
FCSlib: User Guide

Release 1.3.0

November 1st, 2020

R. Pinto-Cámara, A. Linares, D.S. Moreno-Gutiérrez, H.O. Hernández, J.D. Martínez-Reyes, J.M. Rendón-Mancha, C.D. Wood & A. Guerrero

Correspondence: support.fcslib@mail.ibt.unam.mx

Contents

1. The Basics.....	1
1.1 Installation	1
1.1.1 Installing R and RStudio	1
1.1.2 Installing FCSlib using the RStudio GUI	2
1.1.2.1 Manual installation	3
1.2 Loading FCSlib into the R environment.....	3
1.3 Keeping FCSlib up to date	4
1.4 Sample data and documentation.....	4
1.6 Support.....	5
1.7 Community contributions	5
2. Data Input and Structure	6
2.1 Exporting your data into R	6
2.1.1 Supported data file formats.....	6
2.2 Data structure	7
2.2.1 Transforming multiple-image TIFF files for use with FCSlib.....	9
2.3 Data binning for better visualization	9
3. Data Detrending.....	14
3.1 Photobleaching correction.....	14
3.1.1 Boxcar feature filtering	17
3.2 Sample movement correction	19
3.2.1 Compensation value for trend correction (‘max’ parameter)	22
4. Model Fitting.....	24

4.1 Goodness-of-fit	28
4.2 Using a custom-built physical model	29
5. Fluorescence Correlation Spectroscopy	31
5.1 Single-Point FCS	31
5.1.1 Data exploration	31
5.1.2 Building the autocorrelation curves.....	34
5.1.2.1 Compensating for negative $G\tau$ values	36
5.1.3 Physical model fitting and coefficient extraction	37
5.1.3.1 Simplifying the correlation data.....	39
5.1.4 Calculating the dimensions of the Point Spread Function	41
5.1.5 Fluorescence Cross-Correlation Spectroscopy.....	42
5.1.6 Two-component Single-Point FCS.....	45
5.2 Scanning-FCS.....	48
5.2.1 Data exploration	48
5.2.2 Two-dimensional Autocorrelation Function	51
5.2.2.1 Simplifying the ACF carpet	51
5.2.2.2 ACF carpet analysis	53
5.2.3 Pair Correlation Function	55
5.2.3.1 Data exploration	55
5.2.3.2 Building the pCF carpet.....	58
5.2.3.3 Fixed-column pCF.....	61
6. Fluorescence Fluctuation Spectroscopy	62
6.1 Number and Brightness	62
6.1.1 Data exploration	64
6.1.2 Building the Number and Brightness graph.....	67
6.1.3 Moving Number & Brightness.....	68
6.2 Pair Correlation of Molecular Brightness.....	71
6.2.1 Building the pCOMB carpet.....	71
References	73

1. THE BASICS

FCSlib comprises a set of tools for fluorescence correlation and fluorescence fluctuation spectroscopy (FCS and FFS, respectively) data analysis performance. It is cost-free, open-source and can run on most systems.

- ✓ *In this section, you will learn how to get the latest version of FCSlib up and running on your computer, as well as where to find sample data, relevant documentation and how to reach out for support and contributions.*

1.1 Installation

FCSlib was built under R version 3.6.3. The latest versions of R and RStudio must be first installed.

1.1.1 Installing R and RStudio

Open your web browser and navigate to <https://cloud.r-project.org>. Then, select the right download link according to your OS (i.e. Linux, Mac OS or Windows) and follow the instructions.

1 CRAN Mirrors

The Comprehensive R Archive Network is available at the following URLs, please choose a location close to you. Some statistics on the status of the mirrors can be found here: [main page](#), [windows release](#), [windows old release](#).

If you want to host a new mirror at your institution, please have a look at the [CRAN Mirror HOWTO](#).

0-Cloud

<https://cloud.r-project.org/> Automatic redirection to servers worldwide, currently sponsored by Rstudio

2 The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

3 R for Windows

Subdirectories:

[base](#) Binaries for base distribution. This is what you want to [install R for the first time](#).

4 R-4.0.2 for Windows (32/64 bit)

[Download R 4.0.2 for Windows](#) (84 megabytes, 32/64 bit)

[Installation and other instructions](#)

[New features in this version](#)

5

Setup - R for Windows 4.0.2

Installing

Please wait while Setup installs R for Windows 4.0.2 on your computer.

Extracting files...

C:\Program Files\R\R-4.0.2\library\tools\help\tools.rdb

Cancel

Figure 1.1. Example of R installation for Windows OS.

Once R has been installed on your system, navigate to <https://rstudio.com/products/rstudio/download> and choose the latest RStudio installer.

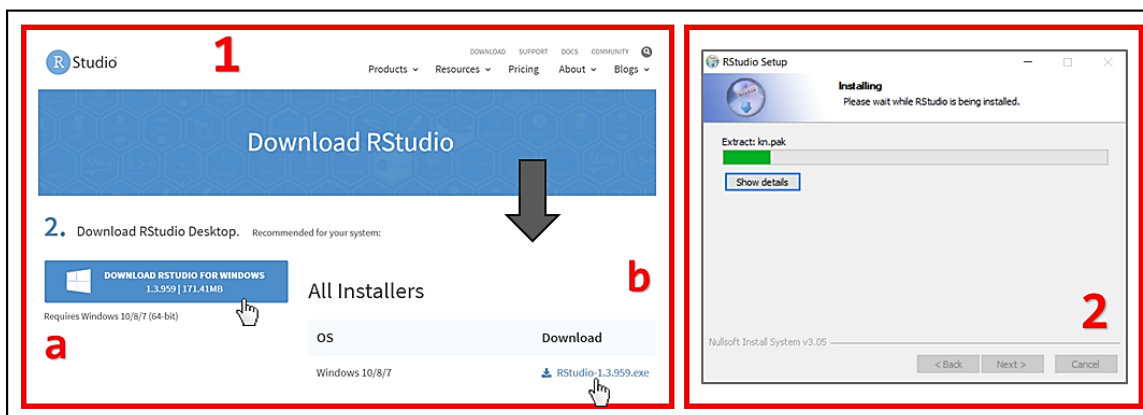


Figure 1.2. Example of RStudio installation for Windows OS.

1.1.2 Installing FCSlib using the RStudio GUI

FCSlib package can be installed directly from the Comprehensive R Archive Network (CRAN) server by using RStudio's useful graphical user interface (GUI). Once the latest versions of R and RStudio have been installed, open RStudio on your computer.

1. Head to the **Packages** tab located on the lower left panel on the screen.
2. Click on the **Install** button.
3. Type "FCSlib" on the pop-up window and select it.
4. Select **Install**. Make sure that the 'Install dependencies' option is checked.

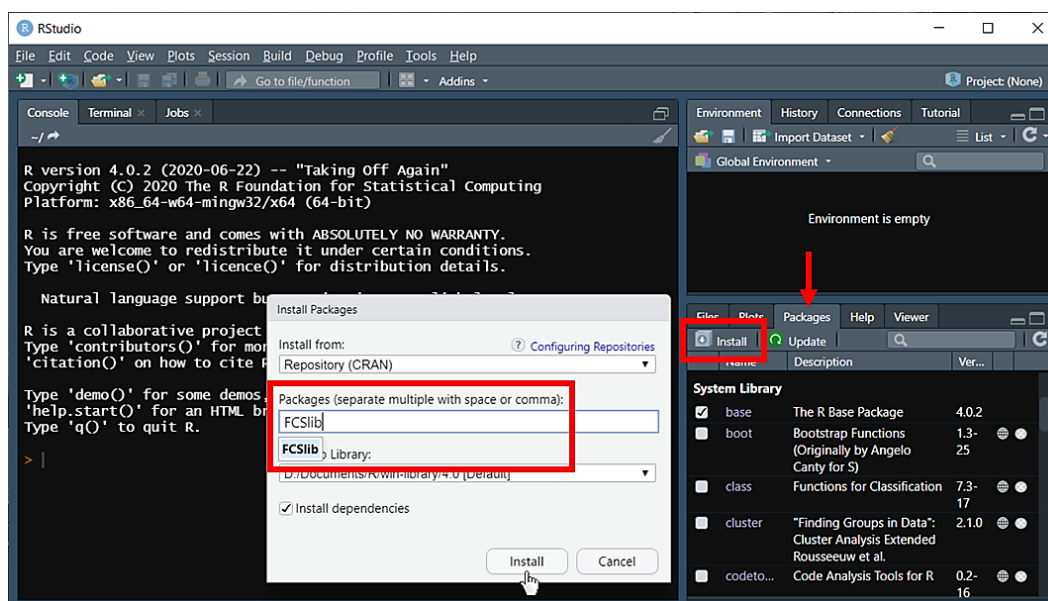


Figure 1.3. Direct installation of FCSlib via the CRAN server in RStudio.

1.1.2.1 Manual installation

You can also install the FCSlib package by downloading the compressed file from the web and manually selecting it in the RStudio GUI. On your web browser, navigate to <https://cran.r-project.org/web/packages/FCSlib/index.html>, click the file next to 'Package source' under the Downloads section and save it in the desired directory. Next, open RStudio.

1. Head to the **Packages** tab located on the lower left panel on the screen.
2. Click on the **Install** button.
3. Choose to install from a **Package Archive** and select the file you downloaded
4. Select **Install**.

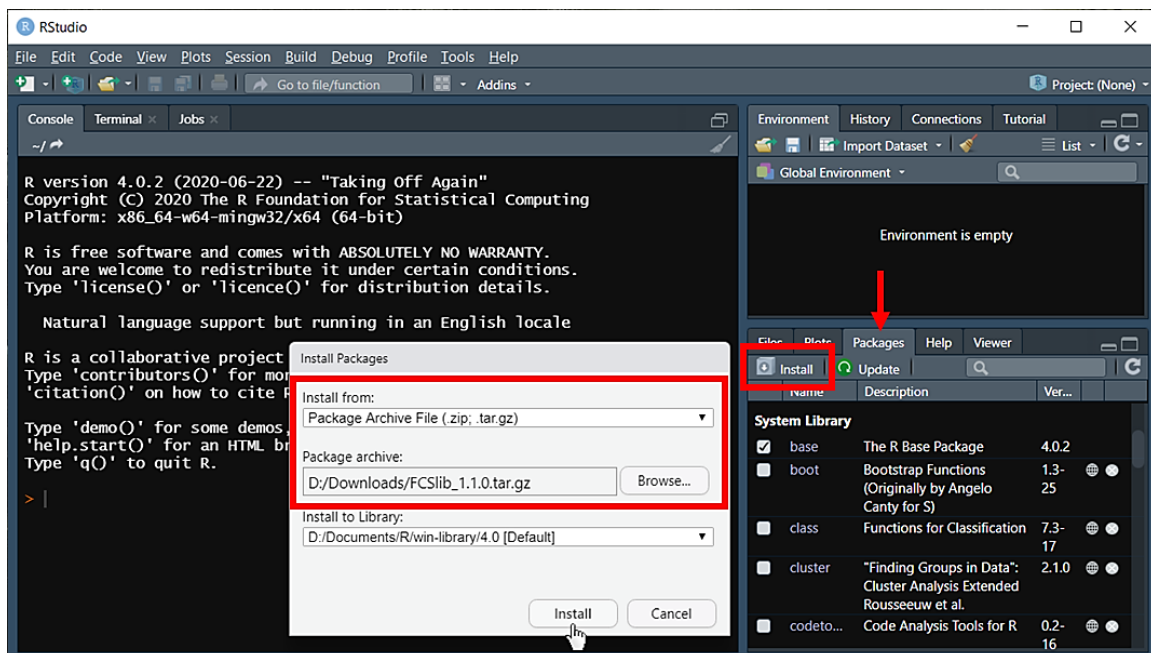


Figure 1.4. Manual installation of FCSlib via the package compressed file in RStudio.

You can also install FCSlib by accessing the RStudio console and typing:

```
install.packages("FCSlib")
```

1.2 Loading FCSlib into the R environment

Once installed, you will need to load the FCSlib library in order to access the functions provided. There's two possible ways for doing this, a) checking the box next to the package name under the 'Packages' tab (see figures 1.3 and 1.4) or b) by typing:

```
library(FCSlib)
```

1.3 Keeping FCSlib up to date

FCSlib is intended to be a constantly evolving tool. Updates of this package are likely to bring new features, bug fixes and community-influenced add-ons.

In order to keep FCSlib up to date, all you need to do is to repeat the steps in [section 1.1.2](#), as new versions of this software will always be available at the CRAN server.

You can always check if there are any updates available for your currently installed packages by typing:

```
old.packages()
```

Also, you can automatically update all your packages by typing:

```
update.packages()
```

Be aware that updating packages can sometimes alter how both the package and your code work. Be cautious when updating as some functionality may change or disappear.

1.4 Sample data and documentation

Sample data for testing the FCSlib package can be found at <https://github.com/FCSlib/FCSlib> inside the ‘Sample Data’ folder, where a set of simulations and real experimental data are readily available for download. Once there, right-click on the file you wish to download and select ‘Save link as...’ to save the file to your drive.

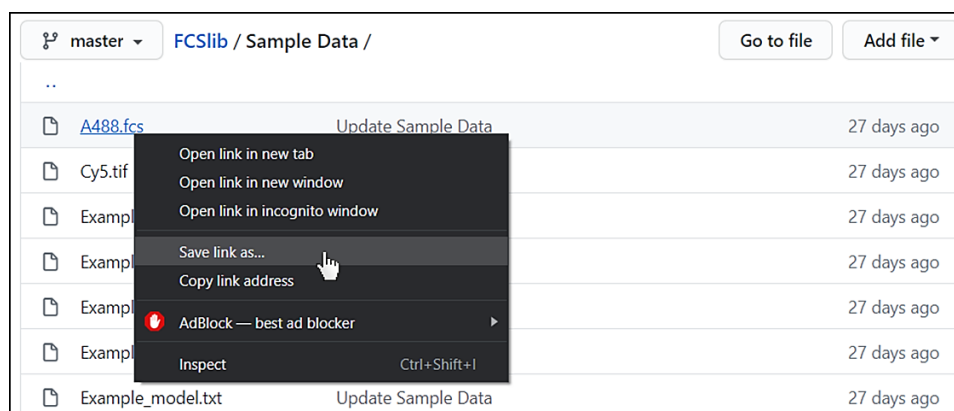


Figure 1.5. Downloading Sample Data from the GitHub repository.

For a manual that contains a detailed description of each of the functions provided with FCSlib, navigate to <https://cran.r-project.org/web/packages/FCSlib/FCSlib.pdf>. This document is automatically generated by CRAN when uploading a new package to the server. It contains information about how each function was built, how they work and useful examples to put them to the test. You can also access this information by clicking on the package name under the ‘Packages’ tab (see figures 1.3 and 1.4).

Other relevant documentation such as the main article where this software is presented, its supplementary material, FCSlib version notes and this user guide can be found at <https://github.com/FCSlib/FCSlib> inside the ‘Documentation’ folder.

1.6 Support

If you find a bug or run into an error, feel free to reach out for support at <https://github.com/FCSlib/FCSlib/issues>. Before creating a new issue, please verify that your query has not been reported before. Also, make sure that you are using the latest version of the package, as some problems may have already been addressed in the up-to-date version.

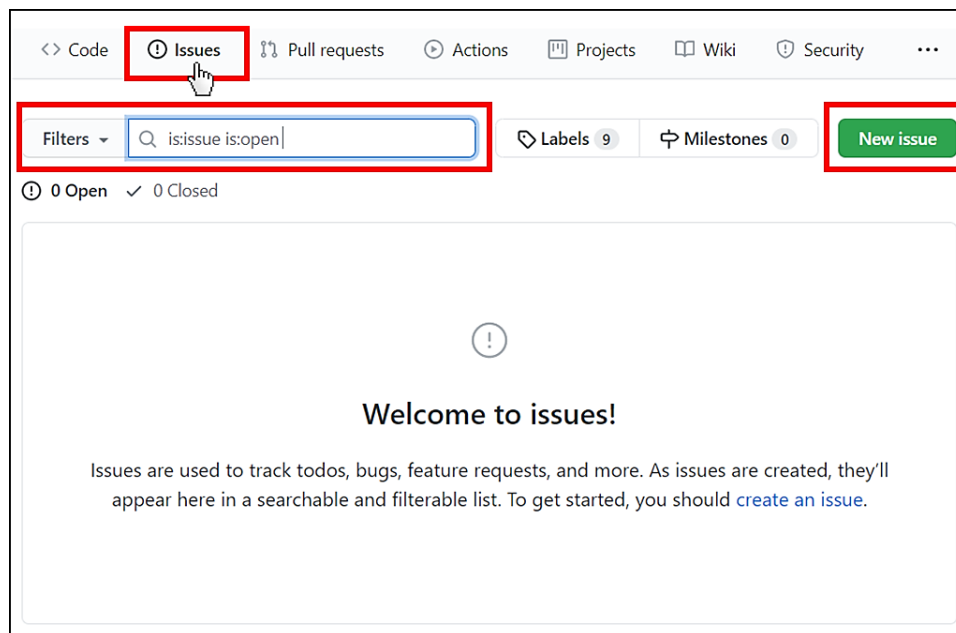


Figure 1.6. Searching and creating package-related issues at the GitHub repository.

1.7 Community contributions

Contributions from the community are encouraged and always welcomed. If you want to develop new features for FCSlib and would like to see them added to the package, please contact the authors by sending an email to: support.fcslib@mail.ibt.unam.mx.

2. DATA INPUT AND STRUCTURE

The current version of FCSlib works with data acquired with confocal-based systems and allows to analyze two types of experimental setups: single-point scanning experiments and line scanning experiments.

2.1 Exporting your data into R

Make sure that the working directory of your current RStudio session matches the location of the files you want to work with. To change your working directory, navigate to the 'Session' tab, head down to 'Set Working Directory' and specify the location in your computer where your files are at.

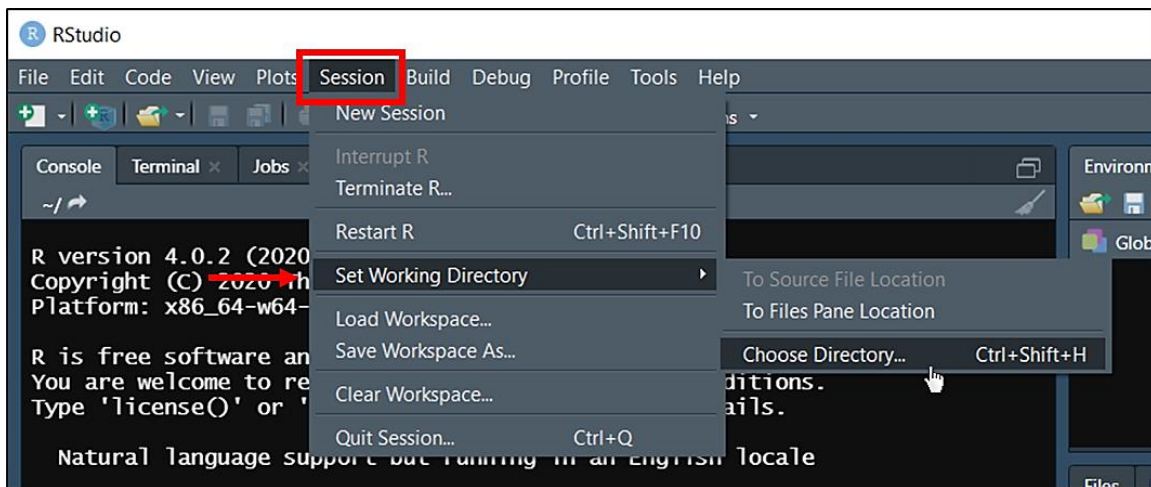


Figure 2.1. Changing the working directory in RStudio.

You can also specify the working directory by typing:

```
setwd("D:/Documents/Example")
```

2.1.1 Supported data file formats

Currently, FCSlib supports the input of **TIF**, **FCS**, **SPC** and **DAT** file formats. In order to read these files and store them as objects in the R environment, type:

```
example_data <- readfileTiff(file = "Example_file_name.tif") # TIFF files
example_data <- readfileFcs(file = "Example_file_name.fcs") # FCS files
example_data <- readfileSpc(file = "Example_file_name.spc") # SPC files
example_data <- read.table(file = "Example_file_name.dat") # DAT files
```

Note that **DAT** and other simple text-based and chart-like file formats can be read and imported directly using the `read.table()` function, which belongs to R base functions. When using `read.table()` to import this kind of files, the generated object is a `data frame`.

2.2 Data structure

Data in FCSlib are managed as vectors and array-like objects. When importing single-image TIFF files, data is stored as matrices and further turned into vectors for processing and analysis. DAT and other simple chart-like text file formats are imported as data frames and further converted into vectors or matrices.

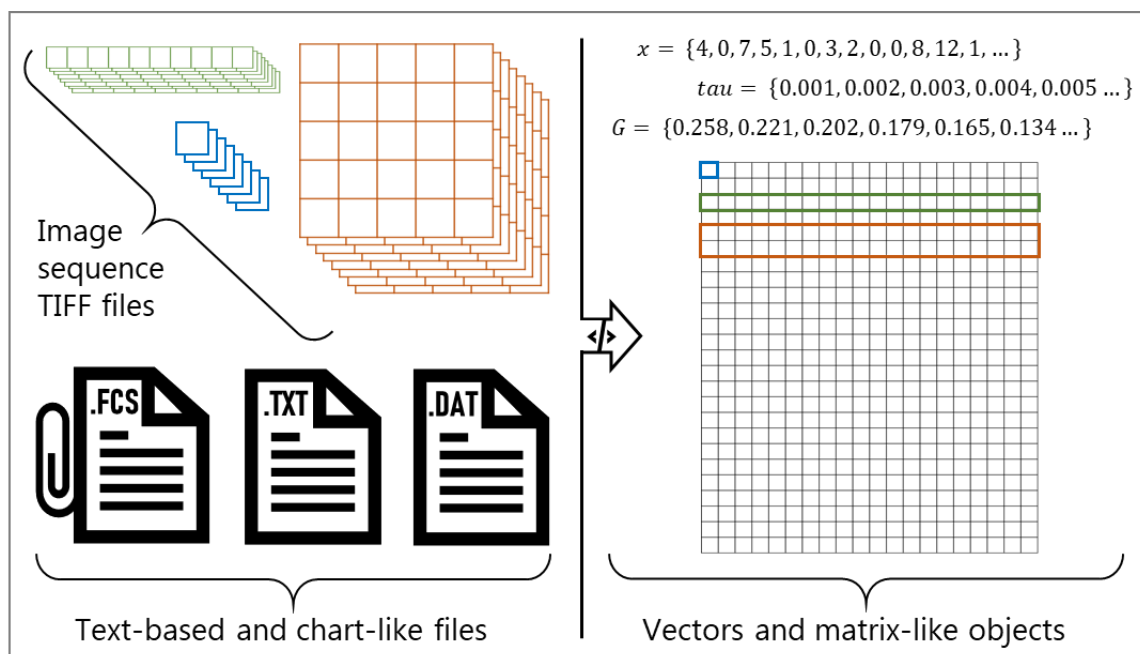


Figure 2.2. Data rearrangement for use with FCSlib.

When importing a TIFF image file, the generated object should look like this:

```
example_data <- readFileTiff(file = "Example_file_name.tif")
class(example_data); dim(example_data)

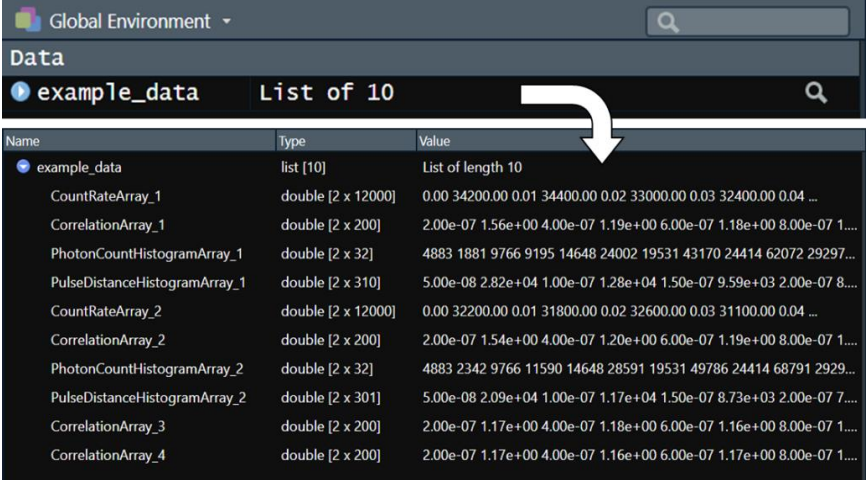
## [1] "array"
## [1] 128 128 200
```

Where "Example_file_name.tif" is a sequence of 200 images of 128x128 pixels and is imported directly as an array.

FCS files can store additional information such as previously calculated correlation curves between multiple channels, photon counting histograms (PCH), pulse distance histograms (PDH), etc. When read, each element in these files is imported as vectors inside a `list` in R, which can be then accessed directly by the functions provided. Now, for the case of imported FCS files, the generated object should look like this:

```
example_data <- readFileFcs(file = "Example_file_name.fcs")
class(example_data); length(example_data)

## [1] "list"
## [1] 10
```



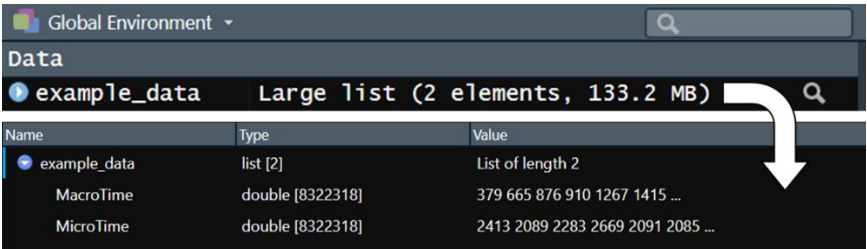
Name	Type	Value
example_data	list [10]	List of length 10
CountRateArray_1	double [2 x 12000]	0.00 34200.00 0.01 34400.00 0.02 33000.00 0.03 32400.00 0.04 ...
CorrelationArray_1	double [2 x 200]	2.00e-07 1.56e+00 4.00e-07 1.19e+00 6.00e-07 1.18e+00 8.00e-07 1....
PhotonCountHistogramArray_1	double [2 x 32]	4883 1881 9766 9195 14648 24002 19531 43170 24414 62072 29297...
PulseDistanceHistogramArray_1	double [2 x 310]	5.00e-08 2.82e+04 1.00e-07 1.28e+04 1.50e-07 9.59e+03 2.00e-07 8....
CountRateArray_2	double [2 x 12000]	0.00 32200.00 0.01 31800.00 0.02 32600.00 0.03 31100.00 0.04 ...
CorrelationArray_2	double [2 x 200]	2.00e-07 1.54e+00 4.00e-07 1.20e+00 6.00e-07 1.19e+00 8.00e-07 1....
PhotonCountHistogramArray_2	double [2 x 32]	4883 2342 9766 11590 14648 28591 19531 49786 24414 68791 2929...
PulseDistanceHistogramArray_2	double [2 x 301]	5.00e-08 2.09e+04 1.00e-07 1.17e+04 1.50e-07 8.73e+03 2.00e-07 7....
CorrelationArray_3	double [2 x 200]	2.00e-07 1.17e+00 4.00e-07 1.18e+00 6.00e-07 1.16e+00 8.00e-07 1....
CorrelationArray_4	double [2 x 200]	2.00e-07 1.17e+00 4.00e-07 1.16e+00 6.00e-07 1.17e+00 8.00e-07 1....

Figure 2.3. Example of how an imported FCS data file looks like in the R environment.

Similar to the FCS data file format, SPC files can store multiple data sets within them, which correspond to the *macrotime* and *microtime* data (see figure 2.4). When created, SPC data undergoes an encryption process that makes reading these files a much more difficult task. The current implementation of the `readFileSpc()` function makes this a process that can take anywhere between 1 and 10 minutes before completion, depending on the system's hardware. Optimization of this function is likely to come in a future update.

```
example_data <- readFileSpc(file = "Example_file_name.spc")
class(example_data); length(example_data)

## [1] "list"
## [1] 2
```



Name	Type	Value
example_data	list [2]	List of length 2
MacroTime	double [8322318]	379 665 876 910 1267 1415 ...
MicroTime	double [8322318]	2413 2089 2283 2669 2091 2085 ...

Figure 2.4. Example of how an imported SPC data file looks like in the R environment.

2.2.1 Transforming multiple-image TIFF files for use with FCSlib

For the case of sequences of images of $n \times n$ dimensions where, for example, each pixel or line of pixels represents a point or a line of single-point and line scanning experiments, respectively, one can use the `tiff_to_mtx()` function to rearrange the information contained in multiple-page files into a single matrix in order to be further processed and analyzed.

Using the `readFileTiff()` function will import the file as a three-dimensional array, regardless of whether there are one or more images. To directly transform any TIFF image sequence into a two-dimensional matrix, type:

```
example_data <- readFileTiff(file = "Example_file_name.tif")
class(example_data); dim(example_data)

## [1] "array"
## [1] 128 128 200

# Transforming into a 2D matrix
example_data_mtx <- tiff_to_mtx(data = example_data, columns = 64)
class(example_data_mtx); dim(example_data_mtx)

## [1] "matrix"
## [1] 64 51200
```

In this example, the argument `columns` defines the number of columns of the resulting matrix. The number of rows is thus calculated as the total number of pixels in the TIFF file divided by the user-specified number of columns. If the total amount of pixels in the original file exceeds the product of the columns by the rows, then only those data necessary to fill up the matrix will be considered. This function also works with already-existing arrays in the R environment.

Note that the total amount of pixels in this last example is $128 \times 128 \times 200 = 3,276,800$. When divided by 64 (number of columns), this is equal to 51,200, which is the number of rows of the resulting matrix, showed above.

2.3 Data binning for better visualization

Fluorescence Correlation Spectroscopy (FCS) experiments usually require the acquisition of a great amount of data, which ranges between several thousands to millions of points.

Trying to graph this amount of information all at once can result in very long computation times and demands high-end hardware that supports this task. Furthermore, these experiments are performed at very high speeds (from milliseconds to even nanoseconds); looking at this data directly on its raw form provides no information as the level of detail is very high. In other words, ‘zooming out’ is necessary in order to be able to observe the existence of global trends possibly affecting the data, such as fluorophore photobleaching, sample movement and other experimental disturbances (e.g. microscope movement or contamination from external light sources).

The `binTimeSeries()` and `binMatrix()` functions allow to group the raw data of single-point and line-scan experiments, respectively, into several intervals of user-specified length. This useful tool significantly reduces the number of data points (or lines) and allows to efficiently graph the result.

When binning the data, one can choose between representing them as the average or as the sum of all the points (or lines) in each interval. This choice does not change the binning result at all but will impact in further quantitative analysis.

As an example for the use of the `binTimeSeries()` function, a simulation of freely diffusing particles with exponential photobleaching will be used. Let us visualize the first and the last 100,000 points. Type:

```
# Importing the data
data <- read.table("PB030.dat", header = F)
dim(data)

## [1] 3276800      3

# Visualizing the first 100000 points and the last 100000 points
acqTime <- 4e-6
tau <- (1:100000)*acqTime
plot(data$V2[1:100000]~tau, type = "l", xlab = "Time (s)", ylab = "Photon counts",
      main = "Raw single-point data (first 100000 points)")
plot(data$V2[3176801:3276800]~tau, type = "l", xlab = "Time (s)", ylab = "Photon counts",
      main = "Raw single-point data (last 100000 points)")
```

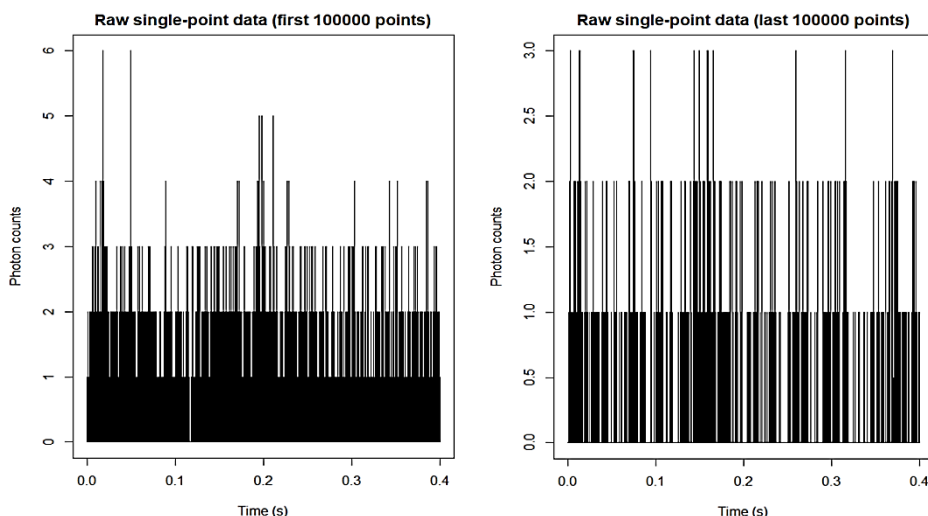


Figure 2.5. Raw single-point data of a simulation of freely diffusing particles with photobleaching.

A two-fold photon count decrease can be observed when looking at the initial and last segments of the time series (figure 2.5). However, this information is not enough to confirm nor deny the presence of a trend in the data.

Let us now bin the data for better visualization:

```
# Binning the data
par(mfrow = c(2,2))
b.data1 <- binTimeSeries(data$V2, acqTime = acqTime, nIntervals = 10000)
b.data2 <- binTimeSeries(data$V2, acqTime = acqTime, nIntervals = 1000)
b.data3 <- binTimeSeries(data$V2, acqTime = acqTime, nIntervals = 100)
b.data4 <- binTimeSeries(data$V2, acqTime = acqTime, nIntervals = 10)
```

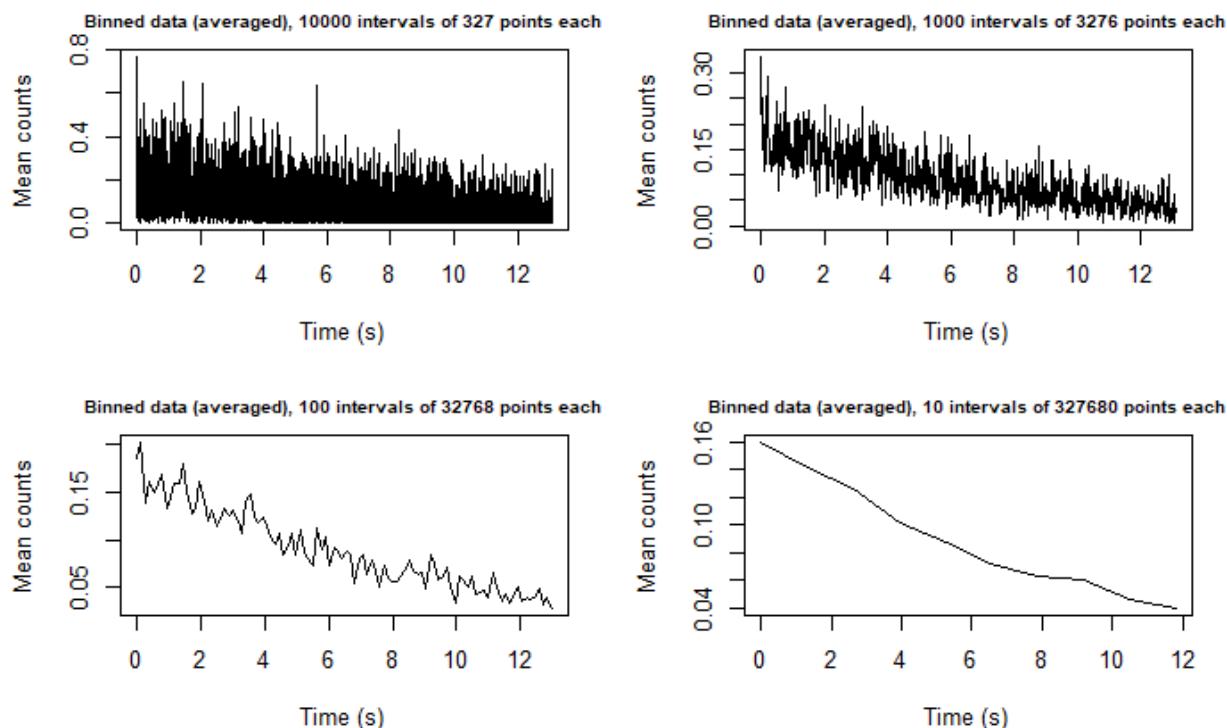


Figure 2.6. Binned single-point data of a simulation of freely diffusing particles with photobleaching.

Each of the graphs in figure 2.6 corresponds to the whole data series, with different binning degrees. Once binned, graphing the original data becomes a much faster and computationally cheaper task. The user-defined number of intervals `nIntervals` will establish the level of temporal detail and should be adjusted based on the user's specific needs. Unlike figure 2.5, a progressive decrease in average photon counts due to photobleaching is much more evident in figure 2.6.

Now, to test the `binMatrix()` function, an orbital-scanning experiment, where the fluorescence intensity data was collected at the focal adhesion of cells expressing a paxillin-EGFP fusion, will be used. 64 adjacent points separated 44 nm of each other were circularly scanned with an orbit period of 1 ms. Type:

```
# Importing the data
data <- read.table("Pax.dat", header = F)
dim(data)
## [1] 19660800      1
```

"Pax.dat" is also a table with one column and several million points. In this case, each point is part of a circularly scanned line of 64 pixels long; this experiment contains spatial and temporal information. In order to visualize this data, it must be first rearranged into a matrix. Type:

```
# Rearranging into a matrix
data.m <- matrix(data$V1, nrow = 64)
dim(data.m)
## [1]      64 307200
```

Now, `data.m` is a matrix of 64 columns (pixels) and 307200 rows (scan lines). To visualize the first and the last 10000 lines, type:

```
# Visualizing the first 10000 Lines and the Last 10000 Lines
lineTime <- 1e-3
tau <- (1:10000)*lineTime
r <- 1:64
image.plot(x = r, y = tau, z = data.m[,1:10000], xlab = "Pixel", ylab = "Time (s)",
           main = "Raw line-scan data (first 10000 points)")
image.plot(x = r, y = tau, z = data.m[,297201:307200], xlab = "Pixel", ylab = "Time (s)",
           main = "Raw line-scan data (last 10000 points)")
```

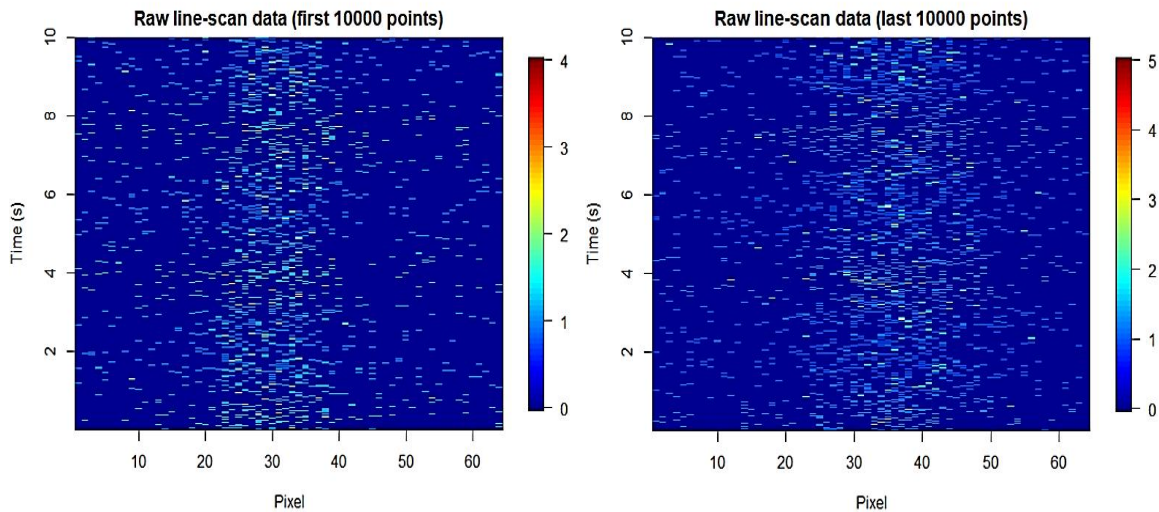


Figure 2.7. Raw line-scan data of an orbital scanning experiment.

The `binMatrix()` function can receive either a vector or a matrix. Now, for the binning process type:

```
# Binning the data
par(mfrow = c(2,2))
b.data.m1 <- binMatrix(data$V1, lineTime = lineTime, nIntervals = 10000, columns = 64)
b.data.m2 <- binMatrix(data$V1, lineTime = lineTime, nIntervals = 1000, columns = 64)
b.data.m3 <- binMatrix(data$V1, lineTime = lineTime, nIntervals = 100, columns = 64)
b.data.m4 <- binMatrix(data$V1, lineTime = lineTime, nIntervals = 10, columns = 64)
```

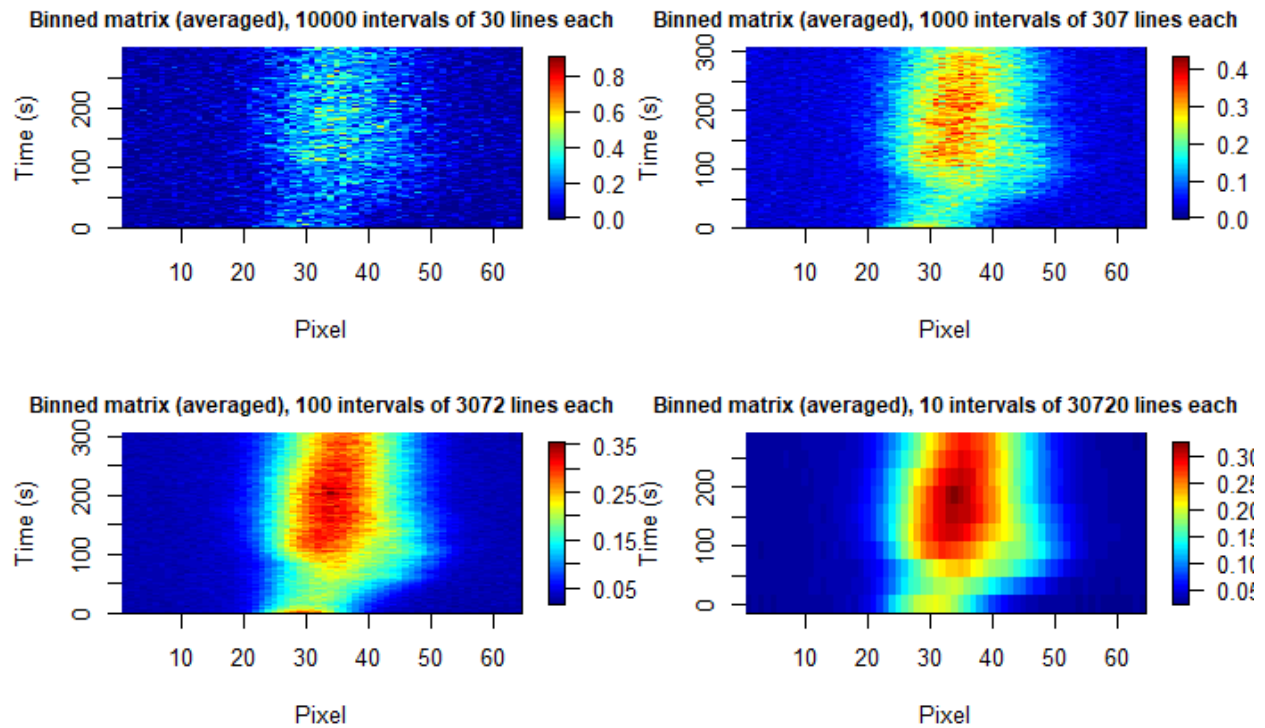


Figure 2.8. Binned line-scan data of an orbital scanning experiment.

The results in figure 2.7 represent only a small portion of the experiment. Fully plotting the data would take a severe amount of time on most systems and is a rather impractical task. Also, the raw graph alone would not provide any useful information because of the high temporal resolution. In figure 2.8, however, once the data has been binned, a much better idea of its structure can be obtained. With this knowledge, the user can now decide which detrending procedure (if any is necessary) will benefit their research the most, based on their goals.

3. DATA DETRENDING

When performing FCS and FFS data analysis, it is commonly assumed that the concentration of fluorophores in the sample is constant throughout the entire experiment. Thus, one might expect the mean intensity to remain constant all along the length of the data. In the practice, this isn't always true, especially when dealing with imaging of live specimens where most fluorescently labelled proteins are susceptible to quenching due to photobleaching. Moreover, factors such as cell or sample movement or even instrumentation-driven signal fluctuations can also introduce undesired bias into the analysis.

In order to avoid the effect these phenomena might cause to the analysis and interpretation of results, a data detrending routine should be first carried out before processing.

✓ *In this section, a review of the available data detrending algorithms in FCSlib is provided.*

3.1 Photobleaching correction

Most bleaching correction algorithms are based on either fitting a mathematical model that best describes or approximates the bleaching phenomenon or simply smoothing the data. Either way, the common goal of these methods is to eliminate global trends (e.g. photobleaching or cell movement) while preserving local trends, which are assumed to be related only to the dynamic process in study.

Along with the `detrendTimeSeries()` function are provided three different data detrending algorithms:

The **exponential detrending** algorithm fits an exponential function to a binned version of the data (see [section 2.3](#)), of the form:

$$y = A * e^{(k*t)}$$

where k is the decay constant and t is the time. Once fitted, the calculated exponential function is then evaluated using the whole data series, from which the evaluated data is subtracted to obtain the residuals. At this point, the coefficient A is finally added to the residuals to compensate for the photobleaching effect.

Similar to exponential detrending, **polynomial detrending** fits a polynomial function of user-specified degree (function parameter `degree`) to the data. The fitting residuals are then calculated and added a compensating factor obtained from the calculated polynomial function.

Boxcar filter detrending applies a moving average filter with a user-specified temporal window size (function parameter `w`) over the original data series. The moving average vector is then subtracted from the raw data to obtain the residuals. Finally, a compensation factor is added to the whole series for trend correction.

Even though adding a raw value to the residuals (based on each detrending algorithm) for trend correction is mathematically correct, it is not physically correct if one considers the statistical nature of photon detection (i.e. a discrete phenomenon, not continuous), which can be well modeled by Poisson distributions [1].

The `pois` parameter of the `detrendTimeSeries()` function allows to compensate for the photobleaching effect by adding random, uncorrelated numbers from a Poisson distribution. In other words, this approximation adds discrete, random photon counts in a quantity for the mean signal intensity value to remain constant throughout the whole data series; though, at the cost of adding some noise due to the random component of the sampling algorithm.

The `max` parameter allows to choose to perform data detrending based on either the highest value of the original data series (`max = TRUE`) or the very first one. Said value will provide the means for trend correction as the whole series will be adjusted to match it.

To use these detrending tools, type:

```
# Importing the data
data <- read.table("PB030.dat")
```

"PB030.dat" corresponds to a simulation of freely diffusing particles with exponential photobleaching. For this experiment, simulated points were collected at a rate of 4 μ s.

```
# Detrending the data
exp <- detrendTimeSeries(data$V2, acqTime = 4e-6, nIntervals = 100, algorithm = "exp")
bc <- detrendTimeSeries(data$V2, acqTime = 4e-6, nIntervals = 100, algorithm = "boxcar", w = 1e5)
p3 <- detrendTimeSeries(data$V2, acqTime = 4e-6, nIntervals = 100, algorithm = "poly", degree = 3)
p10 <- detrendTimeSeries(data$V2, acqTime = 4e-6, nIntervals = 100, algorithm = "poly", degree = 10)
```

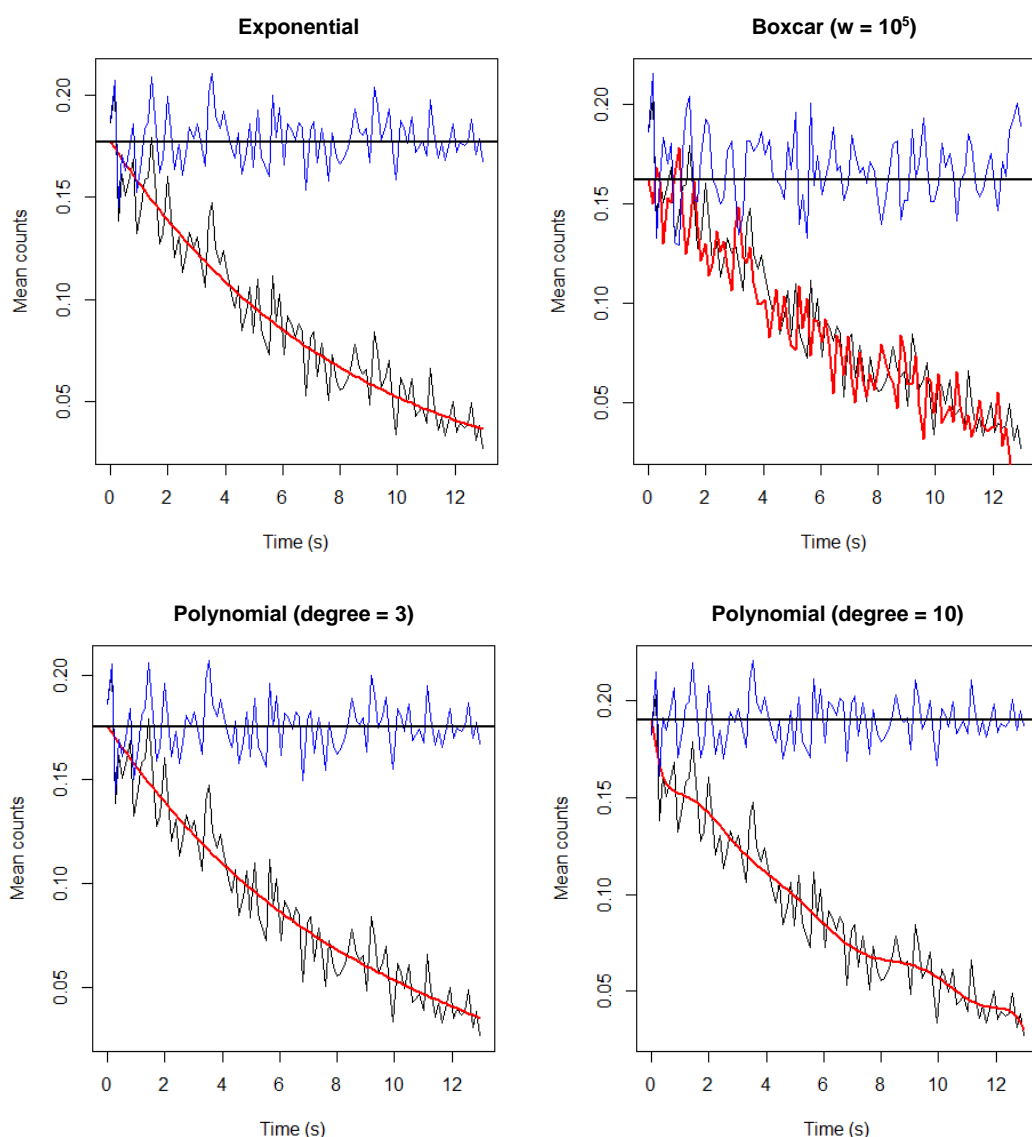


Figure 3.1. Comparison of three detrending algorithms for single-point data. In each graph, the black curve corresponds to the raw, undetrended data; the red curve corresponds to the fitted (exponential, polynomial) or moving average (boxcar) vector; the blue curve corresponds to the detrended data; and the horizontal black line indicates the value to which all the data is adjusted for trend correction (modified by the `max` parameter).

The `detrendTimeSeries()` function automatically plots the result of the detrending process (when parameter `plot = TRUE`, default) with a convenient representation that reflects each stage of the algorithm (figure 3.1). The `plot` parameter allows to decide to whether or not plot the result. Setting `plot = FALSE` will just save the result to the R environment.

For further graph customization, type:

```
# Detrending the data (automatic plot disabled)
exp <- detrendTimeSeries(data$V2, acqTime = 4e-6, nIntervals = 100, algorithm = "exp", plot = FALSE)

# Graphing the result (binned)
exp.b <- binTimeSeries(exp, acqTime = 4e-6, nIntervals = 100, plot = FALSE)
plot(exp.b$Counts~exp.b$Counts, ylab = "Custom 'Y' axis label", xlab = "Custom 'X' axis label",
     main = "Custom graph title", lwd = 2, type = "l", col = "green4", ylim = c(0,0.3))
legend("topright", legend = "detrended data", col = "green4", lwd = 2)

# Graphing the result (raw, first 10000 points)
plot(exp[1:10000]~data$V1[1:10000], ylab = "Custom 'Y' axis label",
     xlab = "Custom 'X' axis label", main = "Custom graph title", type = "l", col = "green4")
legend("topright", legend = "detrended data", col = "green4", lwd = 2)
```

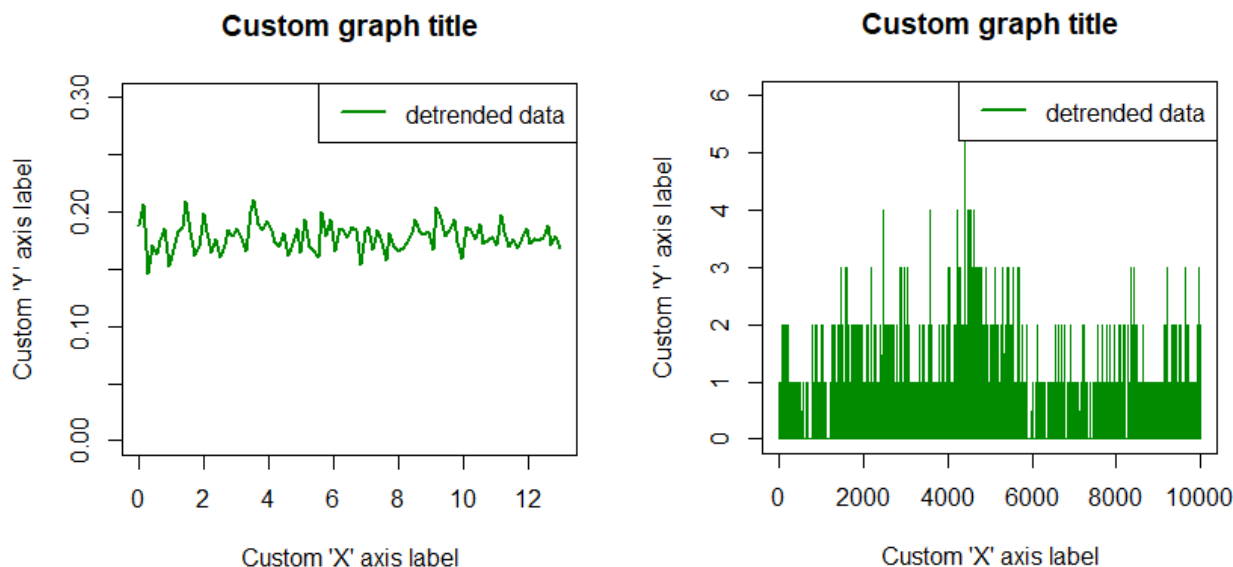


Figure 3.2. Detrended data graphs with custom graphical parameters. For comparison, the green curve in the left panel corresponds to the blue curve in the upper-left panel in figure 3.1.

Photobleaching happens and is only noticeable after several seconds of the experiment (as depicted by figure 3.1), which is orders of magnitude beyond the temporality of the diffusive process (10^{-4} to 10^{-3} seconds). Unless the temporality of the dynamic process of interest matches the temporality of the photobleaching phenomenon, a detrending routine for single-point data is not indispensable for quantifying fast dynamics (see figure S5 of supporting information of [2]). However, a detrending routine that allows to correct the decrease in fluorophore concentration due to quenching might be necessary in order to properly quantify $G0$ and, therefore, calculate the actual concentration of the fluorophores. For more information regarding this topic please refer to [section 5.1.3](#) of this document.

3.1.1 Boxcar feature filtering

When using the boxcar filter detrending algorithm, much care must be taken when determining the right window size. Since a boxcar filter averages out the information in segments of a given size, the dynamics associated with the length of these segments tends to be eliminated from the resulting data. The user has to make sure that the length of the boxcar filter window does not compromise the integrity of the autocorrelation information (for FCS analysis, see [section 5](#)).

To visualize the feature filtering effect, type:

```
# Importing the data
data <- read.table("PB030.dat")

# Detrending the data
bc1 <- detrendTimeSeries(data$V2, 4e-6, 100, algorithm = "boxcar", plot = F, w = 1e1)
bc2 <- detrendTimeSeries(data$V2, 4e-6, 100, algorithm = "boxcar", plot = F, w = 1e2)
bc3 <- detrendTimeSeries(data$V2, 4e-6, 100, algorithm = "boxcar", plot = F, w = 1e3)
bc4 <- detrendTimeSeries(data$V2, 4e-6, 100, algorithm = "boxcar", plot = F, w = 1e4)
bc5 <- detrendTimeSeries(data$V2, 4e-6, 100, algorithm = "boxcar", plot = F, w = 1e5)

# Calculating the autocorrelation curves
acqTime <- 4e-6
nPoints <- 2e5
tau <- (1:nPoints)*acqTime
raw <- fcs(data$V2, nPoints = nPoints)
g1 <- fcs(bc1, nPoints = nPoints)
g2 <- fcs(bc2, nPoints = nPoints)
g3 <- fcs(bc3, nPoints = nPoints)
g4 <- fcs(bc4, nPoints = nPoints)
g5 <- fcs(bc5, nPoints = nPoints)

# Graphing the results
plot(raw~tau, type = "l", log = "x",
      xlab = expression(tau(s)), ylab = expression(G(tau)))
plot(g1~tau, type = "l", log = "x", ylim = c(-0.5,1),
      xlab = expression(tau(s)), ylab = expression(G(tau)))
plot(g2~tau, type = "l", log = "x", ylim = c(-0.5,1),
      xlab = expression(tau(s)), ylab = expression(G(tau)))
plot(g3~tau, type = "l", log = "x", ylim = c(-0.5,1),
      xlab = expression(tau(s)), ylab = expression(G(tau)))
plot(g4~tau, type = "l", log = "x", ylim = c(-0.5,1),
      xlab = expression(tau(s)), ylab = expression(G(tau)))
plot(g5~tau, type = "l", log = "x", ylim = c(-0.5,1),
      xlab = expression(tau(s)), ylab = expression(G(tau)))
```

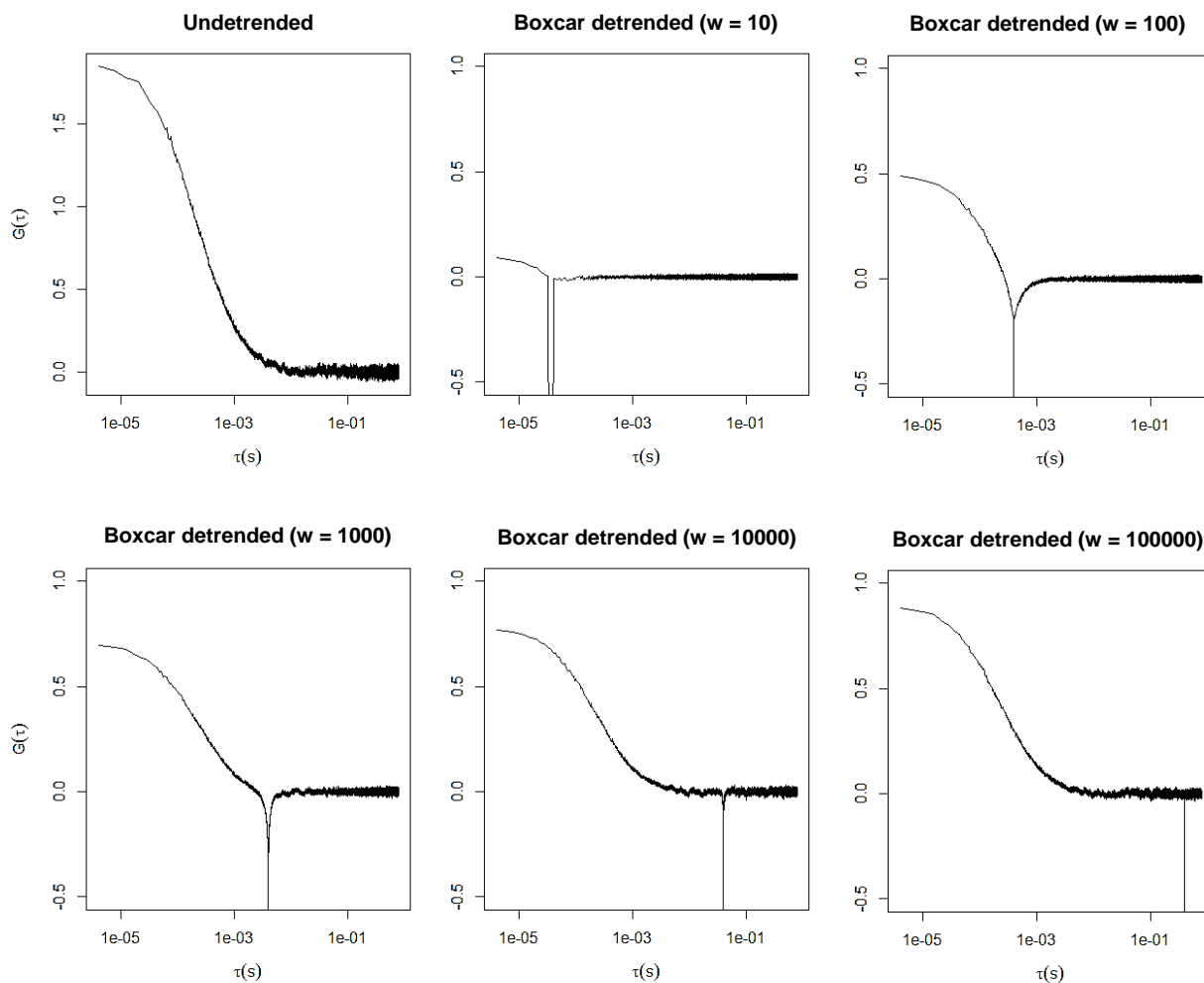


Figure 3.3. Autocorrelation curves of the “PB030.dat” time series data.

In the example above, the acquisition rate for the “PB030.dat” data is 4 μ s, which forces the boxcar window size to be large enough to preserve short-term dynamics information. In this case, in order to be able to retrieve information about the diffusive process of the particles after a boxcar filter detrending routine, a window size of size between 10000 and 100000 should be picked. Notice that smaller window sizes cause the correlation information to disappear, which makes it impossible to study molecular dynamics at this time ranges through FCS analysis.

3.2 Sample movement correction

When correcting for photobleaching, exponential detrending is commonly advised due to the nature of this phenomenon, which depends only on time and a decay constant and is of very similar shape as an exponential function. However, sample movement and other external disturbances do not follow a specific trend and are rather of random shape, which cannot be approximated by an exponential function. In this case, polynomial detrending, with a degree high enough to better approximate the data, is advised.

As an example, a line-scanning experiment of Paxilin-EGFP (see [section 2.3](#)) will be used. Type:

```
# Importing the data
data <- read.table("Pax.dat", header = F)

# Rearranging into a matrix
data.m <- matrix(data$V1, nrow = 64)

# Binning the raw data
b.m <- binMatrix(data$V1, lineTime = 1e-3, nIntervals = 100, columns = 64)

# Detrending the data (Exponential)
det_data.e <- matrix(0, nrow = dim(data.m)[1], ncol = dim(data.m)[2])
for (i in 1:64){
  det_data.e[i,] <- detrendTimeSeries(data.m[i,], 1e-3, 100, algorithm = "exp", plot = F)
}
det.binned <- binMatrix(det_data.e, 1e-3, 500, 64)

# Detrending the data (Polynomial)
det_data.p <- matrix(0, nrow = dim(data.m)[1], ncol = dim(data.m)[2])
for (i in 1:64){
  det_data.p[i,] <- detrendTimeSeries(data.m[i,], 1e-3, 100, algorithm = "poly",
                                     degree = 10, plot = F)
}
det.binned <- binMatrix(det_data.p, 1e-3, 500, 64)

# Detrending the data (Boxcar)
det_data.b <- matrix(0, nrow = dim(data.m)[1], ncol = dim(data.m)[2])
for (i in 1:64){
  det_data.b[i,] <- detrendTimeSeries(data.m[i,], 1e-3, 100, algorithm = "boxcar",
                                     w = 1000, plot = F)
}
det.binned <- binMatrix(det_data.b, 1e-3, 500, 64)
```

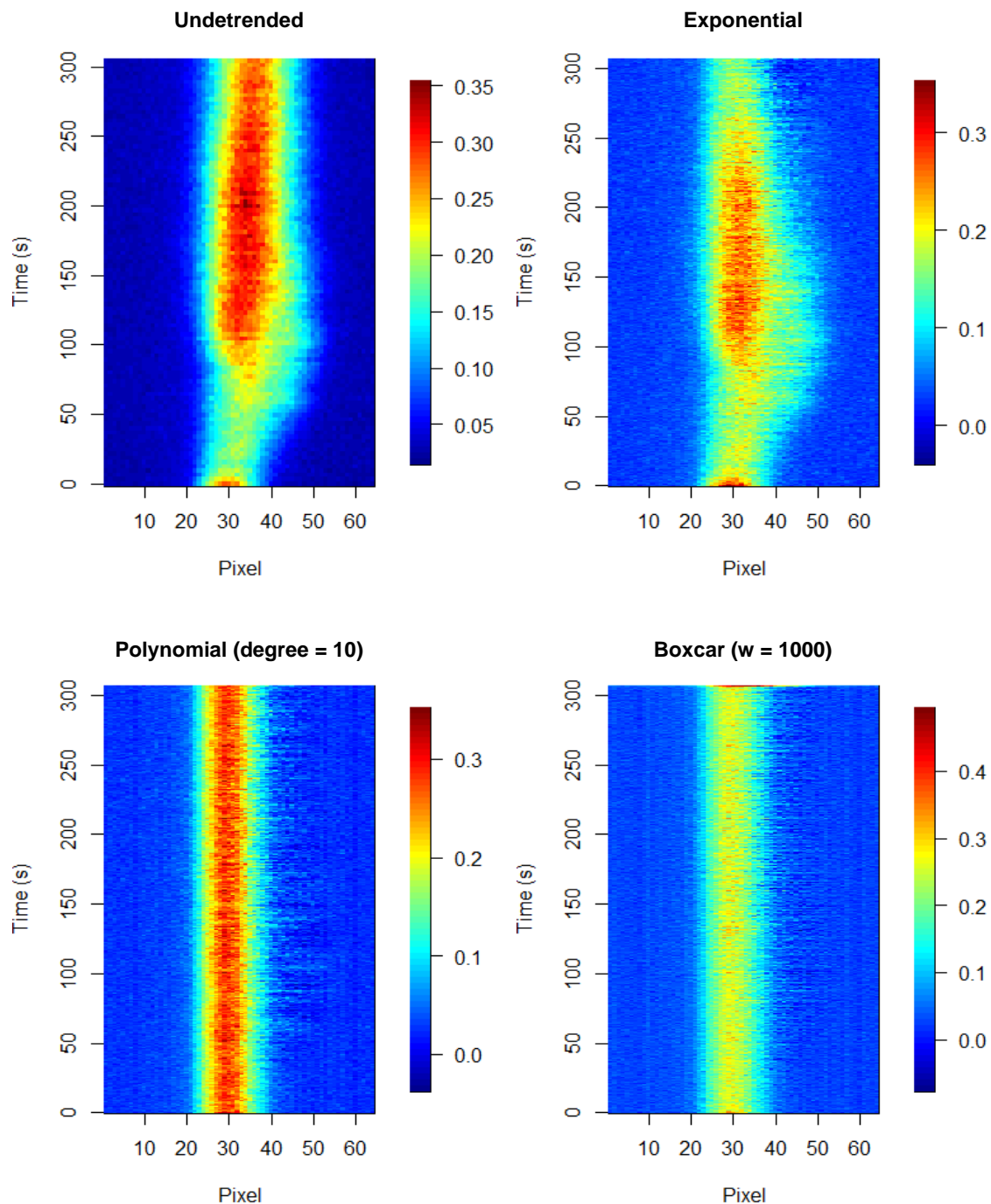


Figure 3.4. Intensity carpets of detrended “Pax.dat” data.

To better visualize the difference in the results, let us only focus on column 30 of the raw data matrix. Type:

```
# Binning the raw data
b.m <- binTimeSeries(data.m[30,], acqTime = 1e-3, nIntervals = 100)

# Detrending the data
d.e <- detrendTimeSeries(data.m[30,], 1e-3, 100, algorithm = "exp")
d.p <- detrendTimeSeries(data.m[30,], 1e-3, 100, algorithm = "poly", degree = 10)
d.b <- detrendTimeSeries(data.m[30,], 1e-3, 100, algorithm = "boxcar", w = 1e3)
```

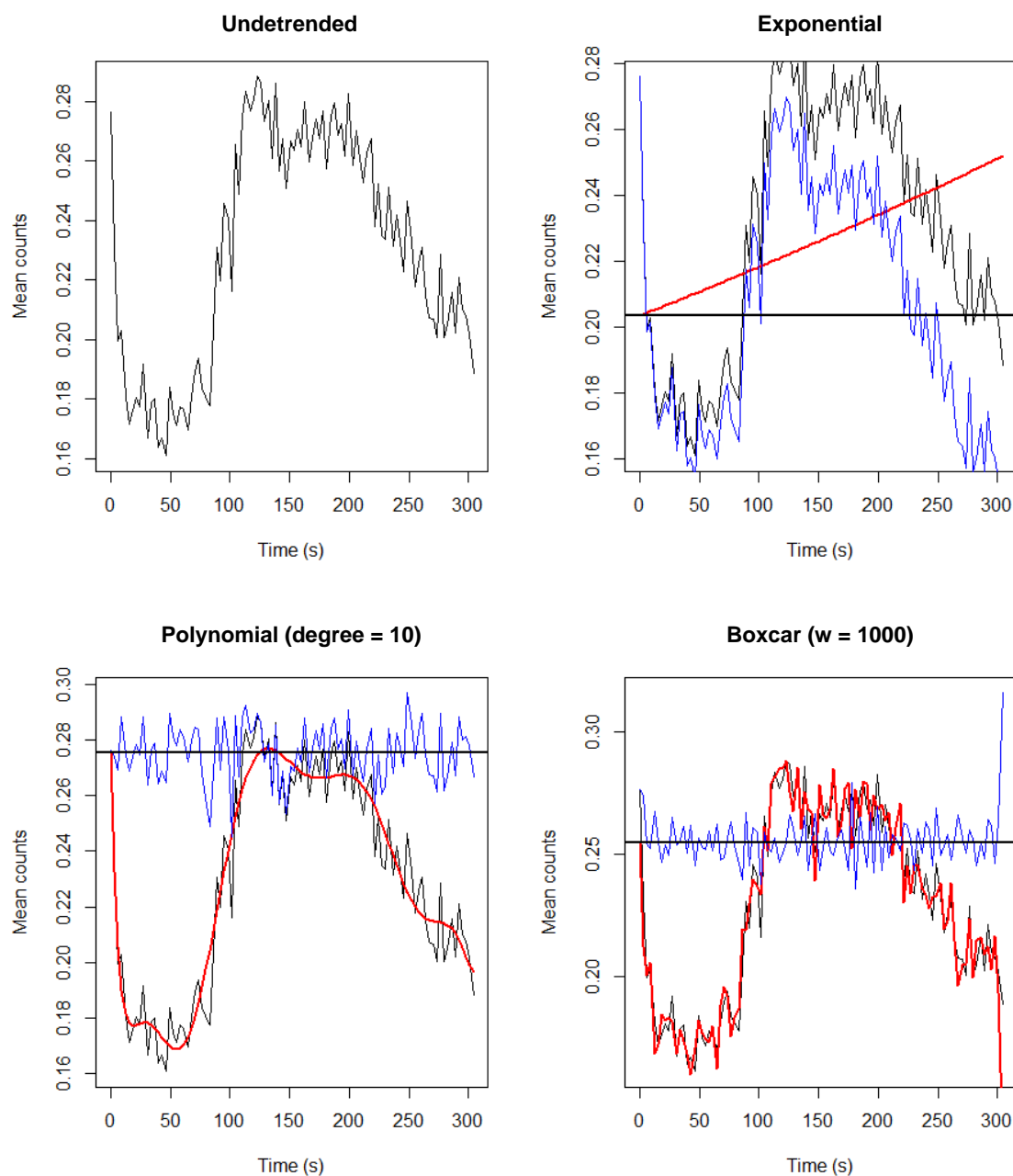


Figure 3.5. Detrended data graphs of column 30 in the matrix in the upper-left panel of figure 3.4.

Note that now the fluorescence fluctuation profile follows a random trend (upper-left panel in figure 3.5), which is best fit a polynomial function with a degree of 10.

3.2.1 Compensation value for trend correction (‘max’ parameter)

The `max` parameter of the `detrendTimeSeries()` function determines if the detrending process will make the whole data match the first or the maximum value of the undetrended time series when `max = FALSE` (default) or `max = TRUE`, respectively. This option helps preserve most of the information as the result is not limited by the information at the very beginning of the data. For the case of photobleaching correction, the first and the maximum value of the time series tend to be located at the same position, so this option does not make any difference for most cases.

To test the `max` parameter, type:

```
# Binning the raw data
b.m <- binMatrix(data$V1, lineTime = 1e-3, nIntervals = 100, columns = 64)

# Detrending the data (Exponential)
det_data.e <- matrix(0, nrow = dim(data.m)[1], ncol = dim(data.m)[2])
for (i in 1:64){
  det_data.e[i,] <- detrendTimeSeries(data.m[i,], 1e-3, 100, algorithm = "exp", plot = F)
}
det.binned <- binMatrix(det_data.e, 1e-3, 500, 64)

# Detrending the data (Polynomial)
det_data.p <- matrix(0, nrow = dim(data.m)[1], ncol = dim(data.m)[2])
for (i in 1:64){
  det_data.p[i,] <- detrendTimeSeries(data.m[i,], 1e-3, 100, algorithm = "poly",
                                     degree = 10, plot = F)
}
det.binned <- binMatrix(det_data.p, 1e-3, 500, 64)

# Detrending the data (Boxcar)
det_data.b <- matrix(0, nrow = dim(data.m)[1], ncol = dim(data.m)[2])
for (i in 1:64){
  det_data.b[i,] <- detrendTimeSeries(data.m[i,], 1e-3, 100, algorithm = "boxcar",
                                     w = 1000, plot = F)
}
det.binned <- binMatrix(det_data.b, 1e-3, 500, 64)
```

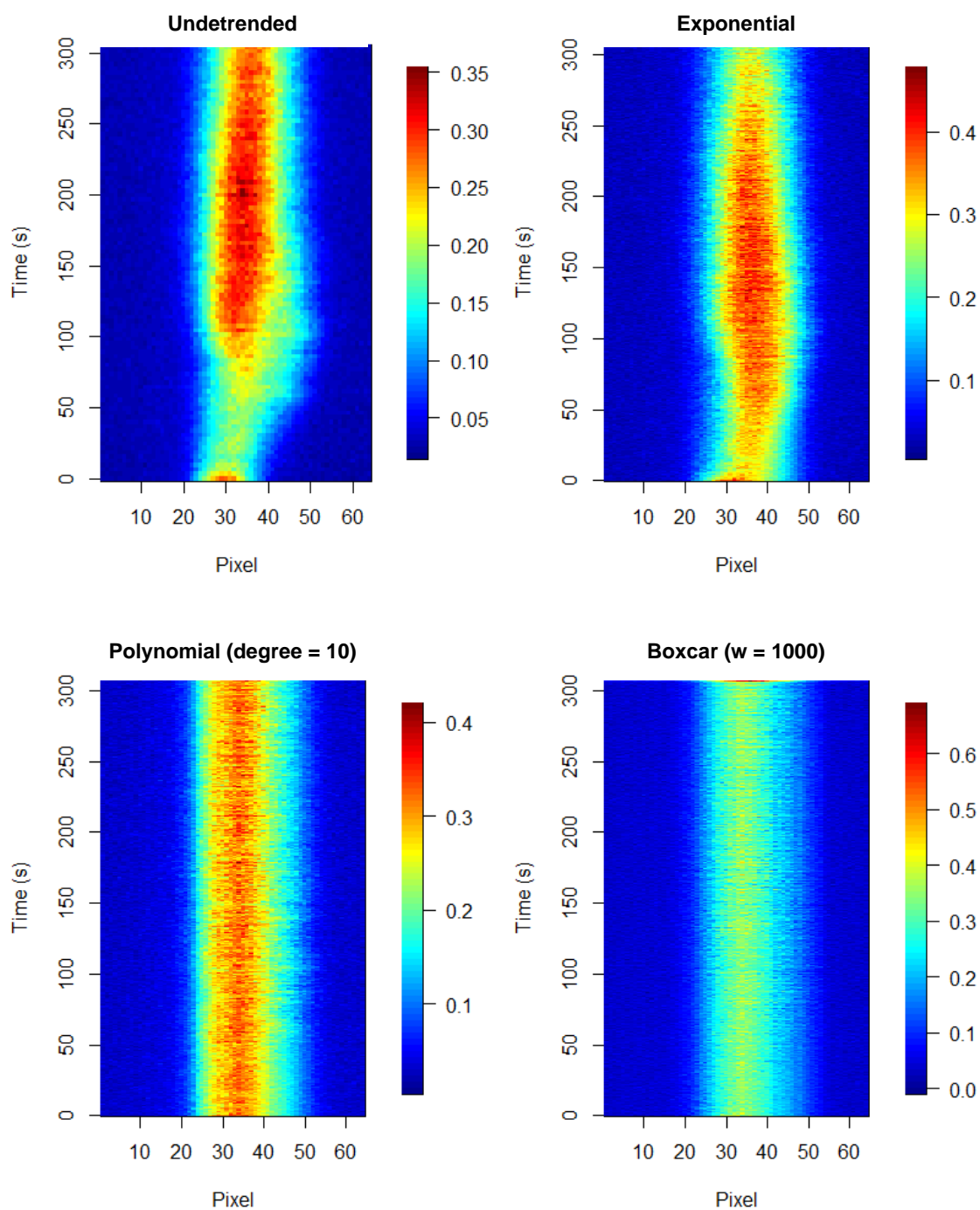



Figure 3.6. Intensity carpets of detrended “Pax.dat” data (‘max’ parameter set to TRUE).

4. MODEL FITTING

A transport model, which contains physical information about the diffusive nature of the fluorophores, can be fitted to the correlation data to know the value of relevant parameters such as the diffusion coefficient, the number of molecules within the observation volume, the triplet time, diffusion time, among others.

The `fitFCS()` function uses the 'Non-linear Least Squares' method (through the `nls()` R base function) to fit a physical model to a data set. This function is preloaded with five different models:

❖ Two-dimensional diffusion (D2D)

$$G(\tau) = \frac{\gamma}{N} \left(1 + \frac{\tau}{\tau_D}\right)^{-1}$$

❖ Two-dimensional diffusion with triplet state (D2DT)

$$G(\tau) = \frac{\gamma}{N} \left(1 + \frac{\text{Exp}(-\tau/\tau_B)}{1-B}\right) \left(1 + \frac{\tau}{\tau_D}\right)^{-1}$$

❖ Three-dimensional diffusion (D3D)

$$G(\tau) = \frac{\gamma}{N} \left(1 + \frac{\tau}{\tau_D}\right)^{-1} \left(1 + \frac{s^2 \tau}{u^2 \tau_D}\right)^{-1/2}$$

❖ Three-dimensional diffusion with triplet state (D3DT)

$$G(\tau) = \frac{\gamma}{N} \left(1 + \frac{\text{Exp}(-\tau/\tau_B)}{1-B}\right) \left(1 + \frac{\tau}{\tau_D}\right)^{-1} \left(1 + \frac{s^2 \tau}{u^2 \tau_D}\right)^{-1/2}$$

❖ Three-dimensional diffusion of two species (D3D2S)

$$G(\tau) = \frac{\gamma}{N} \left[(1-Y) \left(\left(1 + \frac{\tau}{\tau_{D1}}\right)^{-1} \left(1 + \frac{s^2 \tau}{u^2 \tau_{D1}}\right)^{-1/2} \right) + Y \left(\left(1 + \frac{\tau}{\tau_{D2}}\right)^{-1} \left(1 + \frac{s^2 \tau}{u^2 \tau_{D2}}\right)^{-1/2} \right) \right]$$

where γ is a geometric factor that depends on the illumination profile (For confocal excitation its magnitude approaches to $\gamma = 1/\sqrt{8} \approx 0.35$ [3]), N is the total amount of particles in the observation volume and $\frac{\gamma}{N} = G(0)$. The diffusion time is defined as $\tau_D = s^2/4D$, where s and u are the radius and the half-length of the focal volume, respectively. u is usually expressed as $u = ks$, with k being the eccentricity of the focal volume; for single-photon confocal excitation, $k \approx 3$ [4,5]. The fraction of molecules in the triplet state is given by B and τ_B is the triplet time. For the case of two species, Y is the mole fraction of the second component in the mixture.

Starting from a correlation curve vector and a time vector, a data frame containing these and any other parameters inside the physical model to be fit must be first built. As an example, a previously calculated correlation curve that corresponds to freely diffusing AlexaFluor™488 dye in aqueous solution (10 mM Tris/HCl, pH 8) will be used:

```
# Importing the data
data <- readFileFcs("A488.fcs")
class(data); length(data)

## [1] "list"
## [1] 10
```

```
summary(data)
```

```
##               Length Class  Mode
## CountRateArray_1      24000 -none- numeric
## CorrelationArray_1       400 -none- numeric
## PhotonCountHistogramArray_1    64 -none- numeric
## PulseDistanceHistogramArray_1  620 -none- numeric
## CountRateArray_2      24000 -none- numeric
## CorrelationArray_2       400 -none- numeric
## PhotonCountHistogramArray_2    64 -none- numeric
## PulseDistanceHistogramArray_2  602 -none- numeric
## CorrelationArray_3       400 -none- numeric
## CorrelationArray_4       400 -none- numeric
```

“data” is now a list that contains multiple vectors that correspond to multi-channel count, correlation and histogram arrays. Only “CorrelationArray_1”, which contains the tau and correlation vectors for this example, will be considered. Now turn these vectors into a data frame

```
# Creating the time and signal vectors
tau <- as.vector(data$CorrelationArray_1[1,2:100])
g <- (as.vector(data$CorrelationArray_1[2,2:100]) - 1)

# Creating the correlation data frame
G_df <- data.frame(tau = tau, g = g)
dim(G_df)

## [1] 99  2
```

“G_df” is a data frame with two variables: ‘tau’ and ‘g’, both of which have 200 data points. This is what the correlation curve should look like:

```
# Graphing the results
plot(G_df$tau, G_df$g, log = "x", type = "l", main = "A448 autocorrelation",
     xlab = expression(tau(s)), ylab = expression(G(tau)))
```

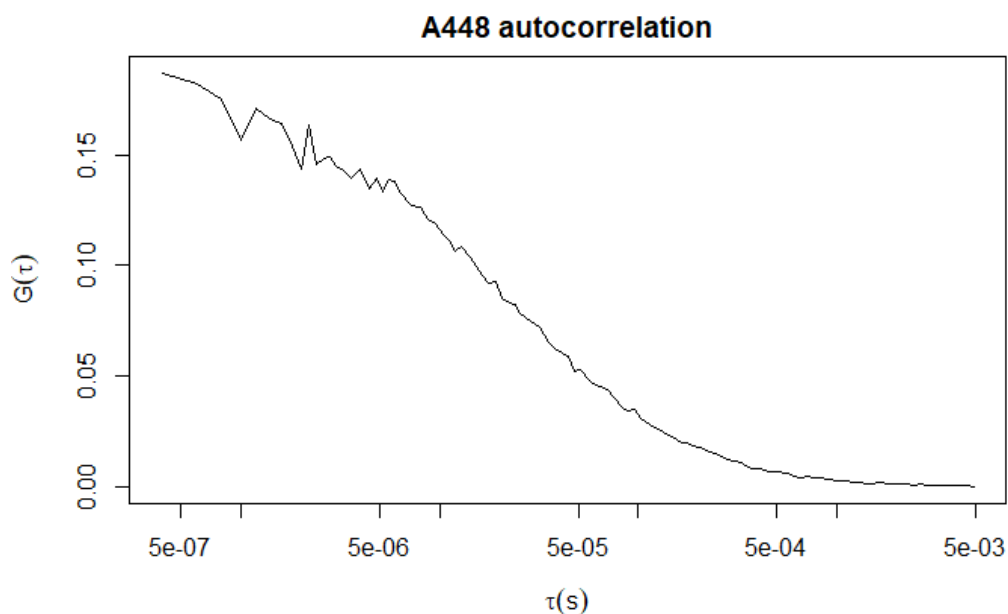


Figure 4.1. Autocorrelation curve of freely diffusing AlexaFluor488 dye in aqueous solution.

Since this experiment corresponds to freely diffusing particles in three dimensions with apparent transition into the triplet state, we'll use the D3DT model to fit the correlation curve. To use the `fitFCS()` function type:

```
# Add parameters 's', 'k', 'B' and 'taub' to existing data frame (this depends on the model)
G_df <- data.frame(G_df, s = 0.3, k = 3, B = 0.2, taub = 0.0000025)

# Define coefficient boundaries for D and G0
start <- list(D = 500, G0 = 0.1, c = 0)
low <- list(D = 1, G0 = 0.01, c = -1)
up <- list(D = 1000, G0 = 10, c = 1)

# Data fitting
G_fit <- fitFCS(data = G_df, start = start, low = low, up = up, type = "D3DT", trace = F)
summary(G_fit)

##
## Formula: g ~ (1 + ((B * exp(-(tau)/taub))/(1 - B))) * ((G0) * ((1 + ((4 *
##      D * tau)/(s^2))))^(-1)) * ((1 + ((4 * D * tau)/((k^2) * (s^2))))^(-1/2))) +
##      c
##
## Parameters:
##      Estimate Std. Error t value Pr(>|t|)
## D  7.104e+02  1.865e+01  38.089   <2e-16 ***
## G0 1.506e-01  8.735e-04 172.424   <2e-16 ***
## c  7.175e-04  6.008e-04   1.194    0.235
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.003352 on 96 degrees of freedom
##
## Algorithm "port", convergence message: both X-convergence and relative convergence (5)
```

Given that the D3DT model includes factors that describe the shape of the three-dimensional space where fluorophores diffuse (s and k) and the triplet state nature (B and $taub$), these have to be added to the data frame that contains the correlation curve in order to the fitting process to work properly.

When defining the coefficient boundaries, one should have a fairly good idea of what the expected values to be obtained are or approximate to. The closer these user-defined boundaries are to the actual phenomenon, the faster and more precise the fitting will perform. This also applies when defining what the constant coefficient values, such as the illumination volume parameters, are.

Since the reported diffusion coefficients for dyes similar to AlexaFluor™488 in water are mostly between 350 and 450 $\mu\text{m}^2/\text{s}$ [6], the chosen starting value was 400 with 1000 and 10 for upper and lower limits, respectively. The boundaries for $G0$ can be defined based on the raw correlation curve (figure 4.1); it appears to reach its maximum and minimum values at ~ 1.55 and ~ 1 , respectively, so a starting value of 1 with 10 and 0.1 for upper and lower limits should be appropriate.

Let us now build the fitted curve:

```
# Graphing the result
fit_curve <- predict(G_fit, G_df$tau)
plot(G_df$tau, G_df$g, log = "x", type = "l", main = "A488 correlation (fitted)",
     xlab = expression(tau(s)), ylab = expression(G(tau)))
lines(G_df$tau, fit_curve, col = "blue")
```

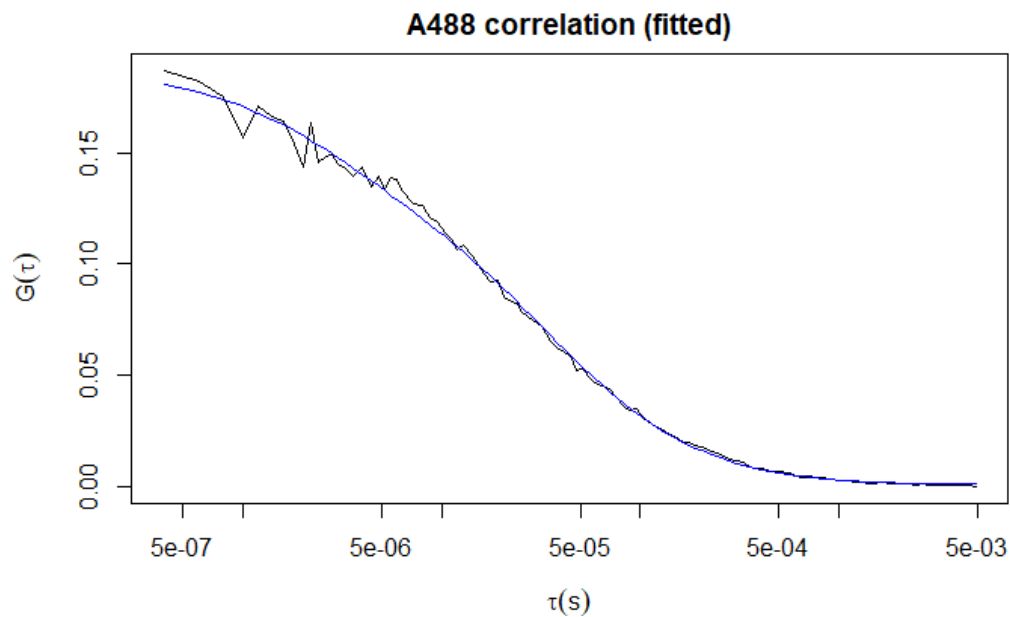


Figure 4.2. Raw correlation curve (black) of freely diffusing AlexaFluor488 dye in aqueous solution and its fitted (blue) version.

Finally, to extract the adjusted coefficients, type:

```
summary(G_fit)$coefficients[,1]
##           D           G0           c
## 7.104462e+02 1.506148e-01 7.174699e-04
```

Please refer to [section 1.4](#) of this document to discover where to find this sample data.

4.1 Goodness-of-fit

Once fitted, looking at the residuals graph can tell us how well the used model describes our data or, in other words, the goodness of the fit.

A residual is defined as the numerical difference between the raw data curve and the adjusted curve for each point. Residuals can be positive or negative if, for a given point, the adjusted curve is located above or under the raw curve, respectively. The closer the residual to zero, the better the fit performed, whereas residuals equal to zero means that the fit performed perfectly.

To visualize the residuals graph, type:

```
# ...continued from previous section

# Graphing the fitting residuals
plot(G_df$tau, G_fit$m$resid(), log = "x", type = "l", main = "A488 correlation fit residuals",
      xlab = expression(tau(s)), ylab = "Residual", ylim = c(-0.02, 0.02))
abline(G_df$tau, 0, col = "red")
```

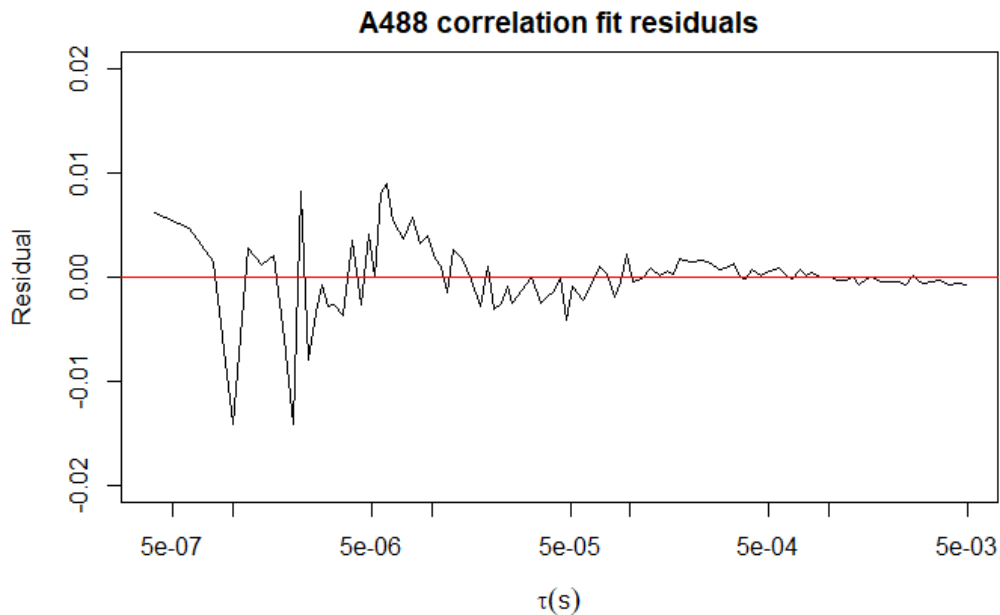


Figure 4.3. Residuals graph of the fitted data in figure 4.2.

4.2 Using a custom-built physical model

The `fitFCS()` function is preloaded with four models that describe diffusion in two or three dimensions while considering (or not) the triplet state of the fluorophore. Though, in order to better study phenomena that drifts from these experimental regimes, the user might want to use a different model that best describes their data and that satisfies their specific research goals. For example, more complex models can be built for better assessing factors such as multiple components, photobleaching, normal or anomalous diffusion in two or more dimensions, one or two-photon excitation, etc.

If none of the default physical models fit your data, FCSlib lets you build and implement your own. First, you may use the following TXT template for defining the expression that defines your model and all the parameters present in it.

```
*Example_model - Notepad
File Edit Format View Help
### MODEL DESCRIPTION (optional)
# Name
# Description

### PARAMETERS
# Parameter 1
a = 1

# Parameter 2
b = 1

# Parameter 3
c = 1

# Parameter n
n = 1

### EXPRESSION
g ~ ((G0,D,COEF1,COEFn,a,b,c,n))

Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

Figure 4.4. Template for custom physical model building.

When building your model, the defined expression should include all the parameters that describe your data. These parameters will be also included in the data frame that has to be created prior to fitting. You will also have to define the value boundaries for any coefficient that you wish to obtain from the fitting process.

Once built, import your model into R by using the `readFileModel()` function:

```
model <- readFileModel(file = "Example_model.txt")
class(model); model$model

## [1] "list"
## [1] "g ~ ((G0,D,COEF1,COEFn,a,b,c,n))"
```

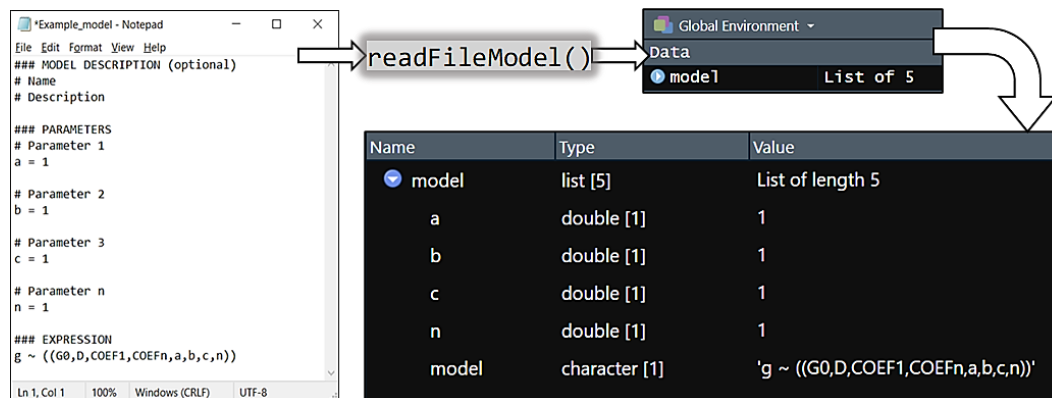


Figure 4.5. When importing a model, a list containing the expression that defines it and the values for all the parameters in it is generated and stored in the R environment.

Now, just like in page 18, build the required elements for the fitting process:

```
# Building the data frame for your data
G_df <- data.frame(G_df, a = model$a, b = model$b, c = model$c, n = model$n)

# Define the boundaries for your coefficients
start <- list(D = 50, G0 = 0.1, COEF1 = 0.025, COEFn = 0.75)
low <- list(D = 1, G0 = 0.01, COEF1 = 0.001, COEFn = 0.01)
up <- list(D = 100, G0 = 10, COEF1 = 0.5, COEFn = 3)

# Data fitting
G_fit <- fitFCS(data = G_df, start = start, low = low, up = up, model = model$model, trace = F)
```

From this point on, follow the same instructions listed at the start of [section 4](#).

5. FLUORESCENCE CORRELATION SPECTROSCOPY

FCS is a set of non-invasive techniques based on the analysis of fluorescence intensity fluctuations caused by the molecular traffic through the observation volume, where the fluorescence signal $F(t)$ is acquired over time at frequencies comparable to those of the molecular dynamics of the process of interest [7]. These fluctuations are obtained by subtracting the mean value of the fluorescence signal at each time point and contain information about the kinetics of the molecules under study, even at low concentrations (i.e. comparable to those of organic molecules in live cell environments). By using single-point analysis, FCS provides information about the dynamics of single molecules within the sample.

A signal fluctuation is defined as

$$\delta F(t) = F(t) - \langle F(t) \rangle$$

where $F(t)$ is the signal intensity at a given time point and $\langle F(t) \rangle$ is the mean value of the whole series. Then, the correlation between $\delta F(t)$ and $\delta F(t + \tau)$ is calculated using the General Autocorrelation Function:

$$G(\tau) = \frac{\langle \delta F(t) \cdot \delta F(t + \tau) \rangle}{\langle F(t) \rangle^2}$$

where t refers to the time point of acquisition, τ is the delay time at which $\delta F(t)$ is computed and $\langle \dots \rangle$ indicates average. The value of $G(\tau)$ represents the similarity of the signal with itself through different time intervals τ . This means that, for example, if the position of a molecule at a given time t is very similar to its position at a time $t + \tau$, there will be a positive contribution to $G(\tau)$. On the other hand, $G(\tau)$ will decrease as the molecule position changes at longer correlation times.

- ✓ *In this section, you will learn how to use the `fcs()` and `pcf()` functions to analyze single-point and line-scanning data, extract information such as diffusion coefficients, particle concentration, observation volume dimensions, etc. and measure single-particle dynamics in time and space.*

5.1 Single-Point FCS

As an example, the fluorescence signal of a solution of Cy5 dye at a concentration of 33 nM will be analyzed. This experiment was carried out in an Olympus FV1000 confocal microscope with single photon excitation, a 60X objective, 1.3 NA and a pinhole of 140 μm for an excitation volume of 344 nm width. Excitation was provided by a 635 nm solid-state laser at 0.1 % power. Please refer to [section 1.4](#) of this document to discover where to find this sample data.

5.1.1 Data exploration

First, use the `readFileTiff()` function to import the data:

```
f <- readFileTiff("Cy5.tif")[,1]
dim(f)

## [1] 2048 5000
```

Note that `f` is a matrix of 2048 x 5000 dimensions. This is because the confocal imaging system used for this experiment arranges the photon counting data as a two-dimensional image for better handling and storage. The sampling rate was 2 μs . Let us now create a data frame with the FCS data which here-and-after will be called `Cy5`:

```
# Creating the time and signal (f) vectors
f <- as.vector(f)
acqTime = 2E-6
time <- (1:length(f))*acqTime

# Creating the raw data frame
Cy5 <- data.frame(t = time, f)
```

The first 10 ms of the time series should look like this:

```
plot(Cy5[1:5000,], xlab = "Time (s)", ylab = "Fluorescence Intensity",
     main = "Cy5", type = "l")
```

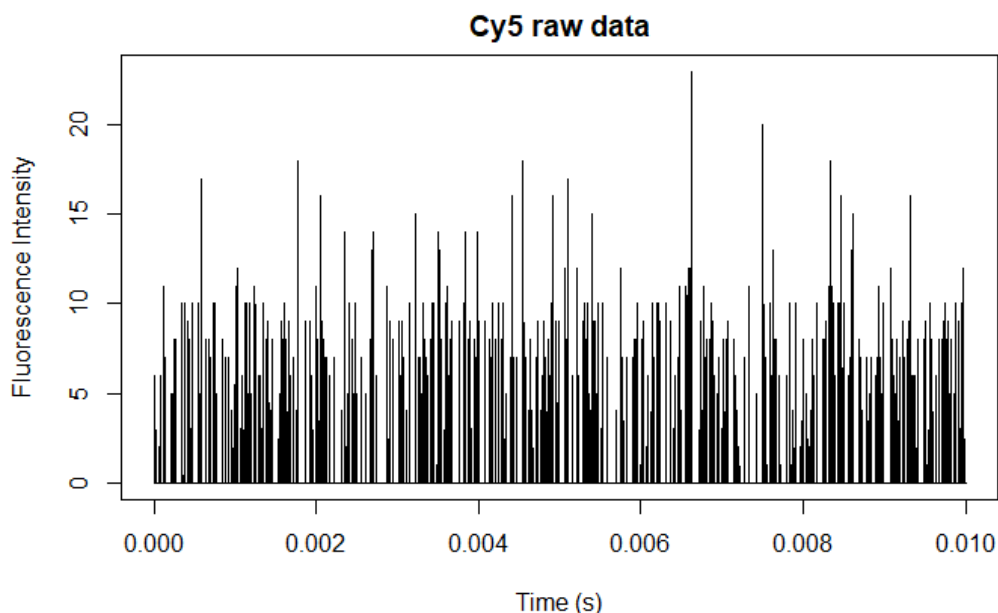


Figure 5.1. First 10 milliseconds (5000 points) of a Cy5 dye single-point scanning experiment.

In order to search for possible global trends in the data (such as photobleaching or sample movement) that may affect the analysis, we'll use the `binTimeSeries()` function to group all the points in several intervals of a given user-specified length and represent them as the cumulative sum of the photon counts, by typing:

```
# Binning the data
f.b <- binTimeSeries(f[length(f):1], acqTime = acqTime, nIntervals = 1000, mode = "sum")
```

The original data vector, `f`, contains a total of 10,240,000 points (2048x5000) that can be grouped in 625 intervals of 16,384 (2^{14}) points each. `f.b` is now a data frame that contains two vectors: the time vector `Time` is also of length equal to 625 and each interval represents ~32.8 ms, which yields a total experiment length of about 20 seconds. This is what the cumulative count sum representation should look like:

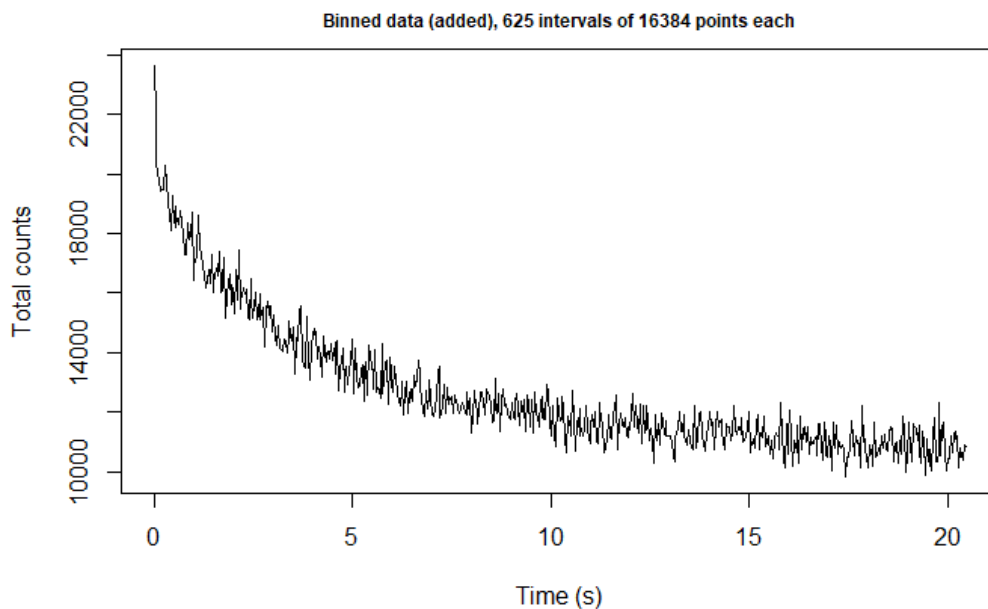


Figure 5.2. Cumulative count sum representation of the whole Cy5 raw data time series.

Note that this step was useful to discover that the data presents a progressive decrease in photon counts due to photobleaching. If the temporality of the dynamic process you are interested in studying matches the temporality of the global trend affecting your data, you should then consider running a detrending routine before proceeding to the analysis.

5.1.2 Building the autocorrelation curves

The `fcs()` function receives three parameters: `x` (mandatory), `y` (optional) and `nPoints` (optional). `x` and `y` are the vectors containing the fluorescence signal information, where `y` is only used when cross-correlation between two channels is to be performed. `nPoints` is the length of the final correlation curve; it is set to 25,000 by default. Briefly, the `fcs()` function divides the original signal vector of length n into sub-vectors of size $nPoints*2$; once all sub-vectors have been analyzed, these are then averaged and showed as the final autocorrelation curve.

To calculate the autocorrelation of the signal, type:

```
g <- fcs(x = Cy5$f)
```

The result of the `fcs()` function has now been assigned to the object `g`, which contains the autocorrelation curve.

```
tau <- Cy5$t[1:length(g)]
G <- data.frame(tau,g)
plot(G, xlab = expression(tau(s)), ylab = expression(G(tau)),
      main = "Cy5 autocorrelation, nPoints = 25000", log = "x", type = "l")
```

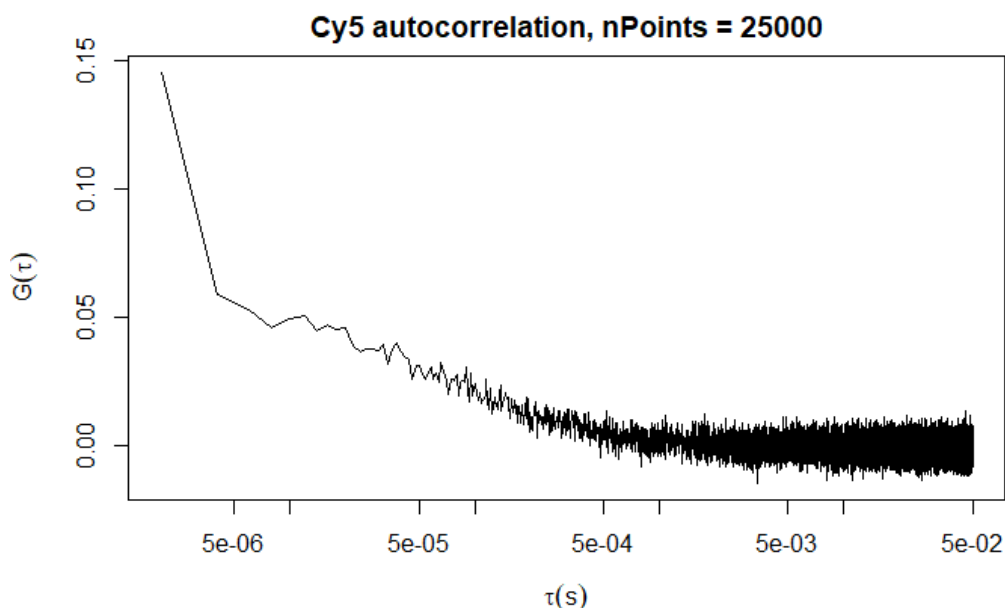


Figure 5.3. Autocorrelation curve of free-diffusing Cy5 dye. Parameter ‘nPoints’ set to 25000.

The parameter `nPoints` can be further adjusted to better assess the dynamic phenomenon in study (e.g. free diffusion in three dimensions, for the case of this example) and for better understanding of the diffusive nature of the fluorophores. Let us then reduce `nPoints` to 2500:

```
g <- fcs(x = Cy5$f, nPoints = 2500)
tau <- Cy5$t[1:length(g)]
G <- data.frame(tau,g)
plot(G, xlab = expression(tau(s)), ylab = expression(G(tau)),
      main = "Cy5 autocorrelation, nPoints = 2500", log = "x", type = "l")
```

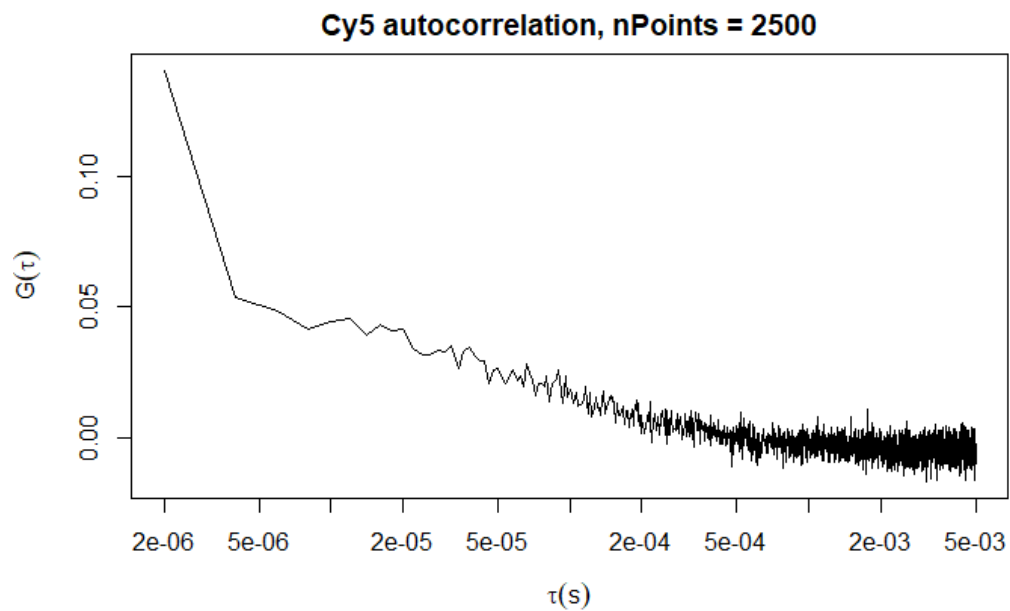


Figure 5.4. Autocorrelation curve of free-diffusing Cy5 dye. Parameter 'nPoints' set to 2500.

Most of the unnecessary portion of the autocorrelation curve has been cut and one can now proceed with the analysis.

5.1.2.1 Compensating for negative $G(\tau)$ values

Sometimes, negative $G(\tau)$ values when $G(\tau \rightarrow \infty)$ can result in a rather poor fitting result. To compensate for this issue, one can estimate the mean of $G(\tau \rightarrow \infty)$ and add its absolute value to the autocorrelation data:

```
# Compensating for negative G(tau) values
c <- abs(mean(tail(G$g, n = 2000)))
Gc <- data.frame(tau = G$tau, g = G$g + c)

# Graphing the result
par(mfrow = c(1,2))
plot(G, xlab = expression(tau(s)), ylab = expression(G(tau)),
     main = "Uncompensated", log = "x", type = "l")
abline(h = 0)
plot(Gc, xlab = expression(tau(s)), ylab = expression(G(tau)),
     main = "Compensated", log = "x", type = "l")
abline(h = 0)
```

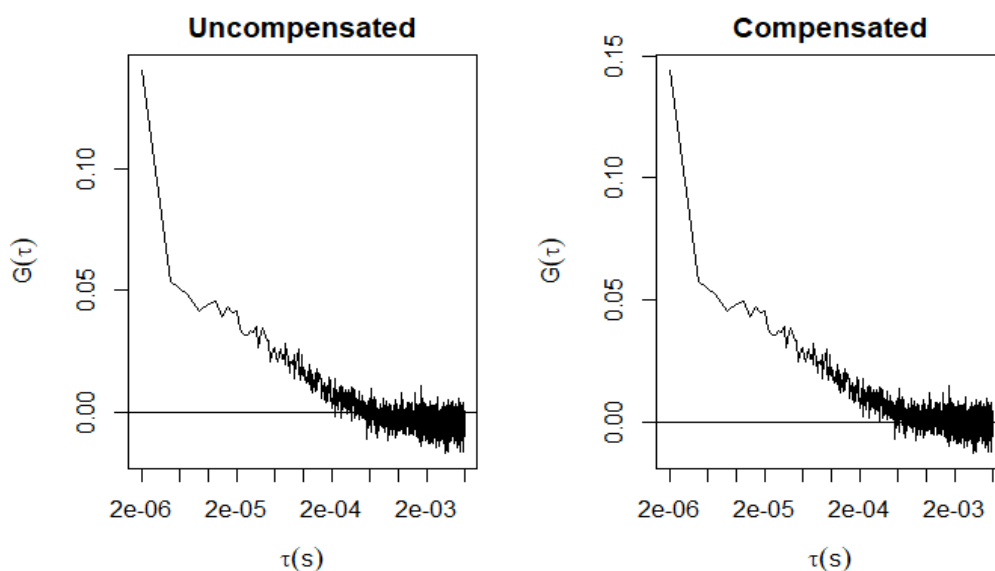


Figure 5.5. Autocorrelation curve of free-diffusing Cy5 dye. Parameter ‘nPoints’ set to 2500.

Note that the last component of the autocorrelation curve is now centered around $G(\tau) = 0$, which will help obtain more accurate results.

This compensation feature is included in the built-in physical models contained in the `fitFCS()` function. Though, if a custom-made model is to be used, the compensating component ‘c’ must be either included in the model or manually introduced (see code [above](#)) before the fitting process. For the rest of the examples in this document, this compensation process will be carried out automatically.

5.1.3 Physical model fitting and coefficient extraction

The autocorrelation curve in figure 5.4 shows clear presence of the triplet state in the fluorescence signal (relatively high correlation value at $G(\tau \rightarrow 0)$). To account for this phenomenon, a model for free diffusion in three dimensions and the triplet state shall be used to fit the correlation data.

Once the correlation curve `g` has been created, a data frame containing the values of the known parameters must be then defined. The radius of the focal volume must be determined experimentally and varies for each specific optical system. For this example, we'll use a value of 0.27 μm for `s`, as described in [3].

```
# Creating the data frame for the fitting process
G <- data.frame(G, s = 0.27, k = 3, B = 0.9, taub = 0.000001)
head(G)

##      tau      g      s k    B  taub
## 1 2.0e-06 0.14425478 0.27 3 0.9 1e-06
## 2 4.0e-06 0.05747627 0.27 3 0.9 1e-06
## 3 6.0e-06 0.05252053 0.27 3 0.9 1e-06
## 4 8.0e-06 0.04542426 0.27 3 0.9 1e-06
## 5 1.0e-05 0.04835121 0.27 3 0.9 1e-06
## 6 1.2e-05 0.04947705 0.27 3 0.9 1e-06
```

Now, three lists that contain the initial values of the model coefficients, as well as their upper and lower limits, are to be defined. The input values shown here are the expected ones for the real experimental data to be very similar or close to. This step is critical for the accurate computation of the real values.

```
start <- list(D = 100, G0 = 0.1, c = 0)
low <- list(D = 10, G0 = 0.01, c = -1)
up <- list(D = 1000, G0 = 10, c = 1)
```

Next, proceed to use the `fitFCS()` function to adjust the D3DT model to the Cy5 autocorrelation data:

```
Cy5_fit <- fitFCS(data = G, start = start, low = low, up = up, type = "D3DT", trace = F)
summary(Cy5_fit)

##
## Formula: g ~ (1 + ((B * exp(-(tau)/taub))/(1 - B))) * ((G0) * ((1 + ((4 *
##      D * tau)/(s^2)))^(-1)) * ((1 + ((4 * D * tau)/((k^2) * (s^2))))^(-1/2))) +
##      c
##
## Parameters:
##      Estimate Std. Error t value Pr(>|t|)
## D    2.943e+02  1.073e+01   27.42  <2e-16 ***
## G0    6.012e-02  1.083e-03   55.52  <2e-16 ***
## c    -4.518e-03  8.687e-05  -52.01  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.003508 on 2497 degrees of freedom
##
## Algorithm "port", convergence message: relative convergence (4)
```

By using the `predict()` function, the `nls` object generated in the previous step (`Cy5_fit`) is transformed into a vector that contains the adjusted curve.

```
fit_curve <- predict(Cy5_fit, tau)
```

Finally, to obtain the resulting graphs containing the fitted and unfitted data, as well as the goodness-of-fit (residuals), type:

```
plot(G$g~G$tau, xlab = expression(tau(s)), ylab = expression(G(tau)),
     main = "Cy5 fitted data (D3DT)", log = "x", type = "l")

lines(fit_curve~G$tau, col = "blue")

plot(G$tau, Cy5_fit$m$resid(), log = "x", type = "l",
     xlab = expression(tau(s)), ylab = "Residual")

abline(G$tau, 0, col = "red")
```

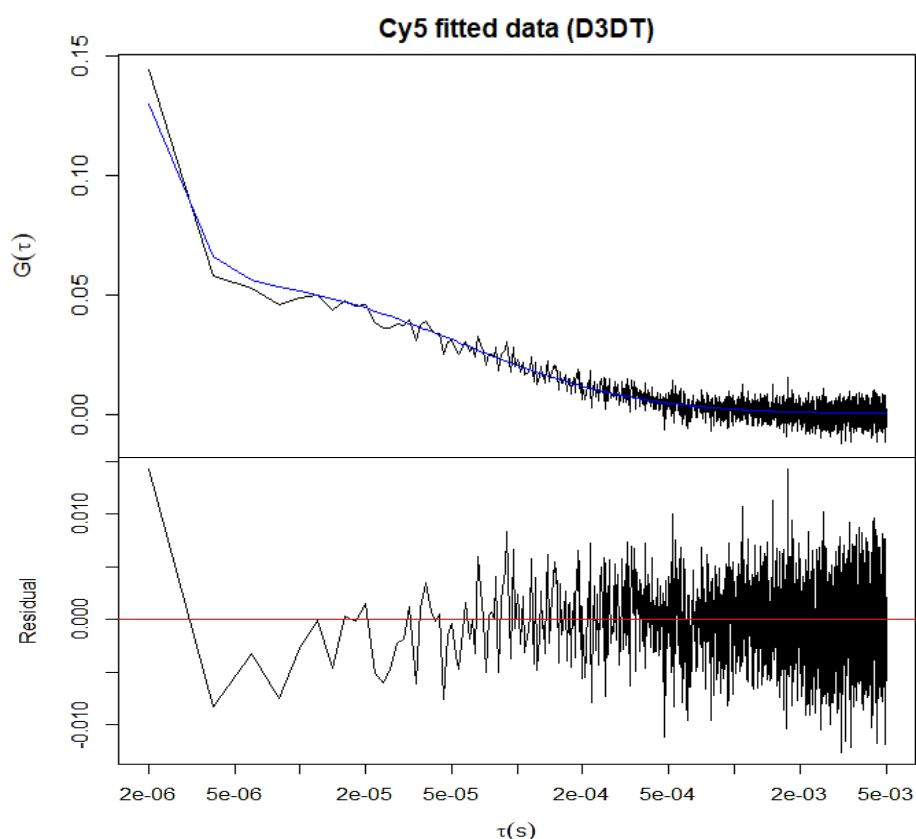


Figure 5.6. Raw autocorrelation curve (black) of free-diffusing Cy5 dye and its fitted (blue) version (upper panel) and the goodness-of-fit (residuals) graph (lower panel).

Now, to access the calculated model coefficients, type:

```
summary(Cy5_fit)$coefficients[,1]
```

##	D	G0	c
##	294.29433459	0.06011953	-0.00451836

5.1.3.1 Simplifying the correlation data

To prevent overweighting of the model in the region of near-to-zero correlation, it is useful to average $G(\tau)$ at longer τ values. The `simplifyFCS()` function performs a \log_{10} weighted binning on the autocorrelation function as it was developed to balance the weight of the long-term temporal scale trending behavior of the curve, which commonly fluctuates around the zero-correlation range, hence overweighting fitting around 'noisy data'. In other words `simplifyFCS()` significantly reduces the amount of data in the autocorrelation vector without altering its overall structure, allowing for further physical model fitting while obtaining consistent results.

The `simplifyFCS()` function receives three parameters: `g`, `tau`, which correspond to the correlation and tau vectors, respectively and `step` (default = 1), which affects the final length of the vector. To simplify the correlation data type:

```
# Simplifying the correlation data
sG <- simplifyFCS(G$g, G$tau)

# Creating the data frame for the fitting process
sG <- data.frame(sG, s = 0.27, k = 3, B = 0.9, taub = 0.000001)

# Model fitting
Cy5_fit_s <- fitFCS(data = sG, start = start, low = low, up = up, type = "D3DT", trace = F)
summary(Cy5_fit_s)

##
## Formula: g ~ (1 + ((B * exp(-(tau)/taub))/(1 - B))) * ((G0) * ((1 + ((4 *
##      D * tau)/(s^2)))^(-1)) * ((1 + ((4 * D * tau)/((k^2) * (s^2))))^(-1/2)))) + c
##
## Parameters:
##      Estimate Std. Error t value Pr(>|t|)
## D   332.107016  42.603312   7.795 1.72e-08 ***
## G0   0.062772   0.001519  41.334 < 2e-16 ***
## c    -0.005045   0.001323  -3.813 0.000693 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.003695 on 28 degrees of freedom
##
## Algorithm "port", convergence message: relative convergence (4)
```

Graph the results:

```
# Creating the fitted data graph
fit_curve_s <- predict(Cy5_fit_s, tau)

plot(sG$g~sG$tau, xlab = expression(tau(s)), ylab = expression(G(tau)),
     main = "Cy5 fitted data (D3DT) - Simplified", log = "x", type = "l")

lines(fit_curve_s~sG$tau, col = "blue")

# Creating the goodness-of-fit graph
plot(sG$tau, Cy5_fit_s$m$resid(), log = "x", type = "l",
     xlab = expression(tau(s)), ylab = "Residual")
abline(sG$tau, 0, col = "red")
```

And finally, extract the coefficients:

```
summary(Cy5_fit_s)$coefficients[,1]

##           D           G0           c
## 332.107016250  0.062771912 -0.005044664
```

The graphs should look like this:

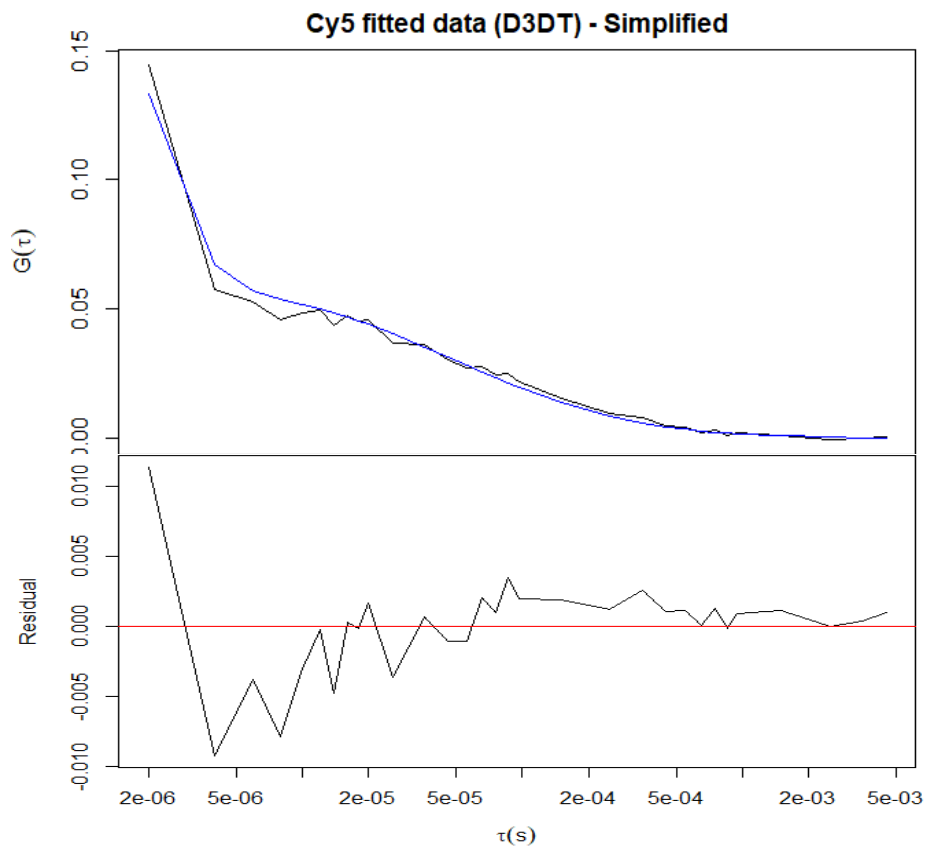


Figure 5.7. Simplified correlation data. Raw autocorrelation curve (black) of free-diffusing Cy5 dye and its fitted (blue) version (upper panel) and the goodness-of-fit (residuals) graph (lower panel).

Now, the autocorrelation analysis over the simplified curves yields a diffusion coefficient of $332.11 \mu\text{m}^2/\text{s}$, which is much closer to the reported value for Cy5 in aqueous solution [6]. The average number of molecules within the observation volume is $N = \gamma/G_0 = 0.35/0.0628 \approx 5.6$, and the concentration of molecules can be calculated as $C = N/V_{\text{eff}}$, where $V_{\text{eff}} = \pi^{3/2}s^2u^2 \approx 0.27 \text{ fL}$ (for single photon excitation).

5.1.4 Calculating the dimensions of the Point Spread Function

By following the same *modus operandi* as in [section 5.1.3](#), one can also determine the dimensions of the observation volume when performing an experiment with fluorophores of known diffusion coefficient. The logic behind this idea is that, instead of fitting a physical model and try to find D while assigning s a constant value in the data frame, these two parameters now swap roles and D remains constant as one tries to find the best adjusted value for s through the fitting process.

The Cy5 data set will continue to be used as an example of this feature. Note that the obtained diffusion coefficient for Cy5 in aqueous solution was $\sim 354 \mu\text{m}^2/\text{s}$ and that, for single photon excitation, k is equal to 3.

In order to calculate the radial waist length of the observation volume, type:

```
# Importing the data
f <- readFileTiff("Cy5.tif")[,1]

# Creating the time and signal vectors
f <- as.vector(f)
acqTime = 2E-6
time <- (1:length(f))*acqTime

# Creating the raw data frame
Cy5 <- data.frame(t = time, f)

# Calculating the autocorrelation curve
g <- fcs(x = Cy5$f, nPoints = 2500)
tau <- Cy5$t[1:length(g)]
G <- data.frame(tau,g)

# Simplifying the correlation data
sG <- simplifyFCS(G$g, G$tau)

# Getting ready for the fitting process
sG <- data.frame(sG, D = 354, k = 3, B = 0.9, taub = 0.000001)
start <- list(s = 0.2, G0 = 0.1, c = 0)
low <- list(s = 0.1, G0 = 0.01, c = -1)
up <- list(s = 0.4, G0 = 10, c = 1)

# Fitting results
Cy5_fit <- fitFCS(data = sG, start = start, low = low, up = up, type = "D3DT", trace = F)
summary(Cy5_fit)$coefficients[,1]

##           s           G0           c
## 0.278757373 0.062771912 -0.005044664
```

The adjusted value for s is $0.28 \mu\text{m}$, which is very close to the reported value in [\[3\]](#).

5.1.5 Fluorescence Cross-Correlation Spectroscopy

Calculating the autocorrelation of a signal fluctuation profile allows to understand the behavior of single channel emitting fluorophores, as an indicator of the similarity of the signal with itself overtime. On the other hand, one can study the dynamics of different fluorescently tagged molecules and their interaction by correlating each of the fluorescence channels' signal with each other. Molecules with similar behavior will generate a correlation curve that will help describe the dynamics of interacting populations within the same experiment. In addition to knowing concentration, diffusion coefficients and other parameters, this method can also help unveiling phenomena such as oligomerization, binding, catalysis or complex formation rates.

When analyzing two different signal profiles, simultaneous fluctuations of similar magnitude are assumed to be caused by co-moving particles. When applied to different regions of the same sample, one can assess interaction dynamics in subcellular compartments.

If two separately acquired fluorescence signals F_1 and F_2 are considered, the correlation between them can be calculated with

$$G_{cc}(\tau) = \frac{\langle \delta F_1(t) \cdot \delta F_2(t + \tau) \rangle}{\langle F_1(t) \cdot F_2(t) \rangle}$$

which is a modification of the general correlation function for two channels.

To demonstrate the capability of the `fcs()` function to perform cross-correlation calculation, a single-point experiment simulation will be used, which consists of freely-diffusing particles in three dimensions that contains two populations that fluoresce on a different channel each and a third interacting population that fluoresces on both channels. The simulated particles' diffusion coefficient is 50 $\mu\text{m}^2/\text{s}$; they then form tetramers that diffuse at 30 $\mu\text{m}^2/\text{s}$. Please refer to [section 1.4](#) of this document to discover where to find this sample data.

This is what the first 10 ms of each channel's time series look like:

```
# Importing the data
cc <- read.table("Sim_ccFCS.dat")
dim(cc)

## [1] 3276800      3

# Creating the time and signal vectors
ch1 <- as.vector(cc[,2])
ch2 <- as.vector(cc[,3])
acqTime = 4E-6
time <- (1:length(ch1))*acqTime

# Creating the raw data frame
CC <- data.frame(t = time, ch1 = ch1, ch2 = ch2)

# Graphing the first 2500 points
plot(CC[1:2500,2]~CC[1:2500,1], xlab = "Time (s)", ylab = "Photon counts",
     main = "Raw ccFCS Simulation Data", type = "l", col = "red")
lines(CC[1:2500,3]~CC[1:2500,1], type = "l", col = "green")
```

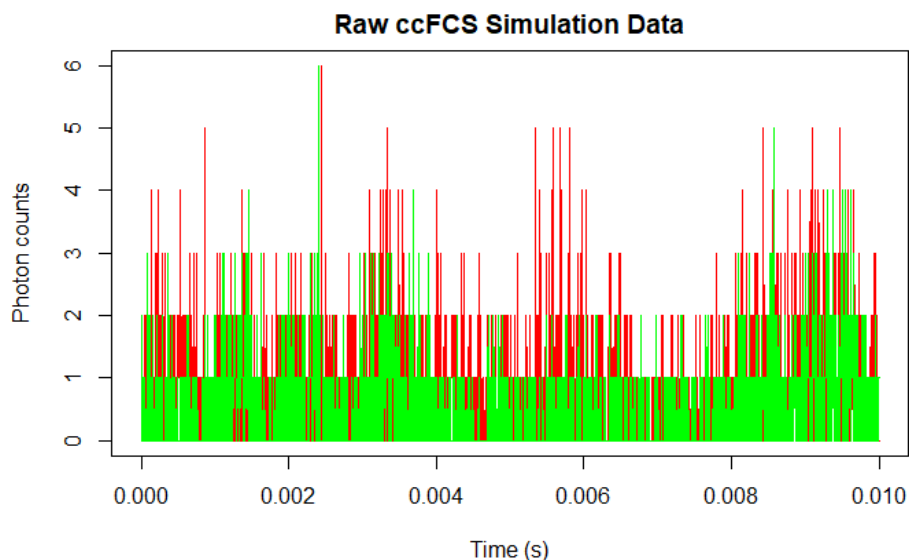


Figure 5.8. First 10 milliseconds (2500 points) of two channel signal intensity profiles.

To calculate the auto and cross-correlation of the two channels type:

```
# Calculating the correlation curves
g1 <- fcs(x = ch1, nPoints = 25000)
g2 <- fcs(x = ch2, nPoints = 25000)
g12 <- fcs(x = ch1, y = ch2, nPoints = 25000)
tau <- CC$t[1:length(g1)]
Gcc <- data.frame(tau, g1, g2, g12)

# Graphing the results
plot(Gcc$g2~Gcc$tau, xlab = expression(tau(s)), ylab = expression(G(tau)),
     main = "Auto and cross-correlation of two channels", log = "x", type = "l", col = "red")
lines(Gcc$g1~Gcc$tau, col = "green"); lines(Gcc$g12~Gcc$tau)
```

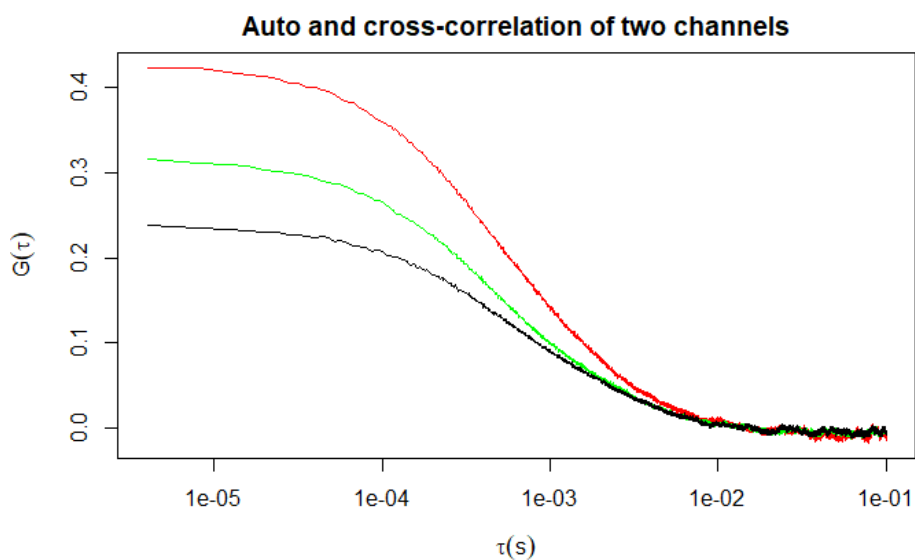


Figure 5.9. Autocorrelation (green and red) and cross-correlation (black) curves.

Note that the interacting population (black curve) represents about 25% of the total particle pool, as depicted by the amplitude of the curves ($G(\tau \rightarrow 0)$).

For simplicity, only the interacting component will be analyzed. Now, for the fitting process, type:

```
# Simplifying the correlation data
sGcc <- simplifyFCS(Gcc$g12, Gcc$tau)

# Setting up the fitting parameters
sGcc12 <- data.frame(tau = sGcc$tau, g = sGcc$g, s = 0.27, k = 3)
start <- list(D = 50, G0 = 0.1, c = 0)
low <- list(D = 1, G0 = 0.01, c = -1)
up <- list(D = 100, G0 = 10, c = 1)

# Fitting results
Gcc12_fit <- fitFCS(data = sGcc12, start = start, low = low, up = up, type = "D3D", trace = F)
summary(Gcc12_fit)$coefficients[,1]

##           D           G0           c
## 25.252273507  0.242793561 -0.005321075
```

5.1.6 Two-component Single-Point FCS

By having a glance at the autocorrelation curve one can get an indicator of the presence of more than one diffusive species in the sample. Naturally, the point in the curve where $G(\tau)$ drastically drops (turning point) indicates the lag time (or diffusion time) at which an average particle of the system has travelled the length of the observation volume away from its initial position. Multiple turning points in the curve depict different particle populations travelling at different speeds; the value of τ at which these turning points are located indicates how fast they move [4].

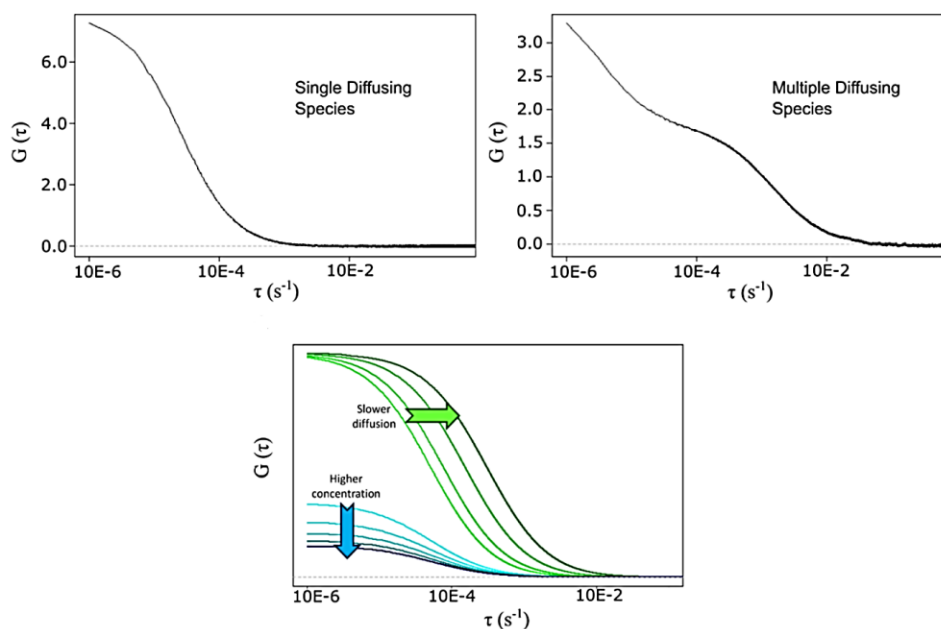


Figure 5.10. The presence of multiple diffusing species will be reflected as multiple turning points in the autocorrelation curves. The value of τ for each of said points is inversely proportional to the diffusion coefficient. The amplitude of the curve ($G(\tau = 0)$) is inversely proportional to the number of particles while its position along the temporal axis reflects their mobility. Adapted from [4].

However, it will become much harder to qualitatively discriminate between the diffusion coefficients of two different particle populations as their values get closer to one another (i.e. less than two orders of magnitude between them).

As an example of this phenomenon, a simulation of two populations of freely diffusing particles of equal concentration with diffusion coefficients of 100 $\mu\text{m}^2/\text{s}$ and 0.1 $\mu\text{m}^2/\text{s}$ will be used. Since the simulated data now encompasses a broader diffusion time τ_D range, a greater value for `nPoints` must be selected. Type:

```
# Importing the data
tc <- read.table("Sim_2CFCS.dat")
dim(tc)

## [1] 3276800      3

# Creating the time and signal vectors
f <- as.vector(tc[,2])
acqTime = 4E-6
time <- (1:length(f))*acqTime
```

```

# Creating the raw data frame
TC <- data.frame(t = time, f = f)

# Calculating the correlation curves
g <- fcs(x = f, nPoints = 100000)
tau <- TC$t[1:length(g)]
G <- data.frame(tau, g)

# Graphing the results
plot(G$g~G$tau, xlab = expression(tau(s)), ylab = expression(G(tau)),
     main = "Two-component autocorrelation", log = "x", type = "l")

# Diffusion time intercepts
tD100 <- (0.27^2)/(4*100)
tD0.1 <- (0.27^2)/(4*0.1)
abline(v = c(tD100, tD0.1), lty = "dashed", col = "blue")

```

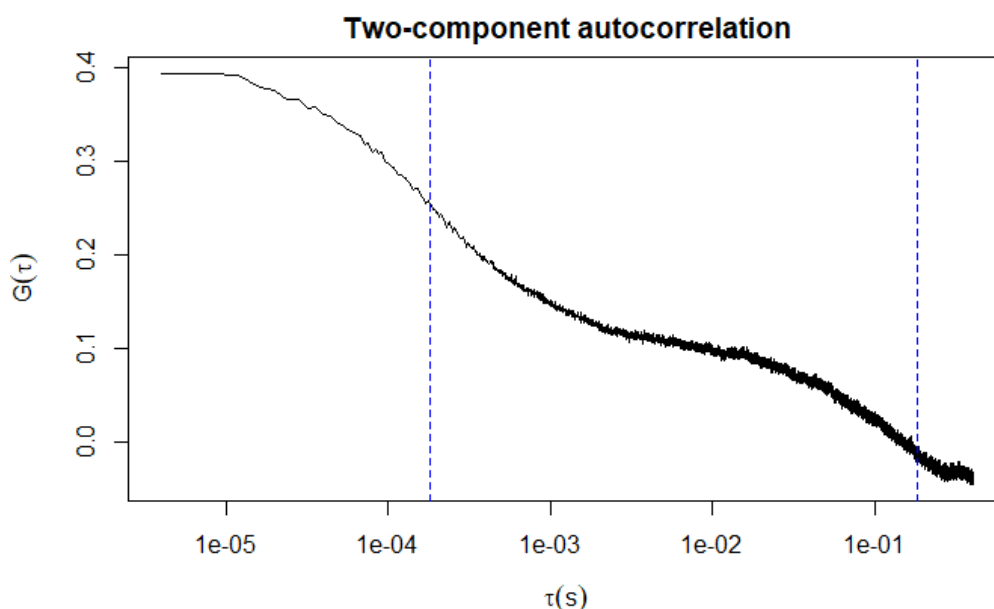


Figure 5.11. Autocorrelation curve of a two-component single-point FCS simulation. The dashed blue lines indicate the predicted diffusion times τ_D for the corresponding simulated diffusion coefficients.

The dashed blue lines in figure 5.11 correspond to the expected $G(\tau)$ decay τ values for each of the simulated diffusion coefficients, which can be predicted with $\tau_D = s^2/4D$. Note that the position of these lines is also aligned with each of the curve's turning points.

Now, for the fitting process, since the simulation contains two different diffusive species, a model for diffusion in three dimensions of two species [8] will be used. Type:

```

# Simplifying the correlation data
sG <- simplifyFCS(G$g, G$tau)

# Setting up the parameters
sG <- data.frame(sG, s = 0.27, k = 3, Y = 0.5)
start <- list(D1 = 100, D2 = 1, G0 = 0.1, c = 0)
low <- list(D1 = 10, D2 = 0.01, G0 = 0.01, c = -1)
up <- list(D1 = 1000, D2 = 10, G0 = 10, c = 1)

```



```

# Fitting results
G_fit <- fitFCS(data = sG, start = start, low = low, up = up, type = "D3D2S", trace = F)
summary(G_fit)$coefficients[,1]

##          D1          D2          G0          c
## 87.43747059 0.05771686 0.59236782 -0.19133133

# Creating the fitted data graph
fit_curve <- predict(G_fit, tau)
plot(sG$sG$tau, xlab = expression(tau(s)), ylab = expression(G(tau)),
     main = "Fitted two-component autocorrelation data", log = "x", type = "l")
lines(fit_curve~sG$tau, col = "blue")

# Creating the goodness-of-fit graph
plot(sG$tau, G_fit$m$resid(), log = "x", type = "l", ylim = c(-0.03,0.03),
     xlab = expression(tau(s)), ylab = "Residual")
abline(sG$tau, 0, col = "red")

```

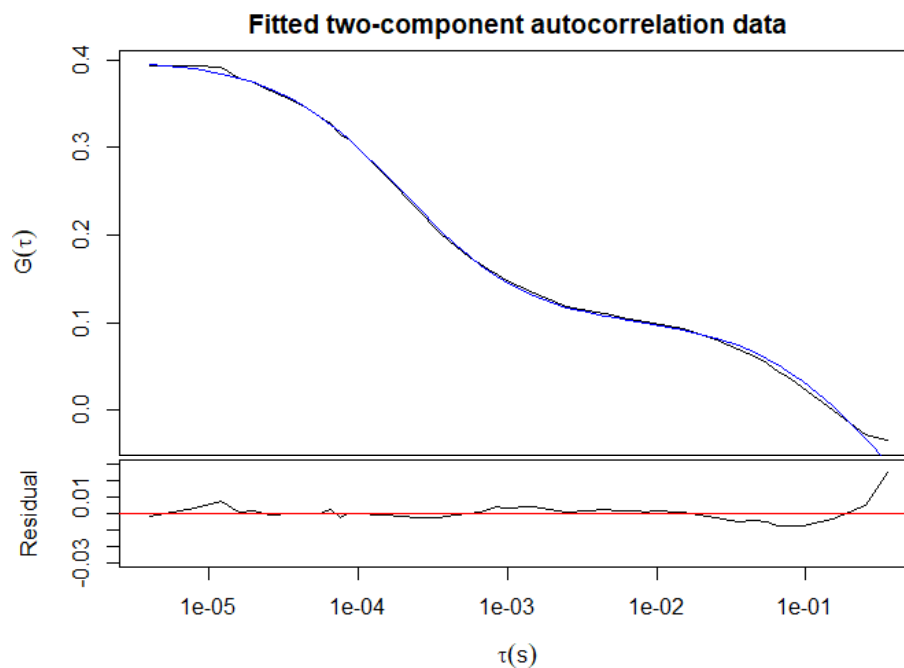


Figure 5.12. Fitted (blue) autocorrelation curve of a two-component single-point FCS simulation and its residuals.

5.2 Scanning-FCS

Even though the classical single-point FCS approach offers high temporal resolution that can help to describe the dynamics of fast-diffusing molecules, it is not intended to provide spatial information within a single experiment.

If the sample is continuously scanned at multiple sequential points across a linear section, a two-dimensional (time and position) fluorescence signal intensity profile is obtained as a result. This allows to have multiple single-point data contiguously separated by a distance as short as the microscope's objective and scanning laser resolutions can provide.

- ✓ *An overview of how the single-point analysis principle can be extended to line-scanning experiments in order to further study the distribution of the molecules' dynamics across a certain region in the sample, is provided in this section. Also, a demonstration of how this can be used to extract different movement and concentration properties based on the particles location and retrieve molecular flux information when performing pair correlation analysis is included.*

5.2.1 Data exploration

As an example, let us study the dynamics of Paxilin-EFGP at the focal adhesion, for which an orbital-scanning experiment will be used. This experiment was performed as described elsewhere [9]. Please refer to [section 1.4](#) of this document to discover where to find this sample data.

First, import the data using the `readFileTiff()` function:

```
# Importing the data
Pax <- read.table("Pax.dat")
class(Pax); dim(Pax)

## [1] "data.frame"
## [1] 19660800      1
```

`Pax` is a `data.frame` with one column, which corresponds to the photon counting signal. In this case, each point in this column is a consecutively scanned point in a 64-pixel long orbit, so the data must be rearranged into a 64-column `matrix`.

```
# Rearranging into a matrix
Pax <- matrix(Pax[,1], nrow = 64)
dim(Pax)

## [1]      64 307200
```

To define the temporal `Time` and spatial `r` ranges of the experiment, type:

```
# Defining experimental parameters
lineTime = 0.001
pixelSize = 0.042
Time <- (1:dim(Pax)[2])*lineTime
r <- 1:64
```

Now, `Pax` is a matrix with 64 columns (pixels) and 307200 rows (line scans) which, considering a line scan time of 1 millisecond, corresponds to an experiment of just over 5 minutes long. To visualize the first and last second of the experiment (1000 lines), type:

```
# Visualizing the first 100 lines
image.plot(x = r, y = Time[1:1000], z = Pax[,1:1000],
          xlab = "Pixel", ylab = "Time (s)", main = "First 1000 lines")
image.plot(x = r, y = Time[306201:307200], z = Pax[,306201:307200],
          xlab = "Pixel", ylab = "Time (s)", main = "Last 1000 lines")
```

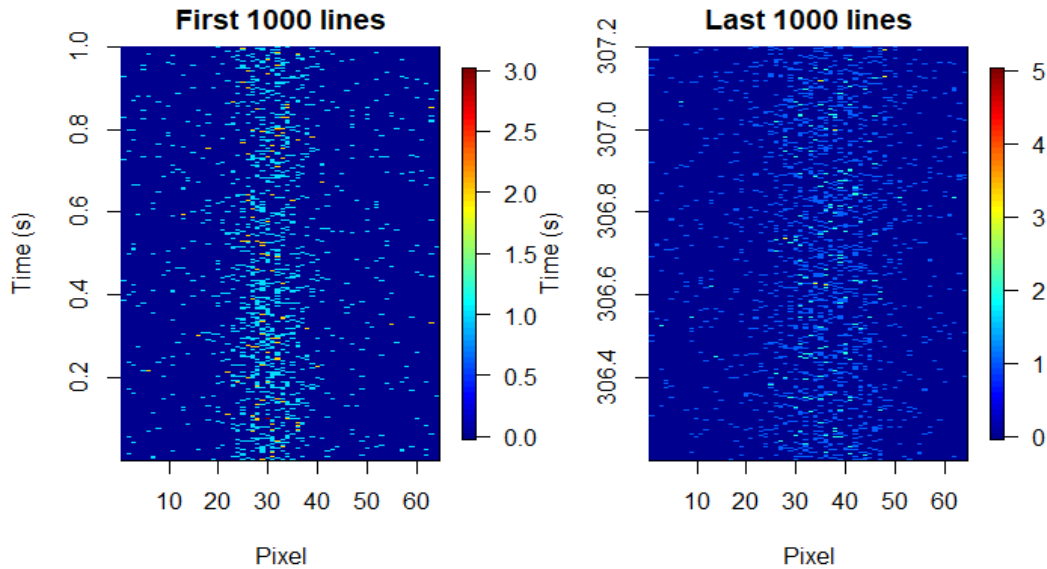


Figure 5.13. Raw line-scan Paxilin-EGFP data.

Given that this experiment is over 5 minutes long, cell and/or organelle movement is very likely to occur at this time scale, so a detrending routine that corrects for this phenomenon is necessary before proceeding to the analysis. To visualize the whole data, type:

```
# Binning the data
Pax.b <- binMatrix(Pax, lineTime = lineTime, nIntervals = 100, columns = 64)
```

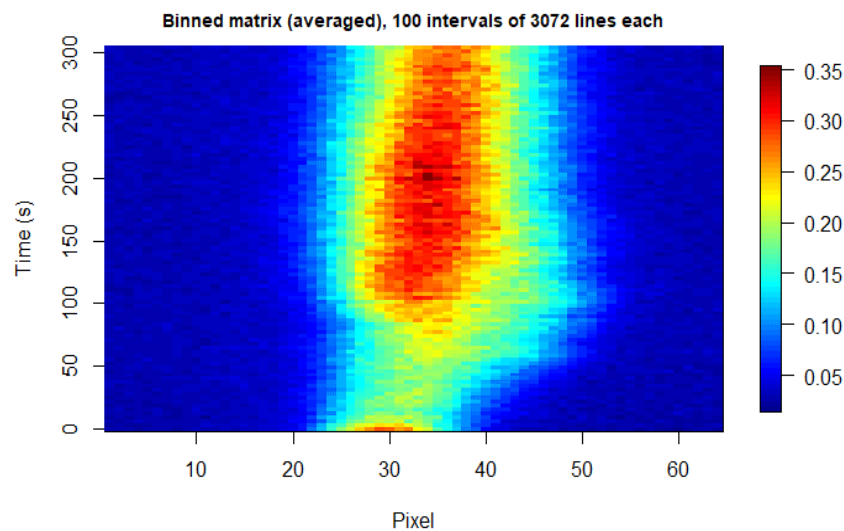


Figure 5.14. Binned line-scan Paxilin-EGFP data. The pseudocolor scale indicates mean photon counts.

To correct for sample movement through polynomial detrending, type:

```
# Detrending the data
det_data <- matrix(0, nrow = dim(Pax)[1], ncol = dim(Pax)[2])
for (i in 1:64){
  det_data[i,] <- detrendTimeSeries(Pax[i,], acqTime = lineTime, nIntervals = 100,
                                   algorithm = "poly", degree = 10, plot = F)
}
det_data.b <- binMatrix(det_data, lineTime = lineTime, nIntervals = 100, columns = 64)
```

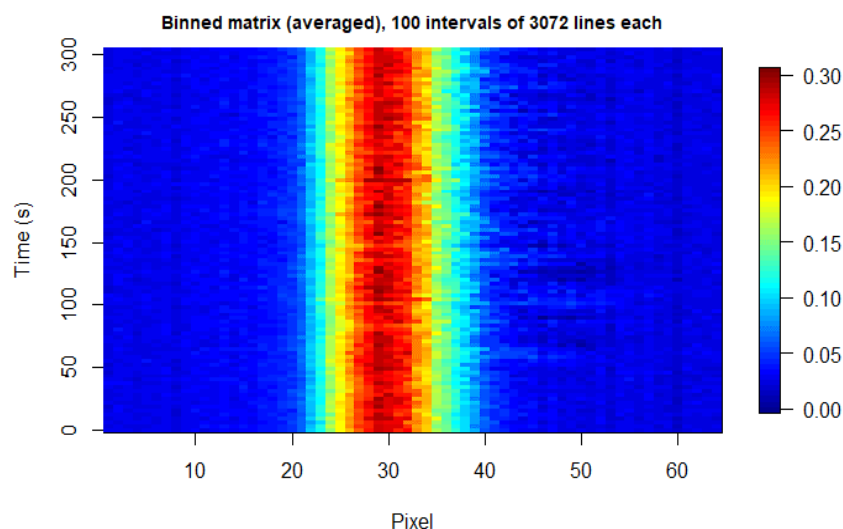


Figure 5.15. Detrended, binned line-scan Paxilin-EGFP data. The pseudocolor scale indicates mean photon counts.

Now that global data trends have been eliminated while preserving the statistics behind the fluorescence fluctuations, one can proceed to the analysis. For simplicity, `Pax` will be overwritten as the detrended data matrix:

```
Pax <- det_data
```

Figures 5.13-15 are graphical representations of fluorescence signal intensity. This kind of graphs is further used throughout the rest of this document to reflect correlation, Number and Brightness magnitude for line-scan experiments; the color bar next to each graph indicates the value for each pixel and is useful for quantitative comparisons. This representation will henceforth be referred to as an *intensity carpet*.

5.2.2 Two-dimensional Autocorrelation Function

Similar to [section 5.1.2](#), the `fcs()` function will be used to build the autocorrelation curves for each of the 64 columns in `Pax` and extract quantitative information about the dynamics of Paxilin-EGFP in the focal adhesion. To calculate the autocorrelation curves of each pixel along the scan line, type:

```
# Calculating the autocorrelation curves
nPoints <- 2^12
tau <- (1:nPoints)*lineTime
Pax.acf <- NULL
for(i in 1:64){
  Pax.acf <- rbind(Pax.acf, fcs(Pax[i,], nPoints = nPoints))
}

# Graphing the result
image.plot(x = r, y = log10(tau), z = Pax.acf, zlim = c(-1,3),
           xlab = "Pixel", ylab = expression(tau(s)), main = "ACF carpet")
```

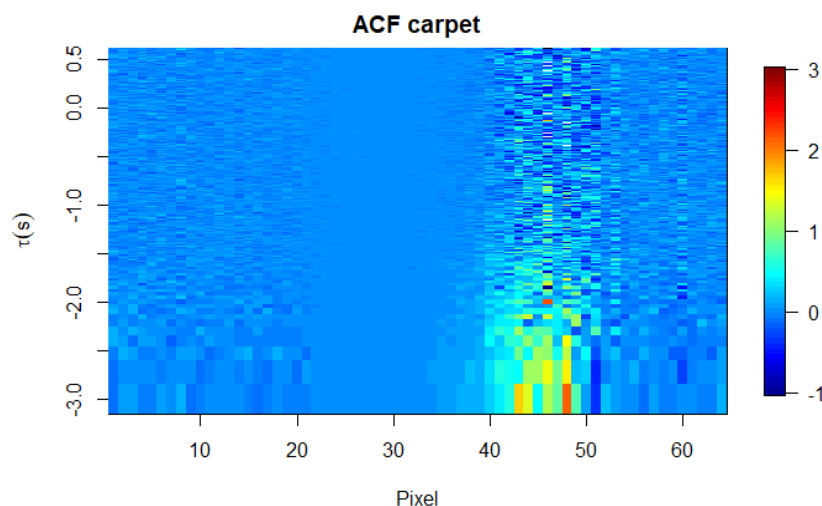


Figure 5.16. ACF carpet of the line-scan Paxilin-EGFP data. The pseudocolor scale indicates correlation ($G(\tau)$).

5.2.2.1 Simplifying the ACF carpet

Like [section 5.1.3.1](#), the `simplifyFCS()` function can help prevent overweighting of the fitting process over the correlation curves in the region of near-to-zero correlation by averaging $G(\tau)$ at longer τ values. To simplify the curves in the ACF carpet, type:

```
# Simplifying the autocorrelation carpet
Pax.acf.s <- NULL
for(i in 1:64){
  acf.s <- simplifyFCS(Pax.acf[i,], tau)
  Pax.acf.s <- rbind(Pax.acf.s, acf.s$g)
}

# Graphing the result
image.plot(x = r, y = log10(acf.s$tau), z = Pax.acf.s, zlim = c(-0.5,3),
           xlab = "Pixel", ylab = expression(tau(s)),
           main = "Simplified ACF carpet")
```

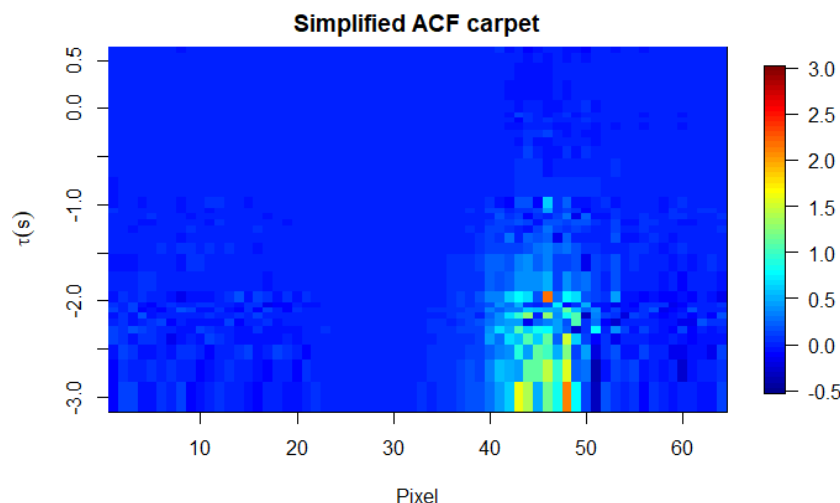


Figure 5.17. Simplified ACF carpet in figure 5.16.

In order to better visualize the transition of the autocorrelation function curves across the scan line, let us use the `smoothCarpet()` function to reduce the contrast between the values of each pixel in the ACF carpets. This function takes advantage of the *smoothing splines* method to smooth the vertical and horizontal axes of a matrix, where the parameters `dfV` and `dfH` are the degrees of freedom for the vertical and horizontal axes, respectively. The more degrees of freedom the less severe the smoothing process will be and vice versa. Given that this process modifies the information in the carpet, it should be only used for qualitative purposes. Type:

```
# Smoothing the autocorrelation carpet
Pax.acf.sm <- smoothCarpet(img = Pax.acf, dfH = 20, dfV = 20)

# Graphing the result
image.plot(x = r, y = log10(tau), z = Pax.acf.sm,
           xlab = "Pixel", ylab = expression(tau(s)),
           main = "Smoothed ACF carpet")
```

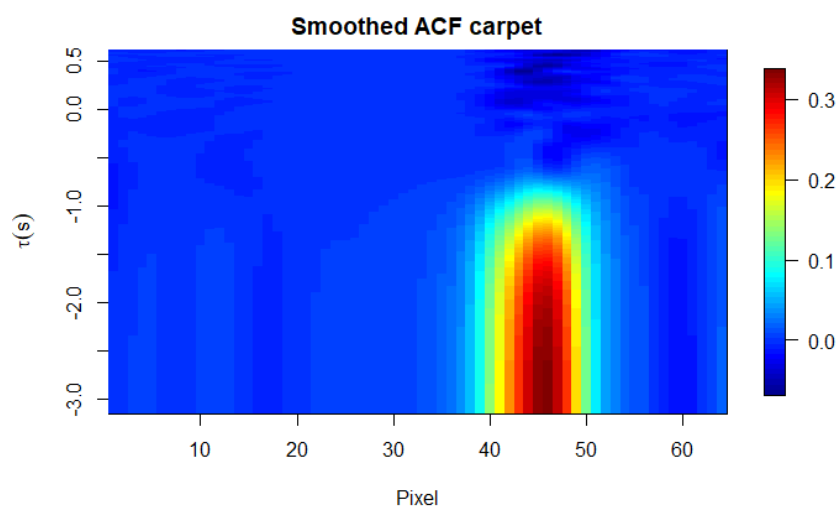


Figure 5.18. Smoothed ACF carpet in figure 5.16. 20 degrees of freedom were used for both the vertical and horizontal axes were used.

5.2.2.2 ACF carpet analysis

Based on the smoothed ACF carpet in figure 5.18, two regions with distinct correlation properties can be identified (pixels 1 to 39 and pixels 40 to 50). In order to extract quantitative information about the dynamics in each of these regions, one can average all the autocorrelation curves encompassed in those pixels and fit a physical model to the resulting curve.

The `norm.vector()` function uses the unity-based normalization method to bring all the values of a vector into the range of zero to one. Unnormalized curves serve as an indicative of concentration ($G(0)$), while the normalized ones become useful to compare mobility (see figure 5.10).

To obtain the average autocorrelation curves of a certain pixel range, type:

```
# Averaging multiple columns
acf.1 <- apply(Pax.acf.s[1:39,], MARGIN = 2, mean)
acf.2 <- apply(Pax.acf.s[40:50,], MARGIN = 2, mean)

# Normalizing the autocorrelation curves
n.acf.1 <- norm.vector(acf.1)
n.acf.2 <- norm.vector(acf.2)

# Graphing the results
par(mfrow = c(1,2))
plot (acf.1~acf.s$tau, log = "x", type = "l",
      xlab = expression(tau(s)), ylab = expression(G(tau)), cex.main = 0.8,
      main = "Averaged autocorrelation curves", col = "blue3")
lines(acf.1~acf.s$tau, col = "red3")
plot (n.acf.1~acf.s$tau, log = "x", type = "l",
      xlab = expression(tau(s)), ylab = expression(G(tau)), cex.main = 0.8,
      main = "Normalized, averaged autocorrelation curves", col = "blue3")
lines(n.acf.1~acf.s$tau, col = "red3")
```

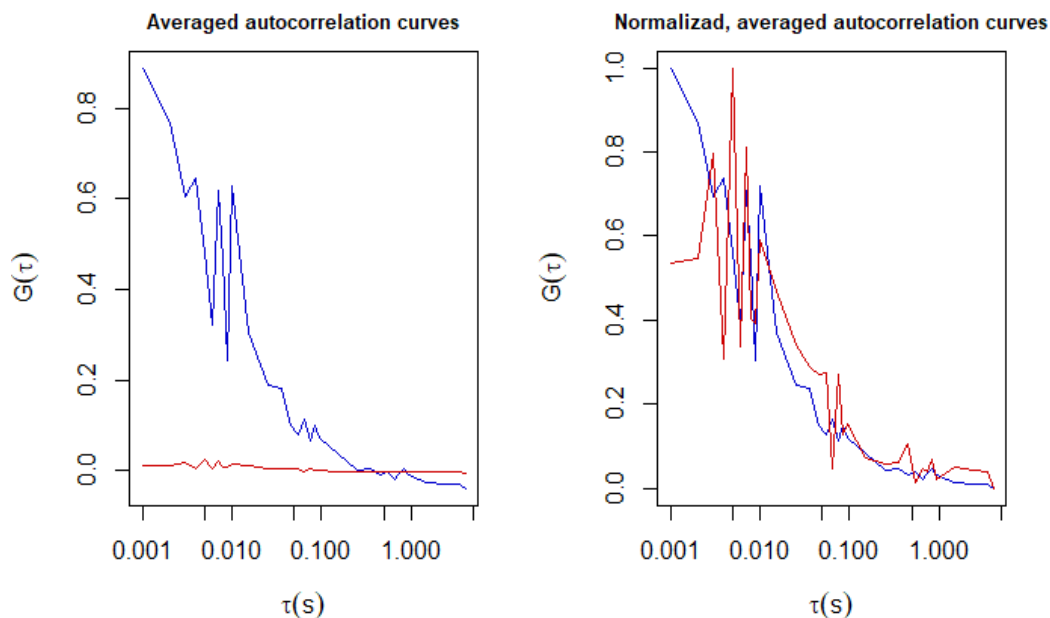


Figure 5.19. Averaged (left) and normalized averaged (right) autocorrelation curves of the regions encompassed by pixels 1-39 (red) and 40-50 (blue) in figure 5.16, respectively.

Outside columns 40 to 50, the averaged curves (red) are very unstable. The reason is that there is no adhesion in this region, and only noise is being analyzed.

Now, for the fitting process, only the region with high correlation (pixels 40-50) will be analyzed. Type:

```
# Setting up the fitting parameters
G <- data.frame(tau = acf.s$tau, g = acf.2, s = 0.25, k = 4.73)
start <- list(D = 0.1, G0 = 1, c = 0)
low <- list(D = 0.001, G0 = 0.01, c = -1)
up <- list(D = 10, G0 = 10, c = 1)

# Fitting results
G_fit <- fitFCS(data = G, start = start, low = low, up = up, type = "D3D", trace = F)
D <- signif(summary(G_fit)$coefficients[1,1], 4)

# Graphing the fitted data
fit_curve <- predict(G_fit, acf.s$tau)
plot(acf.2~acf.s$tau, type = "l", log = "x", main = "Fitted ACF",
     xlab = expression(tau(s)), ylab = expression(G(tau)))
lines(fit_curve~acf.s$tau, col = "blue", lwd = 2, lty = 2)
legend("topright", legend = c("Raw", "Fit"), col = c("black", "blue"),
     lty = c(1, 2), lwd = 2, cex = 1)
legend("right", legend = paste("D =", D, "um2/s"), cex = 0.8)
par(mfrow = c(2, 1))
plot(G_fit$m$resid~acf.s$tau, log = "x", type = "l",
     xlab = expression(tau(s)), ylab = "Residuals")
abline(h = 0, col = "red")
```

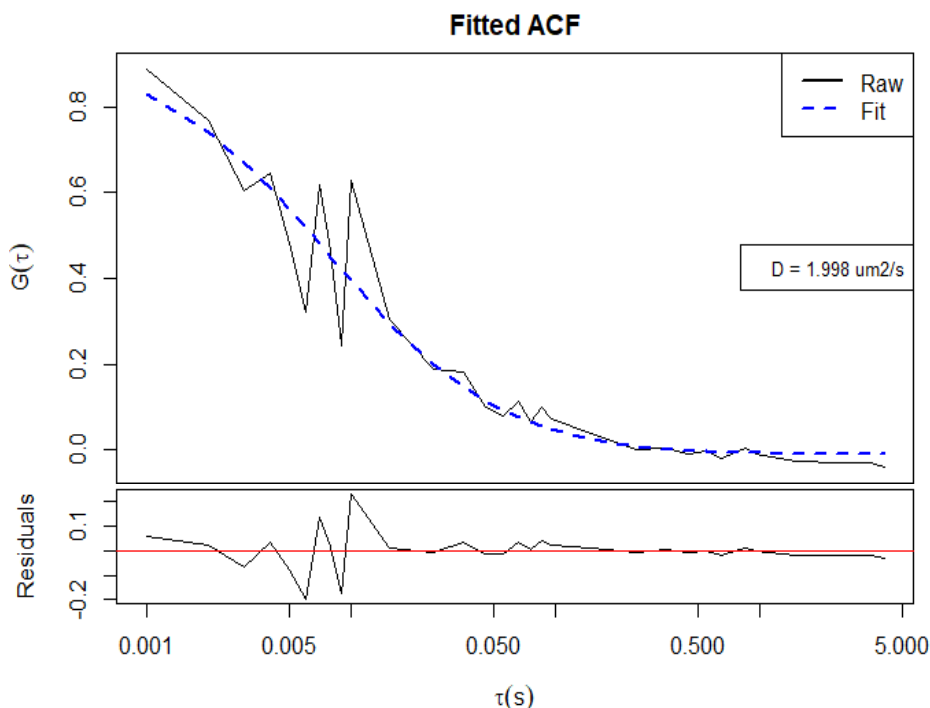


Figure 5.20. Averaged (left) and normalized averaged (right) autocorrelation curves of the regions encompassed by pixels 1-39 (red) and 40-50 (blue) in figure 5.16, respectively.

The analysis yielded a diffusion coefficient of $2 \mu\text{m}^2/\text{s}$, which is very similar to the reported value [9].

5.2.3 Pair Correlation Function

The Pair Correlation approach analyzes the data of a raster line to extract information about the trajectory of single particles moving across a linear region, i.e. it measures the time it takes for a particle to move from one point to another. In this method, the temporal cross-correlation function between different points in space (such as two distant pixels across the scan line) is computed. Hence, it allows to describe the average path of a certain molecule within the cell structures, creating a map of molecular flux with a spatial resolution provided by the focal length of the imaging volume [10]. The pair correlation between two points separated by time and space is given by

$$G(\tau, r) = \frac{\langle F(t, r) * F(t + \tau, r + \delta r) \rangle}{\langle F(t, r) \rangle * \langle F(t, r + \delta r) \rangle} - 1$$

Where $F(t, r)$ is the signal intensity at a given acquisition time t and at a r position across the scan line, δr represents the spatial interval (i.e. distance between pixels) at which the cross-correlation is calculated.

Here, the `pcf()` function is introduced, which compares the fluorescence intensity of two points separated by a given time and distance, correlating their values and creating a molecular flux map of single molecules.

The same mathematical principle is used by both the ACF and pCF methods. In the case of ACF, each pixel is correlated with itself, separated only by a certain time window, thus the parameter `dr` is set to zero. For pCF, on the other hand, `dr` is set to a positive integer, which represents the correlation distance (pixels).

When performing pCF, choosing different values for `dr` can provide the user with different information, depending on the nature of the phenomena in study and, more specifically, the size of the barriers/obstacles to diffusion. Thus, further adjusting `dr` will provide the means to establish molecular mobility across different cellular compartments of variable dimensions.

5.2.3.1 Data exploration

As an example, the mobility of a mixture of dimers and hexamers of Venus, an enhanced yellow fluorescent protein, and their ability to transit through the nuclear membrane in live cells will be analyzed. For this experiment, transfected cells were imaged using a confocal microscope with a 60X water immersion objective, 1.2 NA. Excitation was provided by a 488 nm laser at 0.1% power. Line scans were performed across the nuclear envelope with a 64-pixel line and a pixel size of 50 nm. Fluorescence intensity data was collected using the photon-counting mode, with a 12.5 μ s pixel dwell time, and line scan time of 1.925 ms. Please refer to [section 1.4](#) of this document to discover where to find this sample data.

First, import the data using the `readFileTiff()` function:

```
# Importing the data
V2V6 <- readFileTiff("V2V6.tif")[,1]
class(V2V6); dim(V2V6)

## [1] "matrix" "array"
## [1]      64 32766
```

To define the temporal `Time` and spatial `r` ranges of the experiment, type:

```
# Defining experimental parameters
lineTime = 0.001925
pixelSize = 0.05
Time <- (1:dim(V2V6)[2])*lineTime
r <- (1:dim(V2V6)[1])*pixelSize
```

In this case, `V2V6` is a matrix with 64 columns (pixels) and 32766 rows (line scans) which, considering a line scan time of 1.925 ms, corresponds to an experiment of about a minute long. To visualize the first 100 lines (192.5 ms), type:

```
# Visualizing the first and last 100 Lines
par(mfrow = c(1,2))
image.plot(x = r, y = Time[1:100], z = V2V6[,1:100], legend.mar = 10,
           xlab = expression(Distance(mu*m)), ylab = "Time (s)", main = "First 100 lines")
image.plot(x = r, y = Time[32667:32766], z = V2V6[,32667:32766],
           xlab = expression(Distance(mu*m)), ylab = "Time (s)", main = "Last 100 lines")
```

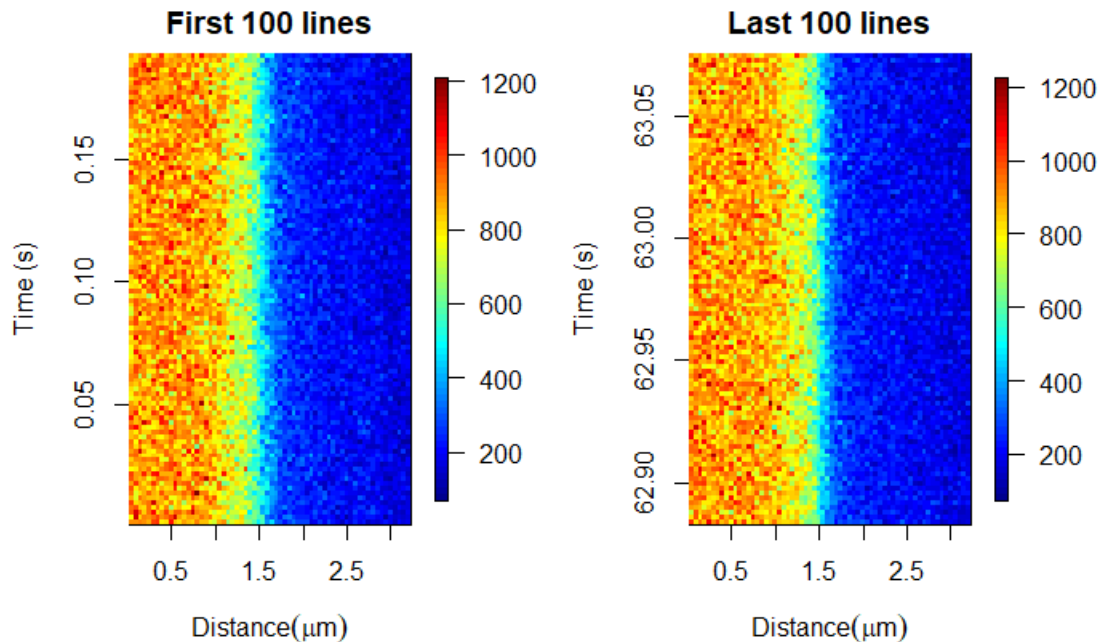


Figure 5.21. Raw line-scan Venus data.

Note that this experiment is about 63 seconds long. Cell and/or organelle movement is very likely to occur at this time scale, so a detrending routine that corrects for this phenomenon is necessary before proceeding to the analysis. To visualize the whole data, type:

```
# Binning the data
V2V6.b <- binMatrix(V2V6, lineTime = lineTime, nIntervals = 100, columns = 64)
```

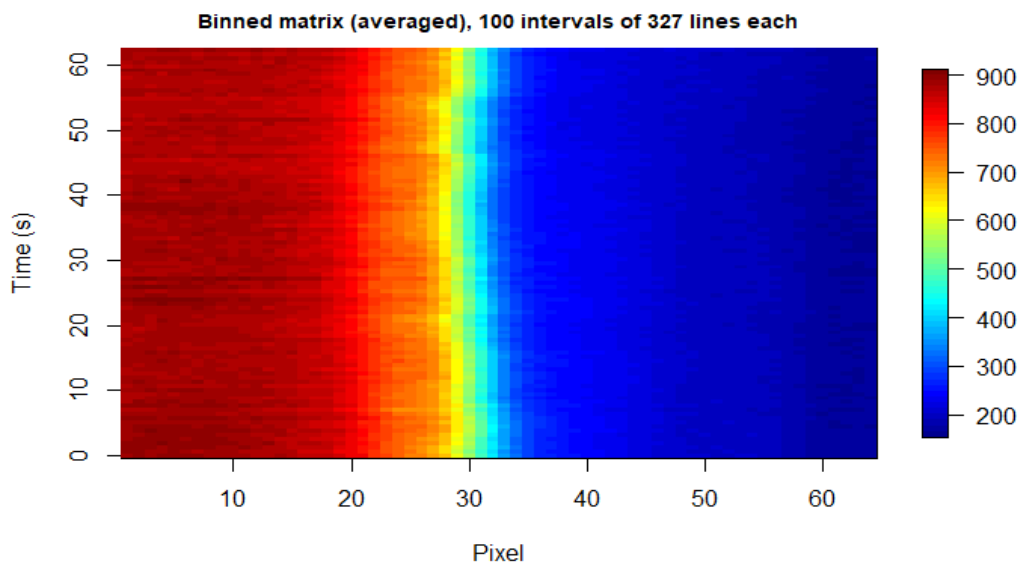


Figure 5.22. Binned line-scan Venus data.

The nucleus envelope, whose position is located approximately at pixel 30 in figure 5.22, seems to move slightly to the left throughout the experiment. To correct for this issue through polynomial detrending, type:

```
# Detrending the data
det_data <- matrix(0, nrow = dim(V2V6)[1], ncol = dim(V2V6)[2])
for (i in 1:64){
  det_data[i,] <- detrendTimeSeries(V2V6[i,], acqTime = lineTime, nIntervals = 100,
    algorithm = "poly", degree = 10, plot = F)
}
det_data.b <- binMatrix(det_data, lineTime, 500, 64)
```

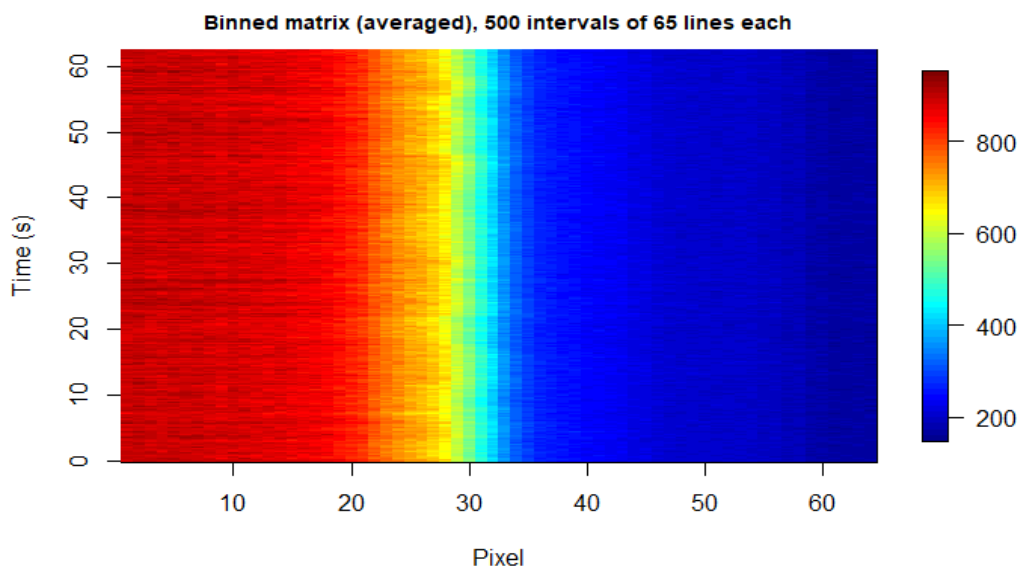


Figure 5.23. Detrended, binned line-scan Venus data.

Now that global data trends have been eliminated, one can proceed to the analysis. For simplicity, `V2V6` will be overwritten as the detrended data matrix:

```
V2V6 <- det_data
```

5.2.3.2 Building the pCF carpet

To calculate the spatial cross-correlation along the line scan using the `pcf()` function, type:

```
# Creating the pCF carpet
dr <- 5
nPoints <- 2^12
tau <- (1:nPoints)*lineTime
V2V6.pcf <- pcf(img = V2V6, nPoints = nPoints, dr = dr)

# Simplifying and smoothing the pCF carpet
V2V6.pcf.s <- NULL
for(i in 1:(dim(V2V6)[1]-dr)){
  pcf.s <- simplifyFCS(V2V6.pcf[i,], tau)
  V2V6.pcf.s <- rbind(V2V6.pcf.s, pcf.s$g)
}
V2V6.pcf.sm <- smoothCarpet(V2V6.pcf.s, dfV = 15, dfH = 15)

# Graphing the results
par(mfrow = c(1,2))
image.plot(x = r[1:dim(V2V6.pcf.s)[1]], y = log10(pcf.s$tau), z = V2V6.pcf.s, legend.mar = 10,
  xlab = expression(r(mu*m)), ylab = expression(tau(s)),
  main = "Simplified ACF carpet", zlim = c(-0.001,0.005))
image.plot(x = r[1:dim(V2V6.pcf.s)[1]], y = log10(pcf.s$tau), z = V2V6.pcf.sm,
  xlab = expression(r(mu*m)), ylab = expression(tau(s)),
  main = "Smoothed ACF carpet")
```

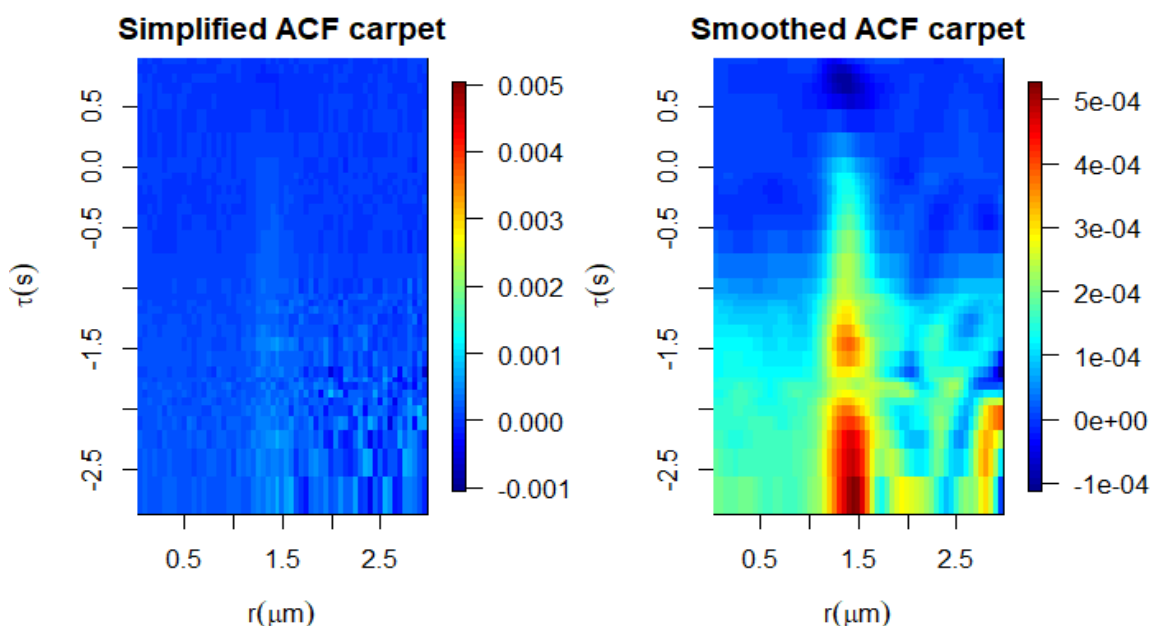


Figure 5.24. Simplified (left) and smoothed (right) pCF carpet. 15 degrees of freedom were used for both the vertical and horizontal axes were used

When using the pCF approach, the resulting matrix will be dr columns shorter, as no more columns are available to correlate when column $N - dr$ has been reached. The 'x' axis represents the physical space scanned in the cell where correlation is computed against a consecutive space $r + \delta r$. The 'y' axis represents the correlation time τ and the pseudocolor scale indicates the magnitude of $G(\tau, \delta r)$.

The presence of high correlation values at the boundaries of the nuclear envelope (approximately at the 1.5 μm mark in figure 5.24) indicates a reduced mobility for both dimers and hexamers of Venus. Such behavior in the pCF data is an indicative of a barrier to diffusion.

To analyze the directional mobility of Venus oligomers in a region between the cytoplasm and the nuclear envelope, type:

```
# Calculating the z-colors for the perspective plot
compPerpsZval <- function(z){
  color <- tim.colors(64)
  nrz <- nrow(z)
  ncz <- ncol(z)
  zfacet <- z[-1, -1] + z[-1, -ncz] + z[-nrz, -1] + z[-nrz, -ncz]
  facetcol <- cut(zfacet, 64)
  return(color[facetcol])
}
zcolor <- compPerpsZval(V2V6.pcf.sm[1:36,])

# Graphing the result
par(mfrow = c(1,2))
image.plot(x = r[1:36], y = log10(pcf.s$tau), z = V2V6.pcf.sm[1:36,],
  xlab = expression(r(mu*m)), ylab = expression(tau(s)),
  main = paste("pCF carpet (cyt-nuc), dr =", dr))
abline(v = c(1.2, 1.7) - lineTime*dr, lty = 2, lwd = 2)
persp(V2V6.pcf.sm[1:36,], log = "y", xlab = "r", ylab = "tau", zlab = "G",
  phi = 30, theta = -30, col = zcolor)
```

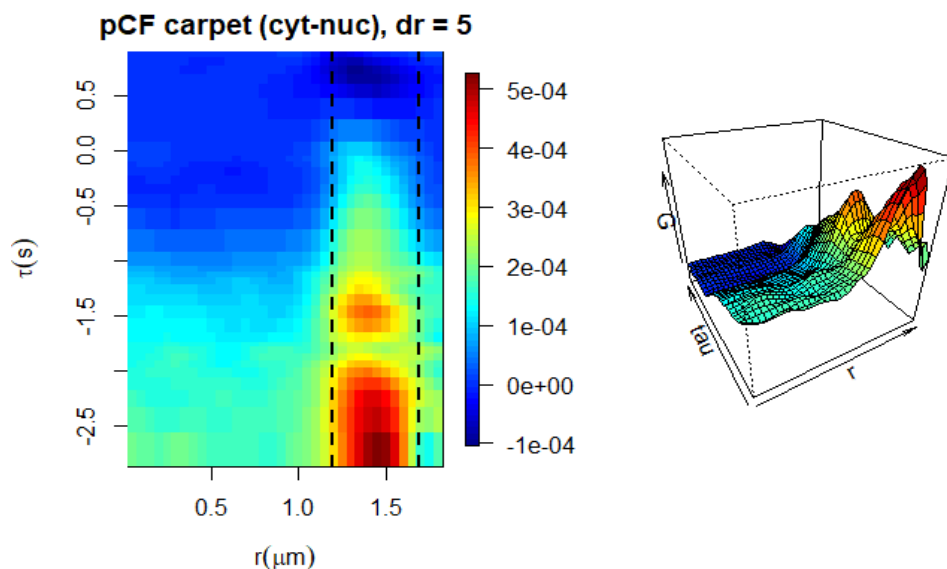


Figure 5.25. Cytoplasm and nucleus (delimited by dotted lines) regions of the pCF carpet in figure 5.24 (left) and its perspective plot (right).

Here is another example with a `dr` value of 10. Type:

```
# Creating the pCF carpet
dr <- 10
V2V6.pcf <- pcf(img = V2V6, nPoints = nPoints, dr = dr)

# Simplifying and smoothing the pCF carpet
V2V6.pcf.s <- NULL
for(i in 1:(dim(V2V6)[1]-dr)){
  pcf.s <- simplifyFCS(V2V6.pcf[i,], tau)
  V2V6.pcf.s <- rbind(V2V6.pcf.s, pcf.s$g)
}
V2V6.pcf.sm <- smoothCarpet(V2V6.pcf.s, dfV = 15, dfH = 15)
zcolor <- compPerpsZval(V2V6.pcf.sm[1:36,])

# Graphing the result
par(mfrow = c(1,2))
image.plot(x = r[1:36], y = log10(pcf.s$tau), z = V2V6.pcf.sm[1:36,],
  xlab = expression(r(mu*m)), ylab = expression(tau(s)),
  main = paste("pCF carpet (cyt-nuc), dr =", dr))
persp(V2V6.pcf.sm[1:36,], log = "y", xlab = "r", ylab = "tau", zlab = "G",
  phi = 30, theta = -30, col = zcolor)
```

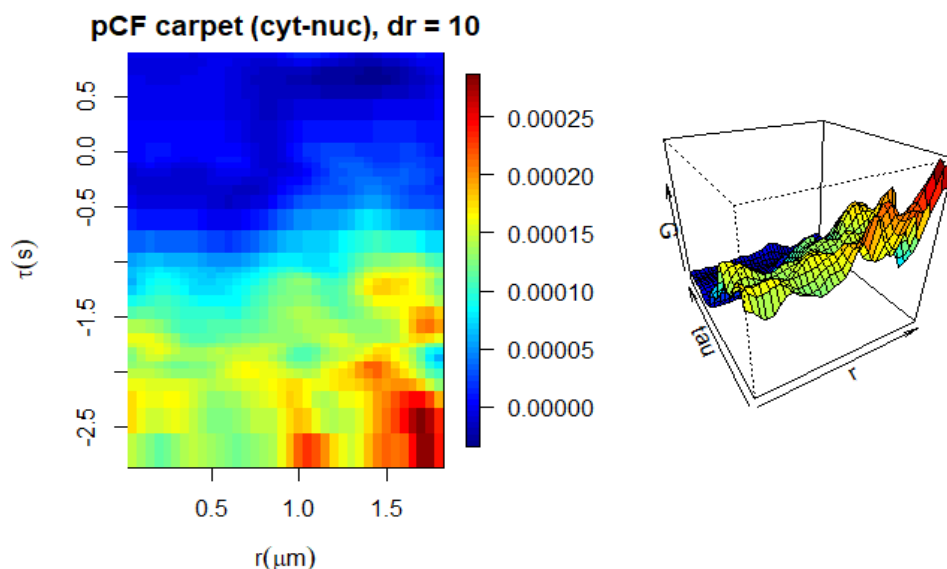


Figure 5.26. Cytoplasm and nucleus regions of the pCF carpet in figure 5.24 (left) and its perspective plot (right).

A molecule observed at a given time t and location r can be found at a distance $r + \delta r$ with a probability that is proportional to the fluorescence intensity at that given distance. Only the same molecule can produce, in average, positive cross-correlation with a given delay time τ at two different points: r and $r + \delta r$.

5.2.3.3 Fixed-column pCF

The `pcf()` function allows to generate the pCF carpet of the pair correlation of a user specified column against all the others. This is particularly useful for comparing the mobility of molecules present in each region with the rest of the spatial domain. In this case, the parameter `one.col` is set to `TRUE` and `dr` is now the column to fix. Type:

```
# Creating the fixed-column pCF carpet
nPoints <- 2^12
tau <- (1:nPoints)*lineTime
V2V6.pcf.fx <- pcf(img = V2V6, nPoints = nPoints, one.col = T, dr = 1)

# Simplifying and smoothing the pCF carpet
V2V6.pcf.fx.s <- NULL
for(i in 1:(dim(V2V6)[1])){
  pcf.s <- simplifyFCS(V2V6.pcf.fx[i,], tau)
  V2V6.pcf.fx.s <- rbind(V2V6.pcf.fx.s, pcf.s$g)
}
V2V6.pcf.fx.sm <- smoothCarpet(V2V6.pcf.fx.s, dfV = 15, dfH = 15)
zcolor <- compPerpsZval(V2V6.pcf.fx.sm)

# Graphing the result
par(mfrow = c(1,2))
image.plot(x = r, y = log10(pcf.s$tau), z = V2V6.pcf.fx.sm,
           xlab = expression(r(mu*m)), ylab = expression(tau(s)),
           main = "Fixed column (1) pCF carpet")
persp(V2V6.pcf.fx.sm, log = "y", xlab = "r", ylab = "tau", zlab = "G",
       phi = 30, theta = -30, col = zcolor)
```

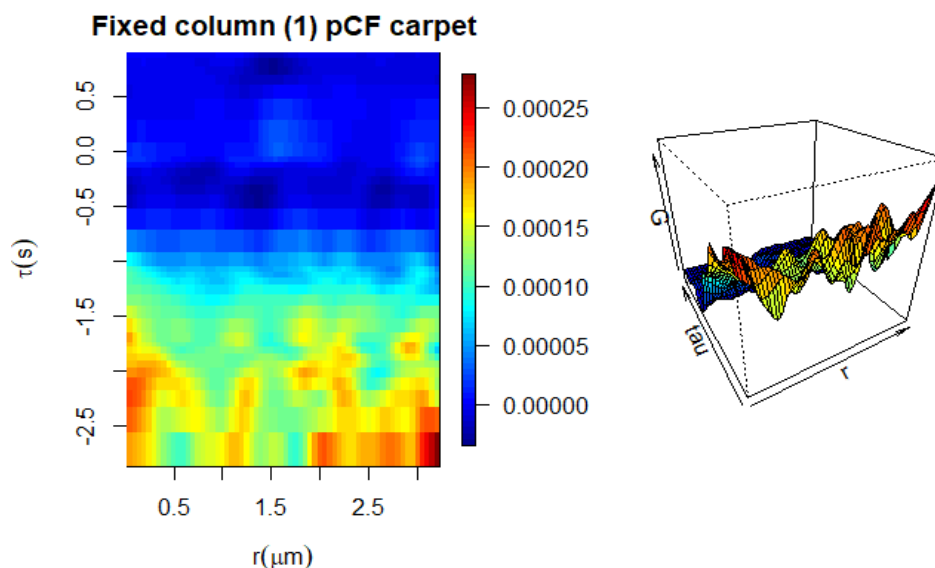


Figure 5.27. Fixed-column pCF carpet for pixel 1 (left) and its perspective plot (right).

Whenever a positive correlation value appears in the fixed-column pCF carpet, is an indicative that, throughout the experiment, a molecule originally present in pixel 1 (or whichever the fixed column is) was able to travel a distance given by the amount of pixels ('x' axis) in a time given by the correlation time τ ('y' axis).

6. FLUORESCENCE FLUCTUATION SPECTROSCOPY

FCS measures particle concentration and mobility based on the magnitude of the fluorescence signal correlation over increasingly larger temporal intervals. Though, FCS is strictly dependent on very high temporal resolution (i.e. 10^{-3} to 10^{-9} seconds). In other words, molecular process' dynamic information recovery through FCS is only possible if data acquisition speed is faster than the process itself.

It is assumed that fluorescence fluctuations are mostly a result of molecular diffusion. The latter is, in turn, influenced by other factors such as molecular interaction, complex formation, polymerization processes, variations in molecular weight and chemical structure, among others.

Fluorescence Fluctuation Spectroscopy (FFS) does not focus on the temporality of the fluorescence signal fluctuations but rather on the statistics behind them. Furthermore, unlike FCS, FFS does not require a nonlinear fit of the data; the average number of particles and their brightness can be obtained directly through moment analysis of the fluorescence intensity. In this scenario, the fluorescence intensity profile in a certain pixel across time is taken as a distribution of given mean and variance. These two parameters (formerly, first and second moments), can be used to study the nature of diffusing particles and their effect on signal intensity fluctuations.

- ✓ *In this section, a brief introduction to the Number and Brightness method is provided. The utility of the functions in FCSlib to measure particle relative concentration and stoichiometry based on the moment analysis of fluorescence fluctuations is put to the test. Moreover, the Pair Correlation of Molecular Brightness (pCOMB) method theory, as well as a demonstration of its capability to track different oligomeric species' mobility, are included.*

6.1 Number and Brightness

Originally proposed by Qian and Elson in 1990 [11,12] to study molecular aggregation, the Number and Brightness (N&B) method is a time-independent technique that provides an estimate of molecular concentration and aggregation state (or stoichiometry), based on the statistical moments of the fluorescence intensity fluctuations. When applied on line-scan data and image analysis, N&B allows to map the spatial distribution of different oligomeric states of a molecule present in the sample (Brightness) and their relative abundance (Number).

For each pixel i in an image or a scan line acquired in the microscope across time with K total acquisitions, the signal intensity k apparent mean value is given by

$$\langle k_i \rangle = \frac{\sum_i k_i}{K}$$

The apparent variance of these fluctuations in each pixel is

$$\sigma_i^2 = \frac{\sum_i (k_i - \langle k_i \rangle)^2}{K}$$

The idea behind the N&B approach is that, the greater the number of particles present in the sample, the greater the mean fluorescence signal intensity will be across time. Additionally, the brighter these particles are, the greater the magnitude of the fluorescence fluctuations around the mean value will be. Based on that, the number of molecules is proportional to the mean intensity, while their brightness (often translated as oligomerization state) is proportional to the variance. Formally, the apparent Number (N) and apparent Brightness (B) are defined as

$$N = \frac{\langle k \rangle^2}{\sigma^2} \quad B = \frac{\sigma^2}{\langle k \rangle}$$

While data acquisition speed does not play a crucial role in this technique, if fast enough, it is possible to study aggregation dynamics (see [section 6.1.3](#)); it also provides a means for excluding shot noise. Although resolution of different oligomeric species within the same pixel is limited by the optical system's spatial resolution, the average particle number and brightness for each pixel-sized area in the sample is calculated.

So far, it has been assumed that the only factor contributing to $\langle k_i \rangle$ and σ^2 is molecular traffic (i.e. molecules entering and leaving the observation volume). Nonetheless, other instrumentation-related factors must be considered prior to a quantitative analysis. In theory, signal intensity should be zero when no photons are present, however, this isn't usually the case. The *offset*, which is a constant quantity that depends on the detector configuration, corresponds to the average signal intensity across all pixels when the microscope's detector is not exposed to any source of light. In order to obtain the real mean value of the fluorescence intensity signal $\langle K \rangle$, it is necessary to subtract the *offset* from the data.

$$\langle K \rangle = \langle k \rangle - \text{offset}$$

Additionally, the proportionality factor S , must be calculated. This factor is defined as the ratio between the number of photons received and the number of electrons (signal) reported by the detector. The real mean value of fluorescence intensity $\langle K \rangle$ is then the product of the actual number of fluorophores inside the observation volume (n) and their actual molecular brightness (ε), both weighted by the proportionality factor S in

$$\langle K \rangle = S\varepsilon n$$

Combining the previous equations, we obtain

$$\langle k \rangle = S\varepsilon n + \text{offset}$$

Moreover, the apparent variance σ^2 is the result of the sum of variance due to the fluorescence fluctuations (σ_n^2) and the variance due to the detector noise (σ_d^2), as in

$$\sigma^2 = \sigma_n^2 + \sigma_d^2$$

While σ_n^2 depends on the proportionality factor S and the real number (n) and brightness (ε) of the molecules, σ_n^2 additionally depends on the detector readout noise (σ_0^2)

$$\sigma_n^2 = S^2 \varepsilon^2 n$$

$$\sigma_d^2 = S^2 \varepsilon n + \sigma_0^2$$

which, in terms of the apparent variance σ^2 , can be expressed as

$$\sigma^2 = S^2 \varepsilon^2 n + S^2 \varepsilon n + \sigma_0^2$$

It is then necessary to know σ_0^2 , which can be obtained by calculating the variance of the signal intensity when the detector is not exposed to any light sources. Altogether, considering these instrumentation-related adjustments, the Number and Brightness equations can be rewritten as

$$N = \frac{S\varepsilon n}{S\varepsilon + n} \quad B = S\varepsilon + S$$

or

$$N = \frac{\varepsilon n}{\varepsilon + 1} \quad B = S(\varepsilon + 1)$$

Note that, while n depends on ε , ε does not depend on n . By performing the proper substitutions, the real number and brightness equations can be expressed in terms of the S , *offset* and σ_0^2 factors:

$$n = \frac{(\langle k \rangle - offset)^2}{\sigma^2 - \sigma_0^2 - S(\langle k \rangle - offset)} \quad \varepsilon = \frac{\sigma^2 - \sigma_0^2 - S(\langle k \rangle - offset)}{S(\langle k \rangle - offset)}$$

In order to calculate the S factor, one can measure the detector variance in the absence of signal fluctuations (i.e. by acquiring a time series while exposing the detector to a light source of constant intensity). Considering that $B = S(\varepsilon + 1)$, and that, in the absence of fluorophores, the real brightness ε is zero, the apparent brightness B is reduced to the S factor:

$$B = S = \frac{\sigma^2 - \sigma_0^2}{\langle k \rangle - offset}$$

By isolating σ^2 , we obtain an equation of the form $y = mx + b$, where the slope is the S factor and the intercept is σ_0^2 :

$$\sigma^2 = S(\langle k \rangle - offset) + \sigma_0^2$$

6.1.1 Data exploration

As an example, the abundance of dimers of Venus and their aggregation state in live cells will be analyzed. This experiment was performed as described in [section 5.2.3.1](#). Type:

First, import the data using the `readFileTiff()` function:

```
# Importing the data
V2 <- readFileTiff("V2.tif")[,1]
class(V2); dim(V2)

## [1] "matrix" "array"
## [1] 64 32766
```

To define the temporal `Time` and spatial `r` ranges of the experiment, type:

```
# Defining experimental parameters
lineTime = 0.001925
pixelSize = 0.05
Time <- (1:dim(V2)[2])*lineTime
r <- (1:dim(V2)[1])*pixelSize
```

In this case, `V2` is a matrix with 64 columns (pixels) and 32766 rows (line scans) which, considering a line scan time of 1.925 ms, corresponds to an experiment of about a minute long. To visualize the first 100 lines (192.5 ms), type:

```
# Visualizing the first and last 100 Lines
par(mfrow = c(1,2))
image.plot(x = r, y = Time[1:100], z = V2[,1:100], legend.mar = 10,
           xlab = expression(Distance(mu*m)), ylab = "Time (s)", main = "First 100 lines")
image.plot(x = r, y = Time[32667:32766], z = V2[,32667:32766],
           xlab = expression(Distance(mu*m)), ylab = "Time (s)", main = "Last 100 lines")
```

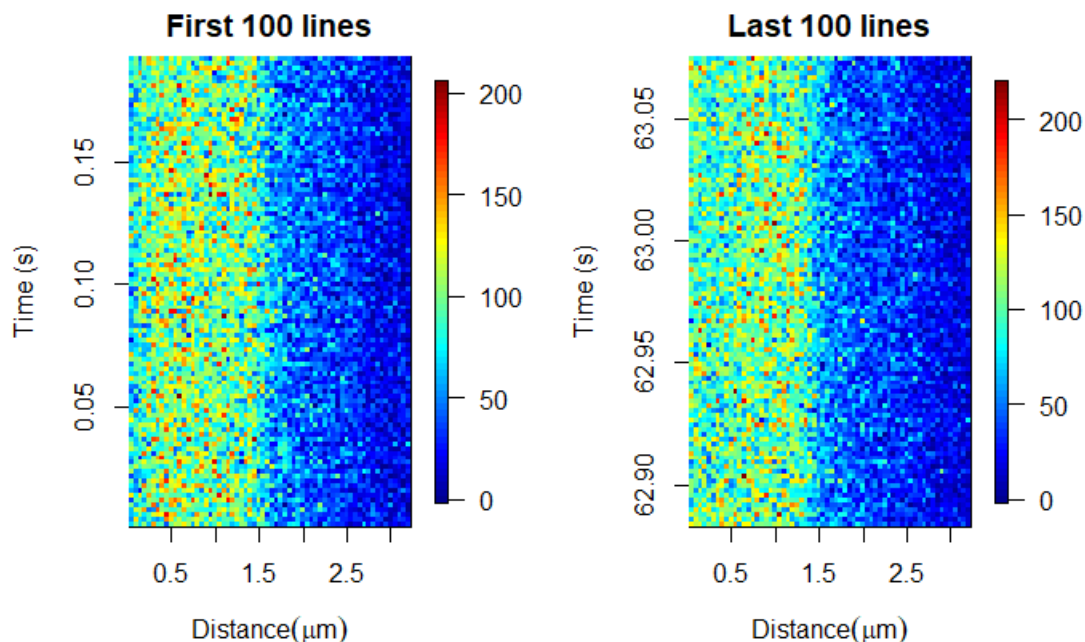


Figure 6.1. Raw line-scan Venus data.

Note that this experiment is about 63 seconds long. Cell and/or organelle movement is very likely to occur at this time scale, so a detrending routine that corrects for this phenomenon is necessary before proceeding to the analysis. To visualize the whole data, type:

```
# Binning the data
V2.b <- binMatrix(V2, lineTime = lineTime, nIntervals = 100, columns = 64)
```

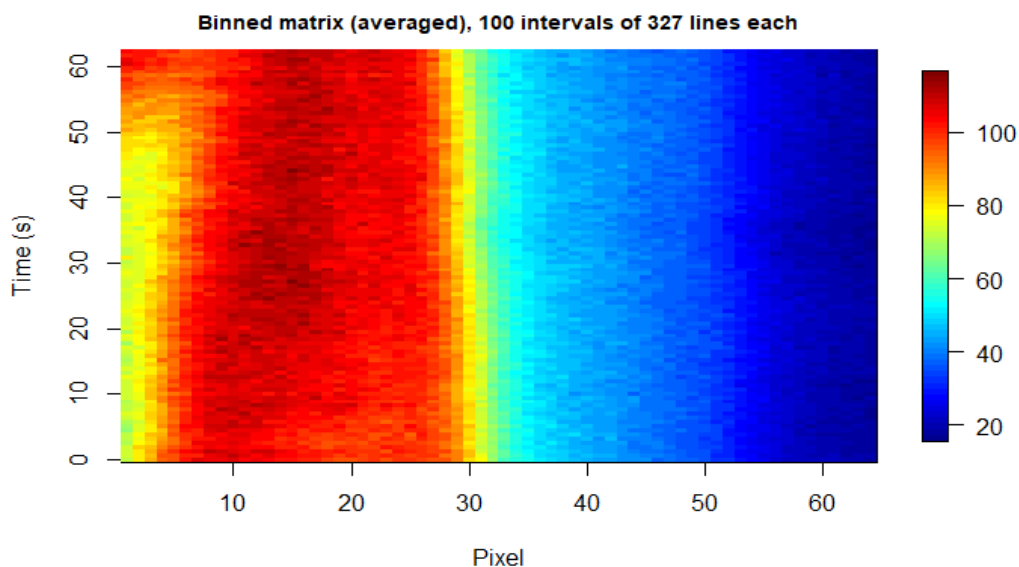


Figure 6.2. Binned line-scan Venus data.

The nucleus envelope, whose position is located approximately at pixel 30 in figure 6.2, seems to move slightly to the left throughout the experiment. Also, possible organelle movement is observed in pixels 1-10. To correct for these issues through polynomial detrending, type:

```
# Detrending the data
det_data <- matrix(0, nrow = dim(V2)[1], ncol = dim(V2)[2])
for (i in 1:64){
  det_data[i,] <- detrendTimeSeries(V2[i,], acqTime = lineTime, nIntervals = 100,
                                   algorithm = "poly", degree = 10, plot = F)
}
det_data.b <- binMatrix(det_data, lineTime, 500, 64)
```

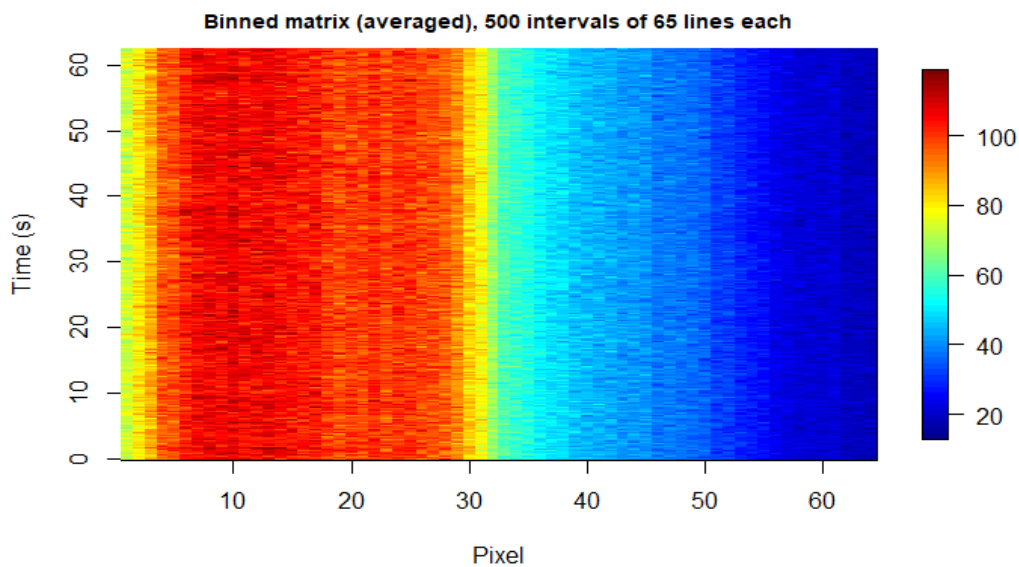


Figure 6.3. Detrended, binned line-scan Venus data.

Now that global data trends have been eliminated, one can proceed to the analysis. For simplicity, `V2` will be overwritten as the detrended data matrix:

```
V2 <- det_data
```

6.1.2 Building the Number and Brightness graph

The `nbline()` function allows to calculate the apparent, as well as the real number and brightness in all the pixels across the scan line, based on the theory and equations previously shown. The result is a `list` with two `vectors` (each corresponding to the Number and the Brightness) that can be readily graphed for analysis.

To compute the apparent Number and Brightness type:

```
# Calculating the apparent Number and Brightness
nbV2 <- nbline(img = V2)

# Graphing the result
par(mar = c(5,5,3,5))
plot(nbV2$AppNumber, type = "l", col = "red", ylim = c(2,15), axes = F, ann = F)
mtext(side = 3, text = "Apparent Number and Brightness", line = 1, las = 1, cex = 1.5)
mtext(side = 2, text = axTicks(4), at = axTicks(4), col = "red", line = 1, las = 1)
mtext(side = 2, text = "N", line = 3, col = "red", las = 1)
axis(1); mtext(side = 1, text = "Pixel", line = 3, las = 1)
par(new = T)
plot(nbV2$AppBrightness~r, type = "l", col = "blue", ylim = c(2,15), axes = F, ann = F)
mtext(side = 4, text = axTicks(2), at = axTicks(2), col = "blue", line = 1, las = 1)
mtext(side = 4, text = "B", line = 3, col = "blue", las = 1)
```

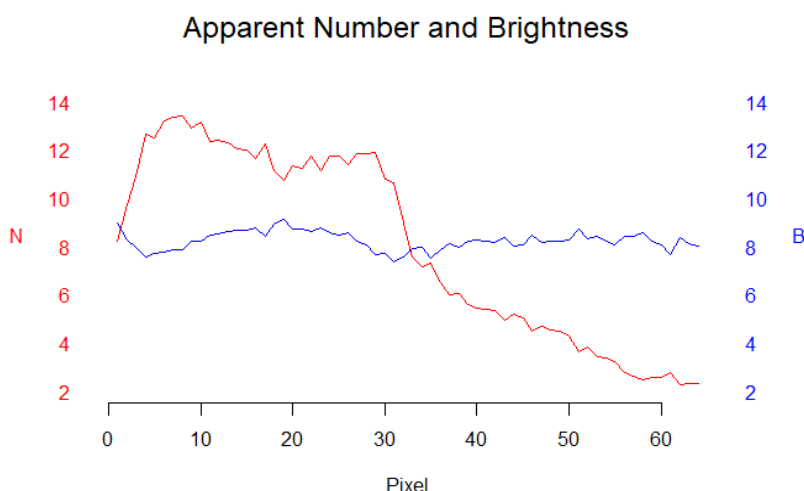


Figure 6.4. Apparent Number and Brightness for each pixel of the Venus line-scan experiment.

Now, let us calculate the real Number and Brightness, for which values of $\sigma_0^2 = 1$, $offset = 0$ and $S = 3.5$ will be used. Type:

```
# Calculating the real Number and Brightness
rnbV2 <- nbline(img = V2, sigma0 = 1, offset = 0, S = 3.5)

# Graphing the result
par(mar = c(5,5,3,5))
plot(rnbV2$RealNumber, type = "l", col = "red", ylim = c(3,25), axes = F, ann = F)
mtext(side = 3, text = "Real Number and Brightness", line = 1, las = 1, cex = 1.5)
mtext(side = 2, text = axTicks(4), at = axTicks(4), col = "red", line = 1, las = 1)
mtext(side = 2, text = "n", line = 3, col = "red", las = 1)
axis(1); mtext(side = 1, text = "Pixel", line = 3, las = 1)
par(new = T)
```

```
plot(rnbV2$RealBrightness~r, type = "l", col = "blue", ylim = c(0,3), axes = F, ann = F)
mtext(side = 4, text = axTicks(2), at = axTicks(2), col = "blue", line = 1, las = 1)
mtext(side = 4, text = expression(epsilon), line = 3, col = "blue", las = 1)
```

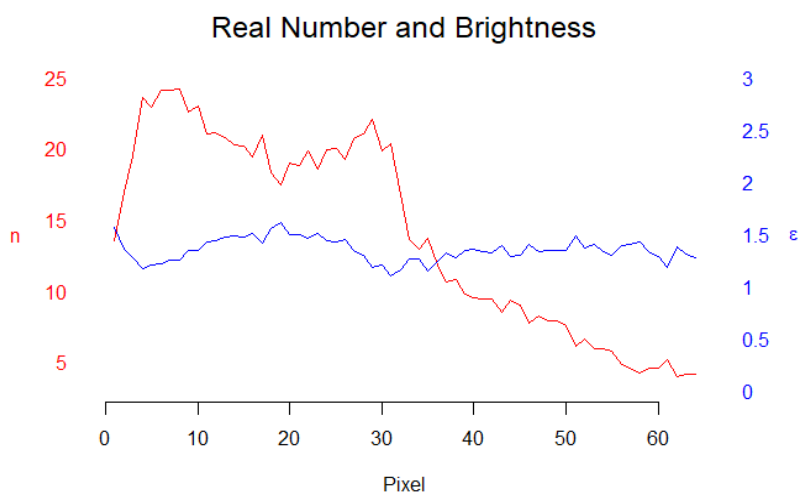


Figure 6.5. Real Number and Brightness for each pixel of the Venus line-scan experiment.

When plotting the N&B graph, one can modify the parameter `ylim` of the `plot()` function to adjust each axis scale for better visualization of the result.

6.1.3 Moving Number & Brightness

Figures 6.4 and 6.5 are graphical representations of the average apparent and real Number and Brightness values for each pixel of a line-scan experiment through the whole time series. Analysis of this result is appropriate if one assumes that the fluorescent molecules' behavior remained constant in time and space throughout the whole experiment. However, it might be more useful (especially in a biological context) to rather observe an evolution of the fluorophores' concentration and aggregation state through time.

In order to achieve this, one can calculate the moving average Number and Brightness of several time windows of a given size and place each of them on top of one another. This would allow have a spatiotemporal map of molecular concentration and study oligomerization and/or aggregation dynamics in highly dynamic microenvironments such as the cytoplasm of a cell.

The `nbline()` function allows to create moving N&B intensity carpets by selecting a time window size with the `w` parameter. This value indicates how many lines will be used to generate a single average N&B line before proceeding to the next segment of size equal to `w`. Note that, when `w` is greater than zero (default), the two original `vectors` are no longer returned and two `matrices` that correspond to the N&B intensity carpets are obtained instead. The N&B intensity carpets will contain the same number of pixels across the scan line as the original signal intensity carpet, however, it will be `w` lines shorter. Keep in mind that the size of `w` can be further adjusted in order to obtain the desired level of detail in the carpets.

Let us create the N&B intensity carpets for the first 10 seconds of the experiment. Type:

```
# Creating the N&B intensity carpets
appNB <- nbline(img = V2[,1:(10/lineTime)], w = 100)
realNB <- nbline(img = V2[,1:(10/lineTime)], sigma0 = 1, offset = 0, S = 3.5, w = 100)

# Graphing the result
par(mfrow = c(1,2))
image.plot(x = r, y = Time[1:dim(appNB$AppNumber)[2]], z = appNB$AppNumber,
  main = "N", xlab = expression(r(mu*m)), ylab = "Time (s)")
image.plot(x = r, y = Time[1:dim(appNB$AppBrightness)[2]], z = appNB$AppBrightness,
  main = "B", xlab = expression(r(mu*m)), ylab = "Time (s)")
image.plot(x = r, y = Time[1:dim(realNB$RealNumber)[2]], z = realNB$RealNumber,
  main = "n", xlab = expression(r(mu*m)), ylab = "Time (s)")
image.plot(x = r, y = Time[1:dim(realNB$RealBrightness)[2]], z = realNB$RealBrightness,
  main = expression(epsilon), xlab = expression(r(mu*m)), ylab = "Time (s)")
```

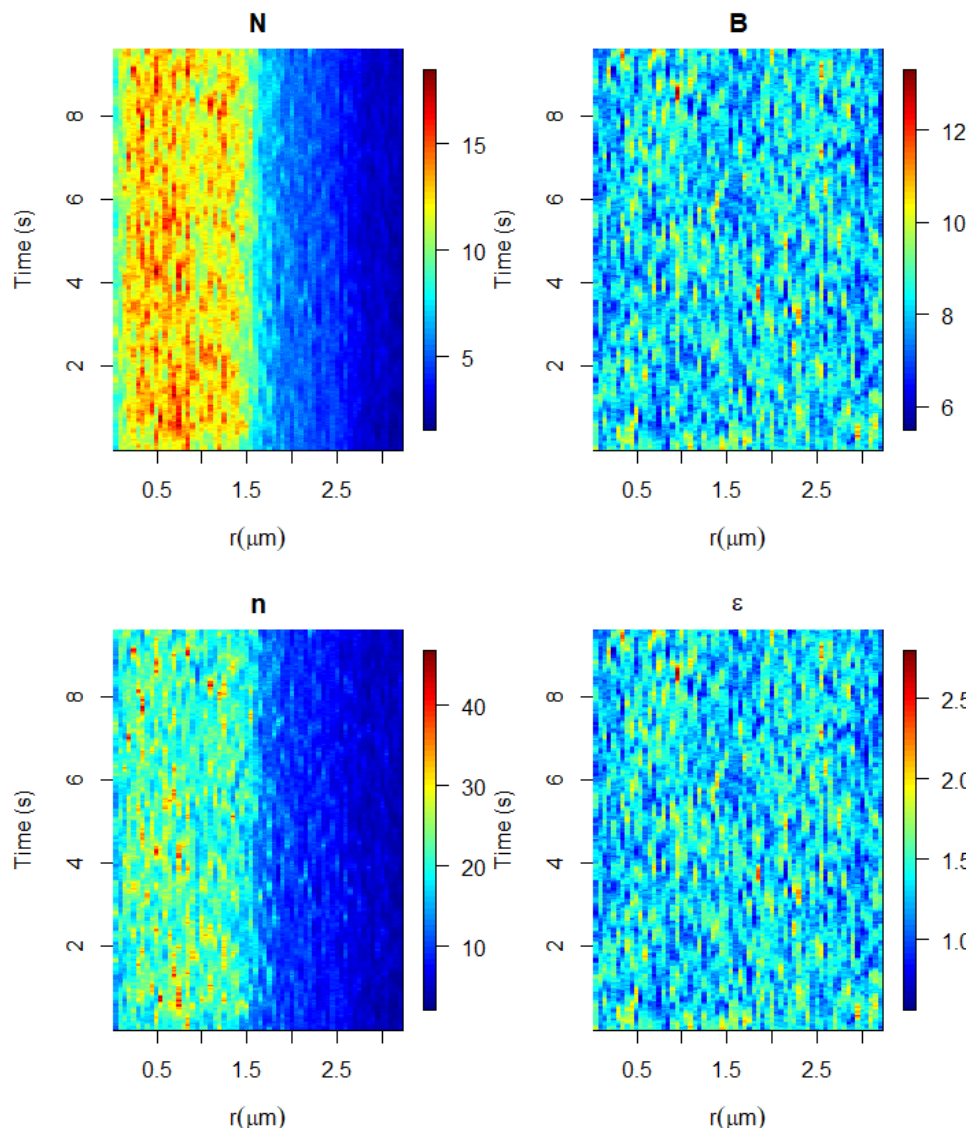


Figure 6.6. Moving N&B intensity carpets. Upper Row corresponds to the apparent N&B while the lower row corresponds to the real N&B. Only first 10 seconds of the line-scan experiment are displayed.

In order to better visualize the transitions in concentration/aggregation state of the molecules, let us use the `smoothCarpet()` function to reduce the contrast between the values of each pixel in the N&B carpets. Type:

```
# Smoothing the N&B intensity carpets
s.realN <- smoothCarpet(img = realNB$RealNumber, dfH = 30, dfV = 30)
s.realB <- smoothCarpet(img = realNB$RealBrightness, dfH = 30, dfV = 30)

# Graphing the result
par(mfrow = c(1,2))
image.plot(x = r, y = Time[1:dim(s.realN)[2]], z = s.realN, main = "n (smoothed)",
           xlab = expression(r(mu*m)), ylab = "Time (s)")
image.plot(x = r, y = Time[1:dim(s.realN)[2]], z = s.realB,
           main = expression(epsilon("smoothed")),
           xlab = expression(r(mu*m)), ylab = "Time (s)")
```

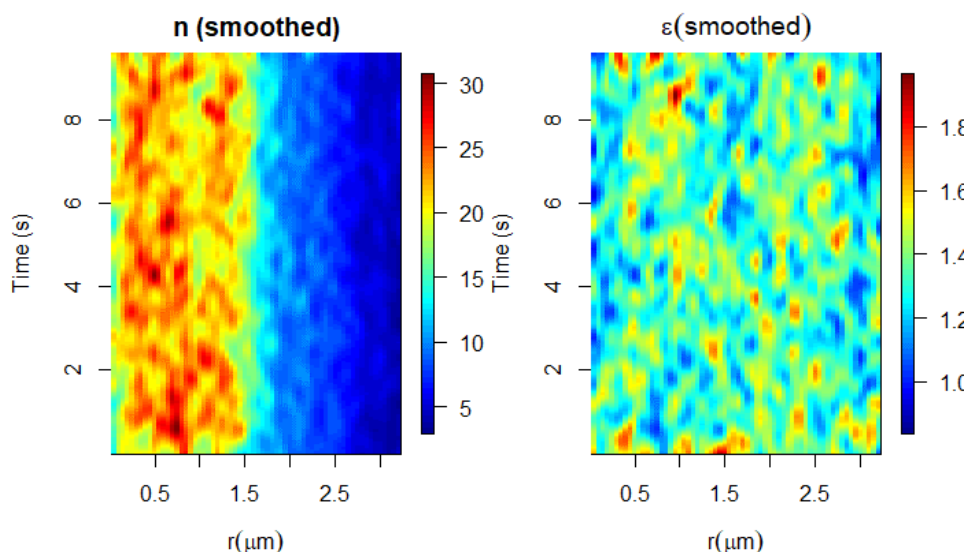


Figure 6.7. Smoothed moving real N&B intensity carpets in figure 6.6. 30 degrees of freedom were used for both the vertical and horizontal axes.

The intensity carpets in figure 6.7 are graphical representations of how the average number of Venus dimers and their aggregation state evolves over time. Note that, while the concentration of Venus dimers remains higher in the cytoplasm region than inside the nucleus, their aggregation state is more equally distributed across both sides of the cell, with an average real number value between 1.5 and 2 which, given the experimental parameters, corresponds to an oligomerization state of two.

6.2 Pair Correlation of Molecular Brightness

The Pair Correlation approach allows to create a molecular flux map as it measures the time it takes for a particle to move from one position to another along the line scan; though, it does not discriminate mobility based on aggregation state. On the other hand, N&B can determine the relative abundance of fluorophores and their stoichiometry based on statistical moment analysis of the fluorescence fluctuations (but it is mostly insensitive to particle dynamics).

Originally proposed by Hinde and coworkers in 2016 [13], the pCOMB approach combines the virtues of both the pCF and N&B methods as it allows to track the mobility of different oligomeric species within the same microenvironment. pCOMB calculates the spatio-temporal correlation between the brightness fluctuations. Through moment analysis, pCOMB amplifies the signal from the brightest species present in the sample and then uses pair correlation functions to filter the dynamics of the extracted oligomeric population based on the arrival time between two locations. Thus, pair correlation analysis is directly performed over the brightness fluctuations.

Each pixel in the line scan has a temporal signal intensity fluctuation of average $\langle F(t) \rangle$ (first moment) and variance $(F(t) - \langle F(t) \rangle)^2$ (second moment), which can be directly transformed into a brightness fluctuation by following

$$B = \frac{(F(t) - \langle F(t) \rangle)^2}{\langle F(t) \rangle}$$

Then, in order to calculate the pair correlation of the brightness fluctuations, a spatial component (r) is introduced to the general correlation function:

$$G_B(\tau, \delta r) = \frac{\langle B(t, r) * B(t + \tau, r + \delta r) \rangle}{\langle B(t, r) \rangle * \langle B(t, r + \delta r) \rangle} - 1$$

Where t and r are the current time point and position in the line scan, respectively, and τ and δr are the temporal and spatial shifts at which the pair correlation is calculated.

6.2.1 Building the pCOMB carpet

The Venus line-scan experiment data will continue to be used for this example. Starting from the detrended data matrix, `V2`, type:

```
# Creating the pCOMB carpet
nPoints <- 16000
pcombV2 <- pcomb(img = V2, nPoints = nPoints, dr = 10, w = 100)

# Defining new 'tau' and 'r' axis
Tau <- (1:nPoints)*lineTime
r <- (1:dim(pcombV2)[1])*pixelSize

# Smoothing the pCOMB carpet
s.pcombV2 <- smoothCarpet(img = pcombV2, dfH = 20, dfV = 20)

# Graphing the results
par(mfrow = c(1,2))
image.plot(x = r, y = log10(Tau), z = pcombV2, main = "V2 pCOMB, dr = 10",
           xlab = expression(r(mu*m)), ylab = expression(tau(s)), legend.mar = 10)
image.plot(x = r, y = log10(Tau), z = s.pcombV2, main = "V2 pCOMB, dr = 10 (smoothed)",
           xlab = expression(r(mu*m)), ylab = expression(tau(s)), legend.mar = 10)
```

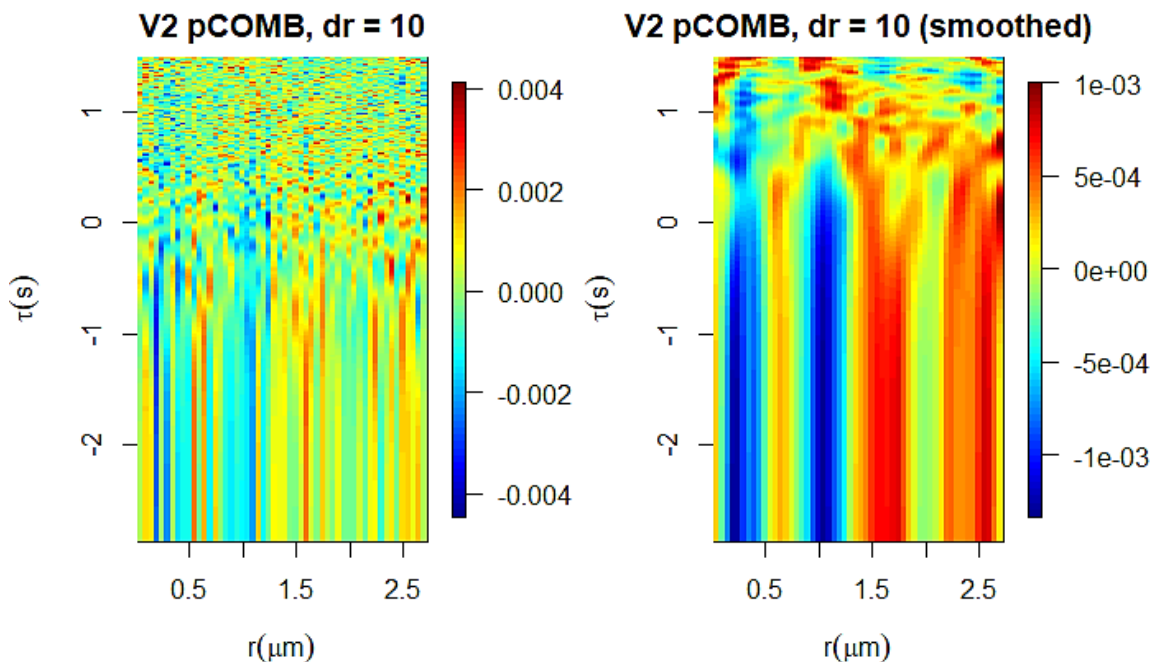


Figure 6.8. pCOMB carpet of the Venus line-scan data. The spatial shift δr used was 10 and a degree of 20 for both the vertical and horizontal axes was used to generate the smooth carpet (right panel).

Note the presence of arcs on the pCOMB carpet; they are indicative of single moving oligomers [13]. Similar to figure 6.7, a smoothing process can be applied to the pCOMB carpet to polish the roughness between pixels and acquire a clearer picture of the analysis.

The `dr` parameter in the `pcomb()` function can be further adjusted in a similar fashion as in [section 5.2.3.2](#) to establish the correlation distance.

REFERENCES

1. Timmermann, K. E., & Nowak, R. D. (1999). Multiscale modeling and estimation of Poisson processes with application to photon-limited imaging. *IEEE Transactions on Information Theory*, **45**(3), 846-862.
2. Pinto-Cámara, R., Linares, A., Hernández, H.O., Moreno-Gutiérrez, D.S., Martínez-Reyes, J.D., Rendón-Mancha, J.M., Wood, C.D. & Guerrero, A. (2020). FCSlib: an open-source tool for fluorescence fluctuation spectroscopy analysis for mobility, number, and molecular brightness in R. *Bioinformatics*.
3. Tovar-Herrera, O. E., Rodríguez, M., Olarte-Lozano, M., Sampedro-Guerrero, J. A., Guerrero, A., Pinto-Cámara, R., . . . & Segovia, L. (2018). Analysis of the Binding of Expansin Exl1, from *Pectobacterium carotovorum*, to Plant Xylem and Comparison to EXLX1 from *Bacillus subtilis*. *ACS omega*, **3**(6), 7008-7018.
4. Migueles-Ramirez, R. A., Velasco-Feliz, A. G., Pinto-Camara, R., Wood, C. D., & Guerrero, A. In Fluorescence Fluctuation Spectroscopy in Living Cells. *Microscopy*.
5. Buschmann, V., Krämer, B., Koberling, F., Macdonald, R., & Rüttinger, S. (2009). Quantitative FCS: determination of the confocal volume by FCS and bead scanning with the microtime 200. *Application Note PicoQuant GMBH*.
6. Kaputsa, P. (2010). Absolute diffusion coefficients: compilation of reference data for FCS calibration. *Application Note PicoQuant GMBH*.
7. Lakowicz, J. R. (Ed.). (2013). Principles of fluorescence spectroscopy. *Springer Science & Business Media*.
8. Meseth, U., Wohland, T., Rigler, R., & Vogel, H. (1999). Resolution of fluorescence correlation measurements. *Biophysical journal*, **76**(3), 1619-1631.
9. Digman, M. A., Brown, C. M., Horwitz, A. R., Mantulin, W. W., & Gratton, E. (2008). Paxillin dynamics measured during adhesion assembly and disassembly by correlation spectroscopy. *Biophysical journal*, **94**(7), 2819-2831.
10. Digman, M. A., & Gratton, E. (2009). Imaging barriers to diffusion by pair correlation functions. *Biophysical journal*, **97**(2), 665-673.
11. Qian, H., & Elson, E. L. (1990). Distribution of molecular aggregation by analysis of fluctuation moments. *Proceedings of the National Academy of Sciences*, **87**(14), 5479-5483.
12. Qian, H., & Elson, E. L. (1990). On the analysis of high order moments of fluorescence fluctuations. *Biophysical journal*, **57**(2), 375-380.
13. Hinde, E., Pandžić, E., Yang, Z., Ng, I. H., Jans, D. A., Bogoyevitch, M. A., ... & Gaus, K. (2016). Quantifying the dynamics of the oligomeric transcription factor STAT3 by pair correlation of molecular brightness. *Nature communications*, **7**(1), 1-14.