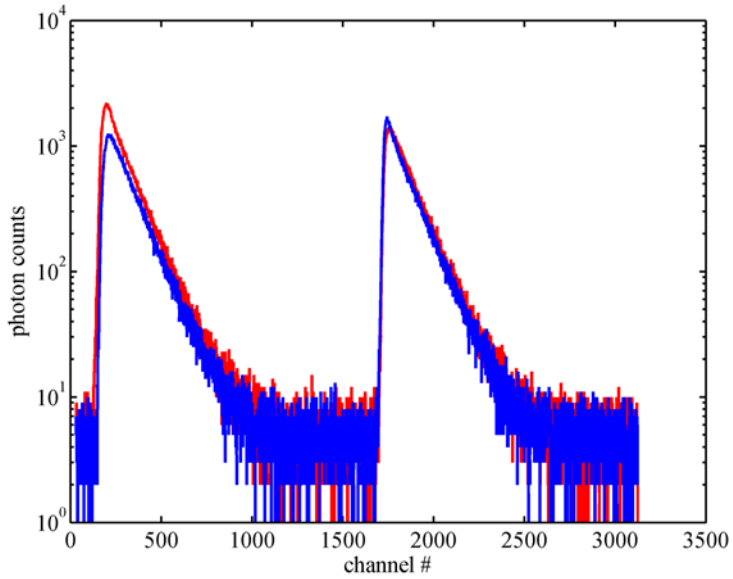
the the j^{th} lag-time value of the correlation function of the p^{th} channel against the q^{th} channel, where the channels are: first channel corresponds to first laser and first detector, second channel to the second laser and first detector, third channel to the first laser and second detector, and fourth channel to the second laser and second detector;

`res.rate(p,s)` and **`res.time(s)`** are the number of counted photons in the p^{th} channel and the measurement time for the s^{th} bunch of processed photons.

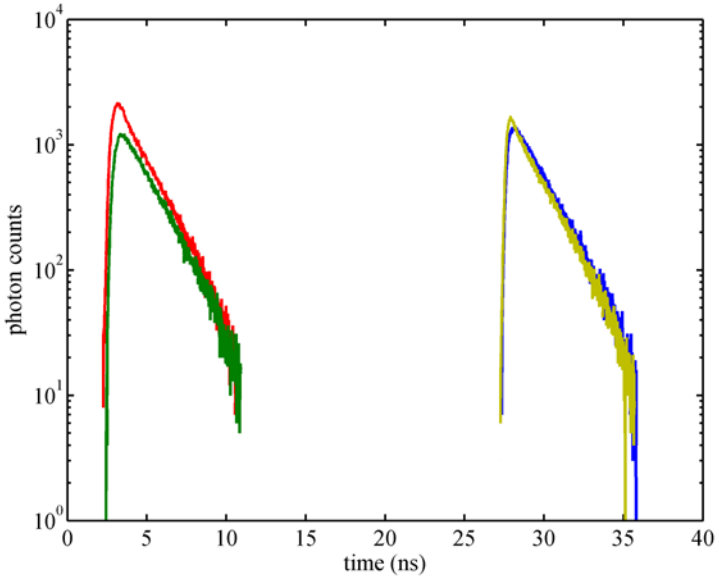
`head` – structure containing all the header information from raw file.

Example:

```
[res, head] = TwoFocus2FCS('AttoSat2Foci.pt3');
semilogy(res.bin,res.tcspcdata);
xlabel('channel #'); ylabel('photon counts')
```

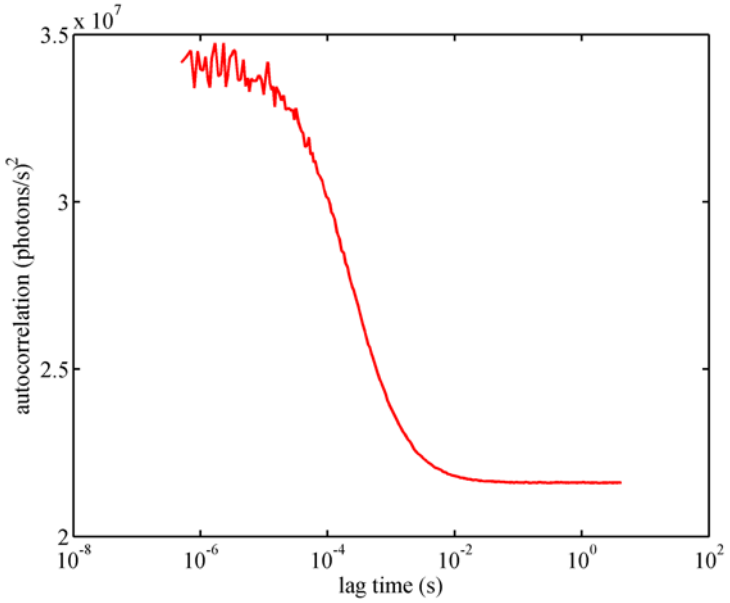



```
semilogy(res.tau,res.tcspc);
xlabel('time (ns)'); ylabel('photon counts')
```


photon counts
time (ns)
10^0
10^1
10^2
10^3
10^4
0
5
10
15
20
25
30
35
40

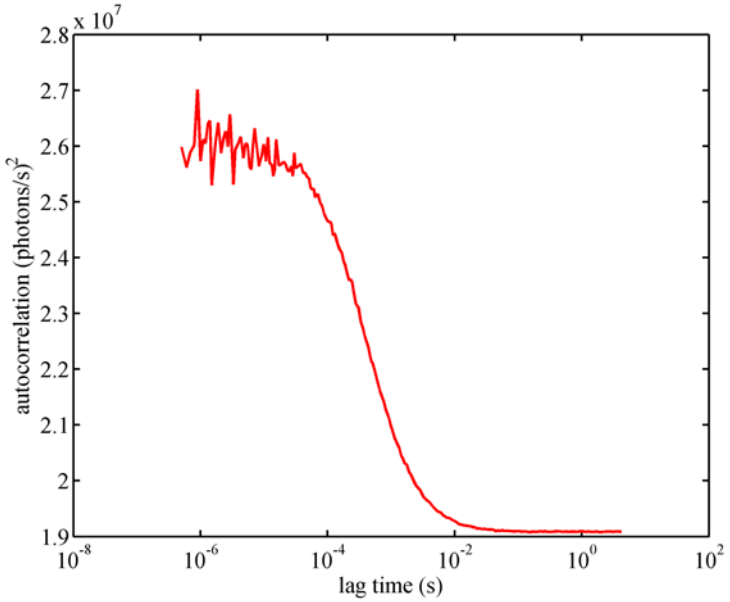
Plotting the total autocorrelation for the first focus:

```
semilogx(res.autotime,mean(res.auto(:,1,3,:)+res.auto(:,3,1,:),4))
xlabel('lag time (s)'); ylabel('autocorrelation (photons/s)^2')
```



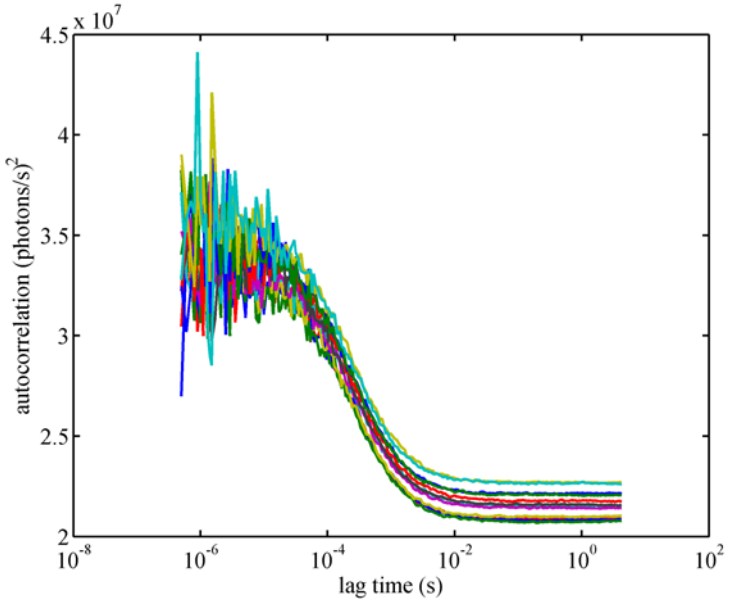
Plotting the total cross-correlation from first focus to second focus:

```
semilogx(res.autotime,mean(res.auto(:,1,4,:)+res.auto(:,3,2,:),4))
xlabel('lag time (s)'); ylabel('autocorrelation (photons/s)^2')
```



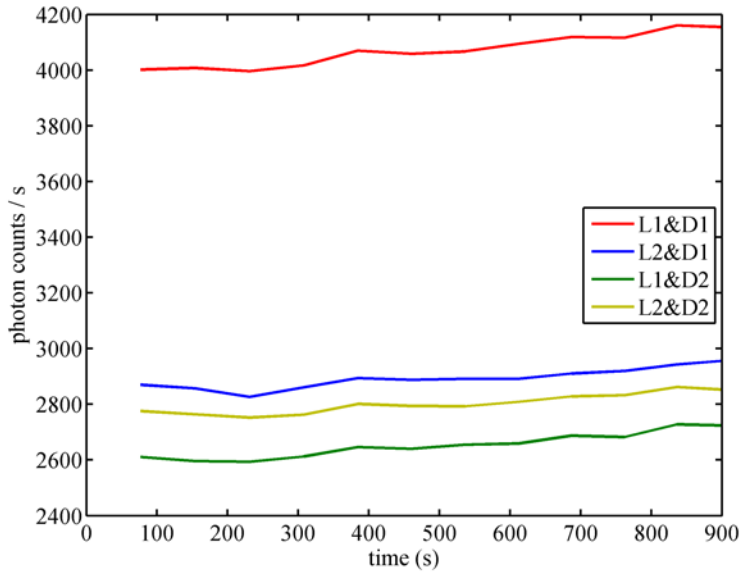
Plotting all autocorrelations for the first focus:

```
semilogx(res.autotime,squeeze(res.auto(:,1,3,:)+res.auto(:,3,1,:)))
xlabel('lag time (s)'); ylabel('autocorrelation (photons/s)^2')
```



Plotting count rate:

```
plot(cumsum(res.time),res.rate)
xlabel('time (s)'); ylabel('photon counts / s')
legend({'L1&D1','L2&D1','L1&D2','L2&D2'},'Location','East')
```



The four curves correspond to the four virtual channels (first laser & first detector, second laser & first detector etc.).

Program FCSCrossRead.m

Syntax:

```
[y, t] = FCSCrossRead(name, t)
```

Description:

`FCSCrossRead` reduces the full correlation functions as calculated by TwoFocus2FCS into the physically interesting auto-and cross-correlation functions.

Input variables:

`name` – full name of the data file including directory, containing the **`res`**-structure as outputted by TwoFocus2FCS. Alternatively, name can be the structure **`res`** itself.

`t` – optional parameter defining lag-time cut-off values: the returned correlation functions will be in the range between **`t(1)`** and **`t(end)`**. Default is the whole available lag time range of the correlation functions.

Output variables:

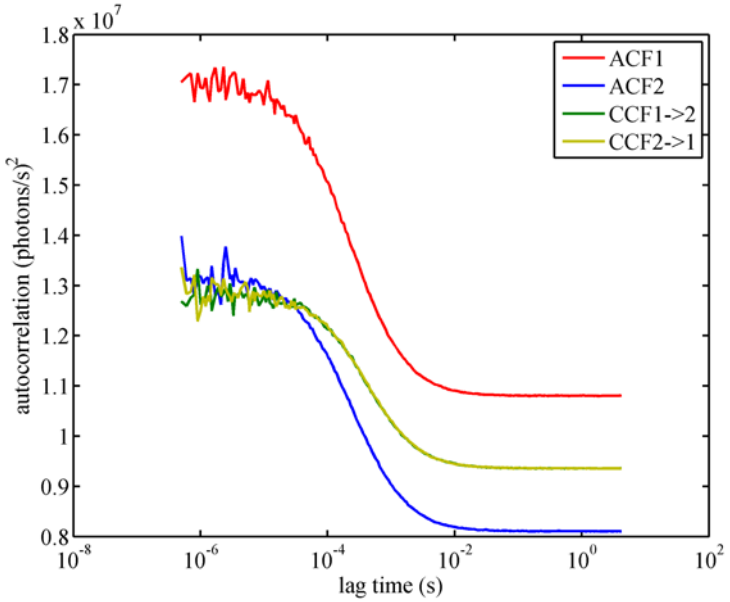
`y` – correlation function: **`y(:,1,:)`** are the full autocorrelations of the first focus, **`y(:,2,:)`** of the second, **`y(:,3,:)`** and **`y(:,4,:)`** are the cross-correlations between foci, where **`y(:,3,:)`** is from first to second focus (i.e. first photon from first and subsequent photon from second focus) and **`y(:,4,:)`** from second to first focus. The third dimension of **`y`** refers again to subsequent bunches of measured 10^6 photons.

`t` – lag-time vector.

Example:

Using the **`res`** structure as calculated by **`TwoFocus2FCS`**, type

```
[y, t] = FCSCrossRead(res);
semilogx(t,mean(y,3))
xlabel('lag time (s)'); ylabel('autocorrelation (photons/s)^2')
legend({'ACF1','ACF2','CCF1->2','CCF2->1'})
```



Program FCSFit.m

Syntax:

```
global pd
pd = 0.1;
[dc w0 a0 triplet c velo err z] = ...
FCSFit(name,p,expflag,bootstrap,para,bounds,pmin,pmax);
```

Description:

FCSFit is a general routine for fitting dual-focus fluorescence correlation spectroscopy (2fFCS) measurements. It operates on already calculated auto- and cross-correlation curves. Before calling the routine, one has to have defined the global variable **pd** with the command **global pd**. This variably should contain initial guess values for the inverse diffusion coefficient(s) (more precisely, $10^{-8}/D$[cm^2/s]) and exponential times of potential photophysical relaxation processes. Thus, **pd** is a vector of variable length, which structure and bounds can be specified by the input variables **expflag** and **bounds**, see below.

Input variables:

name – **name** is either the name of a file on the hard disc containing a structure **res** as generated by TwoFocus2FCS, either that structure **res** itself, either a structure having elements **name.y** and **name.t**, where **t** is a vector of autocorrelation lag-times, and **y(:,1,:)** is the autocorrelation from the first focus, **y(:,2,:)** is the autocorrelation from the second focus, **y(:,3,:)** is the cross-correlation from the first to the second focus, and **y(:,4,:)** is the cross-correlation from the second to the first focus, see also function **FCSCrossRead**. The third dimension of **y** is optional, but if **y** is a three-dimensional array, than it assumed that the third dimension counts over subsequent measurements, for example **y(:,1,1)** is the first measurement, **y(:,1,2)** the second and so on of a set of many measurements. Breaking a single FCS measurement down into several shorter-lasting measurements is useful for estimating standard deviations of the fitted parameters (using the **bootstrap** variable), or to remove measurements with large outliers.

p – vector containing initial guess values for the system fit parameters. **p** has to have, at minimum, two entries: **p(1)** is a guess value for the beam waist radius w_0, and **p(2)** a guess value for the confocal pinhole parameter R_0. If one assumes the presence of fluid flow, then **p(3:5)** are guess values of the flow velocity vector, whereas **p(3)** is the flow velocity along the direction connecting the foci but perpendicular to the optical axis; **p(4)** is the flow velocity along a direction perpendicular both to the focus-connecting line and the optical axis; and **p(5)** is the flow velocity along the

optical axis (all in units of µm/s). If one assigns values only to **p(3)** or **p(3:4)**, the routine assumes that the flow velocity along the remaining directions is zero. Default is **p = [400, 150]**.

2D-FCS fit: One can force a 2D-FCS-fit (diffusion within membranes) by setting **p(3)** equal to the imaginary unit, i.e. **p(3) = 1i**. In that case the **p(1)** and **p(2)** correspond to the two beam waist radius values, which are usually different in the case of 2D diffusion due to slightly different positions of beam foci along the optical axis.

expflag – tells the routine how many exponents have to be fitted to the correlation curve for taking into account photophysical relaxation processes. The routine assumes that **pd(end-expflag+1:end)** are initial guess values for these exponential decay times (in µs), whereas the other **pd(1:end-expflag)** values are guess values for the inverse diffusion coefficients, thus assuming that there are **1:length(pd)-expflag** differently diffusing species in the sample. Default value of **expflag** is **[]**.

bootstrap – value which is important if the data are three-dimensional, i.e. when one processes many succeeding measurements. The value of **bootstrap** tells the routine how often a bootstrap algorithm is applied for obtaining standard deviations of the fit parameters. A reasonable number is equal to the number of correlation functions one has, i.e. **size(y,3)**. If one has *N* correlation curves, the bootstrap algorithm randomly chooses *N* out of these *N* curves (thus some curves appear more than once), sums them up and fits them. This is repeated **bootstrap** times. Default value is **[]**, in which case all data are summed over the third dimension, i.e. the routine fits only **sum(y,3)**.

para – vector specifying the optical setup: **para(1)** is the radius of the confocal aperture divided by magnification; **para(2)** is the excitation wavelength divided by the refractive index of the immersion medium; **para(3)** is the centre emission wavelength divided by the refractive index of the immersion medium; **para(4)** is the distance between foci. All values have to be given in **nanometers**! The default values are defined at the beginning of the FCSFit.m file and are currently set to:

```
para = [75e3/60, [640 670]/1.33, 414].
```

bounds – specifies bounds on fitting the variable **pd**. bounds must be a **2×length(pd)** matrix: the first line of the matrix defines the lower bounds, and the second line the upper bounds. Acceptable entry values are also ±**inf**, but all entries in **bounds(1,:)** have to be smaller or equal than the corresponding entries in **bounds(2,:)**. Optional value is **[]** (no restrictions on **pd** at all).

pmin – lower bounds on the allowed values for **p** during fitting. Optional value is **[]** (no lower bounds).

Output variables:

dc – fitted values of diffusion coefficient(s) in cm^2/s.

w0 – fitted value of beam waist radius (guess value was **p(1)**).

a0 – fitted value of R_0 (guess value was **p(2)**).

triplet – fitted values of exponential decay time(s) of fast photophysical process(es) (guess values have been **pd(end-expflag+1:end)**).

c – amplitudes of different components contributing to correlation: **c(1)** is constant offset; **c(2:length(pd)-expflag)** is(are) amplitude(s) of diffusion related part(s); **c(length(pd)-expflag+1:end)** is(are) amplitude(s) of photophysics related part(s).

velo – fitted values of velocity vector

err – fit error as defined as **sum((y-z)^2)**, where **y** are the data and **z** the fitted curve.

z – fitted curve.

Example:

Using the **y** and **t** as calculated from **FCSCrossRead** in the previous example, one can now fit the these data. First, one has to define the global variable **pd** (correct naming is important here!),

```
global pd
```

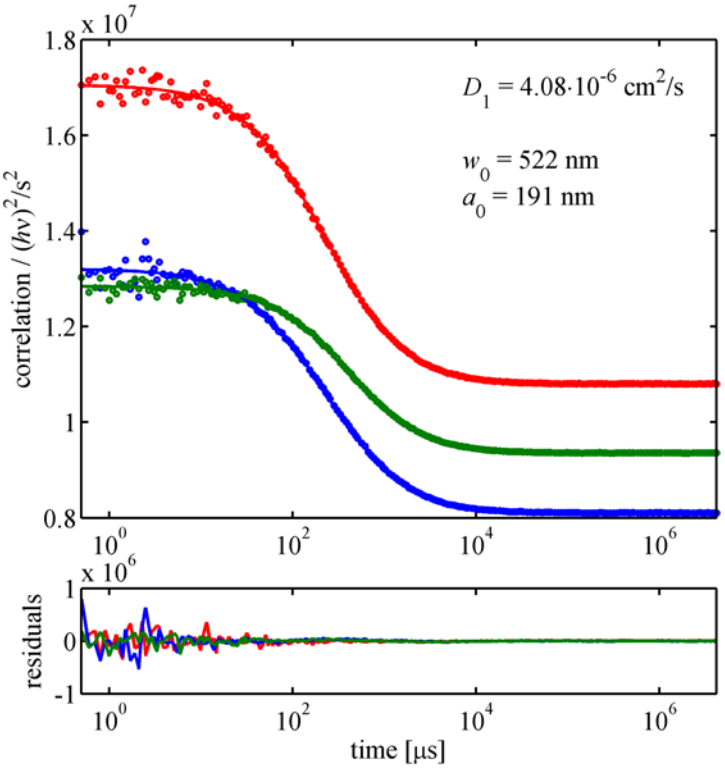
and to assign some initial guess value. Because for our example data, we assume to have only a single species and no photophysics, we may set

```
pd = 1e-8/1e-6;
```

assuming a diffusion coefficient close to 10^{-6} cm^2/s. In the simplest way, fitting is then done by

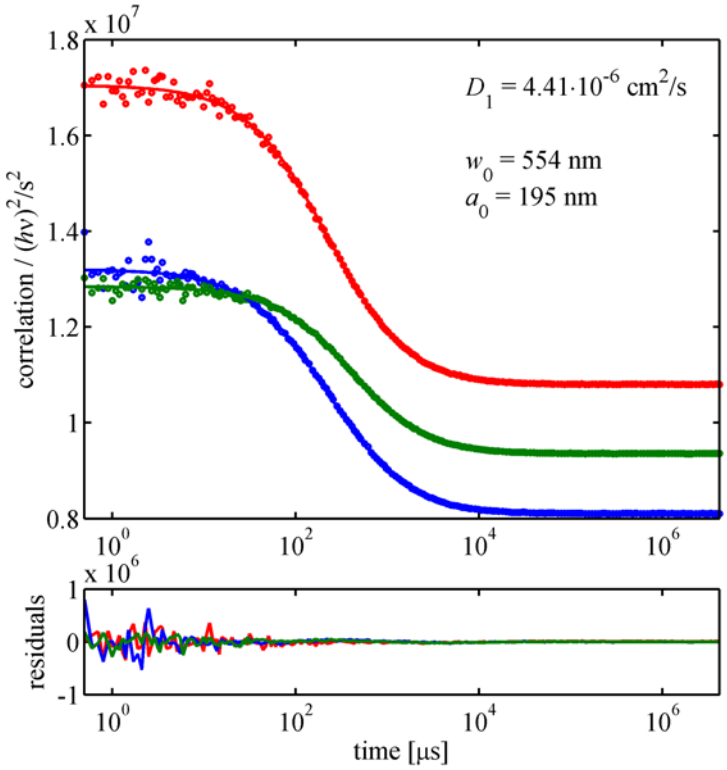
```
[dc w0 a0] = FCSFit(res);
```

This command starts the fitting and finally produces the following figure:



The fitted diffusion coefficient is returned in **dc** (in cm^2/s), the fitted beam waist w_0 and confocal parameter R_0 in **w0** and a**0**. Here, it was assumed that the setup parameters are equal to the default values. If, for example, the measurement had been done with a 150 µm diameter pinhole at 60 times magnification, 470 nm excitation and 530 nm centre emission wavelength, and an inter-focal distance of 430 nm, the function call would have to read

```
[dc w0 a0] = FCSFit(name, [], [], [], [75e3/60, [470 530]/1.33, 430]);
```


x 10^7
1.8
1.6
1.4
1.2
1
0.8
correlation / (hν)^2/s^2
D_1 = 4.41·10^-6 cm^2/s
w_0 = 554 nm
a_0 = 195 nm
10^0
10^2
10^4
10^6
x 10^6
1
0
-1
residuals
time [µs]