# Getting Started with Rcell (Version 1.1-5)

Alan Bush

October 27, 2011

## 1 Introduction

Rcell is a package design to load, manipulate and visualize datasets generated with Cell-ID (Chernomoretz et al. (2008)) in R. These datasets contain about 70 different variables (columns) and can contain thousands of registers (rows). An analysis of the dataset usually includes filtering it for spurious or badly found cells, creating new (normalized) variables from existing ones and creating plots and images to visualize the data. You can download the latest version of the package from http://sourceforge.net/projects/cell-id/files. To load the package open *R* and type

```
> library(Rcell)
```

In this document I will guide you through a standard analysis of a example dataset.

## 2 Load Cell-ID Data to R

Cell-ID creates a series of folders named "Position", in which one can find the "out_all" file containing the dataset for the given position. Cell-ID also creates some other files containing information about the image files used for each channel and time frame, and the parameters with which the program was run. The load.cellID.data function can be used to load this data into R. This function searches for the "Position" folders in a specified directory. The default directory is the working directory, but this can be modified with the path argument. To load your data in the directory "C:\microscopy-data\my-experiment" you can first set it as your working directory, and then execute load.cellID.data

```
> setwd("C:\\microscopy-data\\my-experiment")
> X <- load.cellID.data()
```

Note the use of the double backslash (\\) when setting the working directory. This is required because the backslash (\) is a reserved character in *R*. You can also use a single forward-slash (/) instead of the double backslash, as in Unix paths. load.cellID.data returns a object of class cell.data that has to be assign (<-) to a variable. Through out this tutorial will call this object X. If you don't have a experiment to analyze you can still go through the tutorial with the example dataset (Colman-Lerner et al. (2005)). To load this dataset type

```
> data(ACL394)
```

This replaces the X object if it exists.

## 3 Inspect the Data

A quick way to look at the content of your dataset is to use the summary function.

```
> summary(X)

 Cell-ID data object summary

loaded on: Fri Oct 07 20:09:41 2011
loaded from: F:/microscopy-data/acl/Rcell-example-data
channels: CFP, YFP
positions: 1-3,8-10,15-17,22-24,29-31
time frames: 0-13

id vars: pos, t.frame, cellID
morphological vars: xpos, ypos, a.tot, fft.stat, perim, maj.axis, min.axis,
  a.vacuole, a.local.bg, a.local, a.local2.bg, a.surf, sphere.vol
channel specific morphological vars*: a.nucl, a.nucl1, a.nucl2, a.nucl3,
  a.nucl4, a.nucl5, a.nucl6
channel specific fluorescence vars*: f.tot, f.nucl, f.vacuole, f.bg, f.nucl1,
  f.nucl.tag1, f.nucl2, f.nucl.tag2, f.nucl3, f.nucl.tag3, f.nucl4, f.nucl.tag4,
  f.nucl5, f.nucl.tag5, f.nucl6, f.nucl.tag6, f.local.bg, f.local2.bg
  *append channel postfix (.c, .y) to obtain variable name
select keywords: id.vars, id.vars.deriv, morpho, fluor, QC, as.factor,
  all, CFP, YFP
```

This function returns a brief description of the `cell.data` object, including the path from where it was loaded, the positions and time frames, the available variables and channels. (There might be minor differences between the output in this file and what you get when executing the same commands.)

You can see that the example dataset is a time course and has the positions 1-3,8-10,15-17,22-24,29-31. (Some positions where deleted from the original experiment to reduce the size of the package). To learn more about the experiment you can read the documentation of the dataset.

```
> help(ACL394)
```

Once loaded to *R*, we can use its plotting features to visualize our data (Wickham (2009)). For example to plot the total YFP fluorescence of position one (`pos==1`), type in the following command (Figure 1).

```
> cplot(X, f.tot.y ~ t.frame, subset = pos == 1)
```

`cplot` function is used to plot your cell data. As all the functions of the Rcell package, its first argument is a `cell.data` object as returned by `load.cellID.data`. The second argument specifies what variables should be plotted in the "x" and "y" axis. It uses a formula notation, in the form `y~x`. The `subset` argument is used to subset or filter the dataset before plotting, in this case we specify that only the registers in which the *pos* variable equals 1 should be included. Note the use of the "is equal to" operator (`==`). A common mistake is to use the assignation operator (`=`) in logical conditions of the `subset` argument.

To see all the positions at a glance we can use faceting, specifying a formula to the `facets` argument of `cplot` (Figure 2).

```
> cplot(X, f.tot.y ~ t.frame, facets = ~pos)
```

## 4 Adding Metadata and Creating New Variables

Once the data is loaded in `R` it is a good practice to add metadata to the dataset. By "metadata" I mean variables that describe the experiment, and are not calculated by Cell-ID. For example, in the ACL394 dataset positions 1-3 correspond to well 1, positions 8-10 to well 2, etc. Wells 1 to 5 were stimulated with

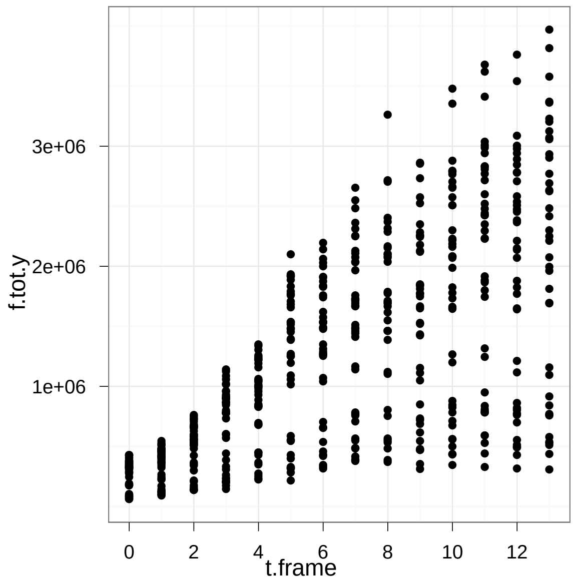

Figure 1: f.tot.y vs t.frame for pos==1

$\alpha$-factor at concentrations 1.25, 2.5, 5, 10 and 20 nM respectively. To add this information to the dataset we can use the `load.pdata` function. This function searches for a file named "pdata.txt" in the working directory. [1] "pdata.txt" should be a tab delimited file with a description of each position (Table 1). You can create such a file in Excel (Save As > Tab Delimited File).

```
merging by pos
merged vars:
  well: integer w/values 1, 2, 3, 4, 5
  AF.nM: numeric w/values 1.25, 2.5, 5, 10, 20
```

```
> X <- load.pdata(X)
```

Adding metadata makes the code easier to read, as you can subset and plot by biologically relevant variables instead of having to use (and remember!) what is in each position.

The `transform` function can be used to create new variables from the existing ones. For instance, we could create a variable of total fluorescence corrected for background fluoresce (i.e. fluorescence from pixels not associated with any cell).

```
> X <- transform(X, f.total.y = f.tot.y - f.bg.y * a.tot)
```

We call this new variable *f.total.y*. Note that *f.bg.y* is the mode (most often value) fluorescence of background pixels, so we need to multiply it by the area of the cell *a.tot*.

For more details on how to transform your dataset and create new variables see the "transform" vignette

```
> vignette("transform")
```

[1]You can change the working directory with setwd('C:/my-folder/')

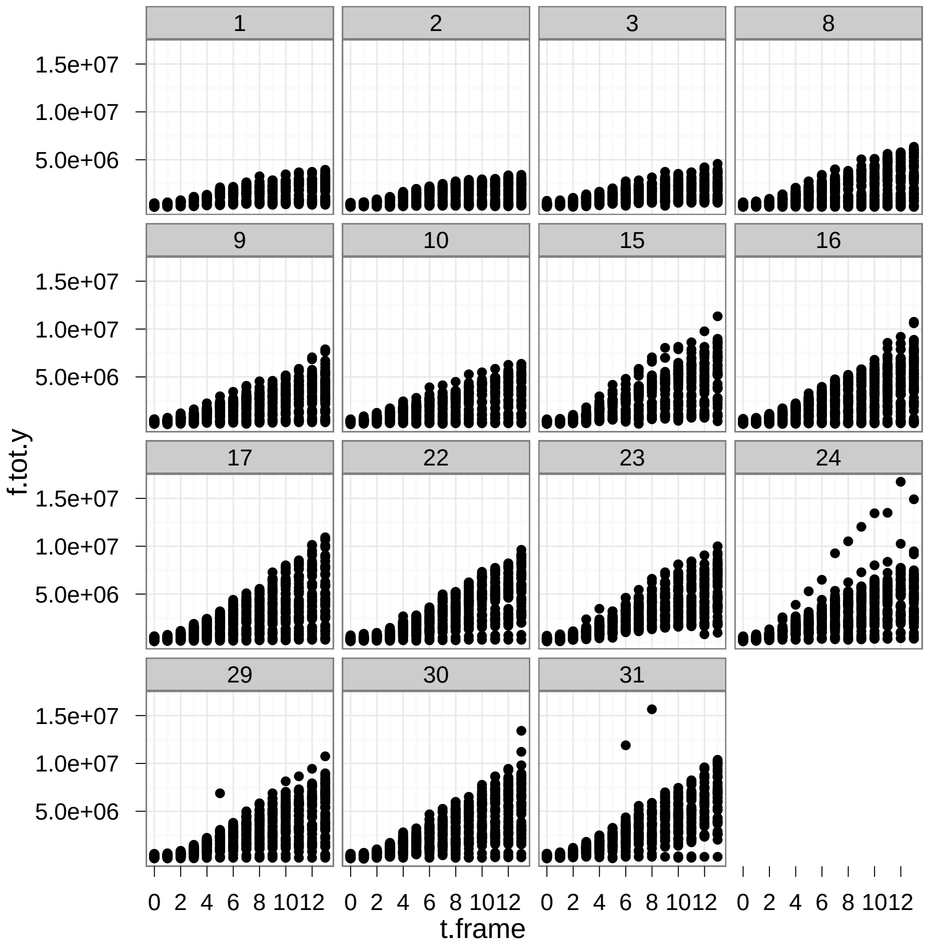

Figure 2: f.tot.y vs t.frame faceted by position

| pos | well | AF.nM |
|---|---|---|
| 1 | 1 | 1.25 |
| 2 | 1 | 1.25 |
| 3 | 1 | 1.25 |
| 8 | 2 | 2.50 |
| 9 | 2 | 2.50 |
| 10 | 2 | 2.50 |
| 15 | 3 | 5.00 |
| 16 | 3 | 5.00 |
| 17 | 3 | 5.00 |
| 22 | 4 | 10.00 |
| 23 | 4 | 10.00 |
| 24 | 4 | 10.00 |
| 29 | 5 | 20.00 |
| 30 | 5 | 20.00 |
| 31 | 5 | 20.00 |

Table 1: example pdata.txt file

## 5 Quality Control

The datasets produced by Cell-ID normally contain spurious or badly found cells. This cells have to be removed from the dataset as they add noise and complicate interpretation of the data. To this end the functions `QC.filter`, `QC.undo` and `QC.reset` are provided. `QC.filter` adds cumulative filters to the *QC* variable. All of Rcell functions by default ignore the registers that don't pass this filter. The decision of which filter to apply is not trivial and will depend on your particular dataset. For example, if all your cells are approximately spherical you can apply a filter over *fft.stat*, a measure of non-circularity (small *fft.stat* indicates very circular boundaries). To define at what value of *fft.stat* to do the cut we can take a look at an histogram of this variable (Figure 3).

```
> cplot(X, ~fft.stat)
```

Note that when only the "x" variable is defined in the formula, `cplot` creates a histogram. From this histogram we can see that very few cells have *fft.stat* larger than 0.5. We can see what these cells look like using the `cimage` function (Figure 4).

```
> img4 <- cimage(X, channel = "BF.out", subset = fft.stat > 0.5 & t.frame == 11 &
+     pos %in% c(1, 8, 15, 22, 29), bg.col = "white")
```

For more details on `cimage` read the functions vignette.

```
> vignette("cimage")
```

You can see in Figure 4 that cells with `fft.stat > 0.5` are badly found cells that we want to eliminate from our dataset. The `QC.filter` function applies the filter and informs us how many registers were excluded by ALL the QC filters applied until now.

```
> X <- QC.filter(X, fft.stat < 0.5)
```

```
cumulative row exclusion: 1.1%
```

An other useful way to filter a time courses experiment is by *n.tot*, the total number of time frames in which a cell appears. Spurious cells usually only appear in a single time frame. First we will use `update.n.tot` to update this variable, and then `QC.filter` to apply it.

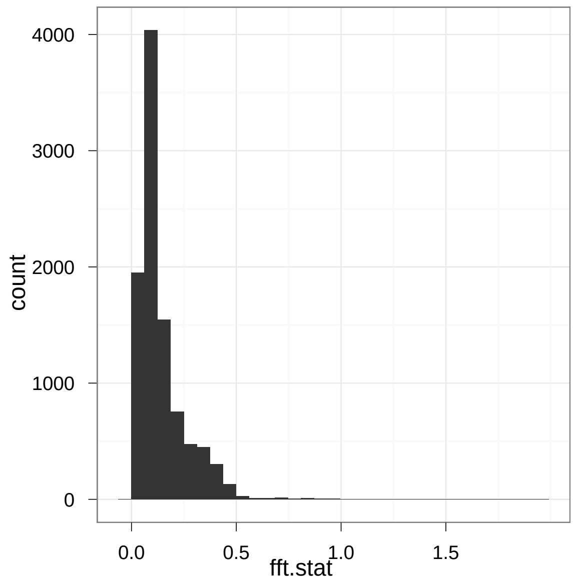

Figure 3: fft.stat histogram

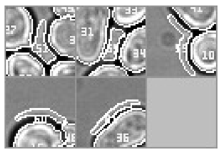

Figure 4: Cells with fft.stat > 0.5

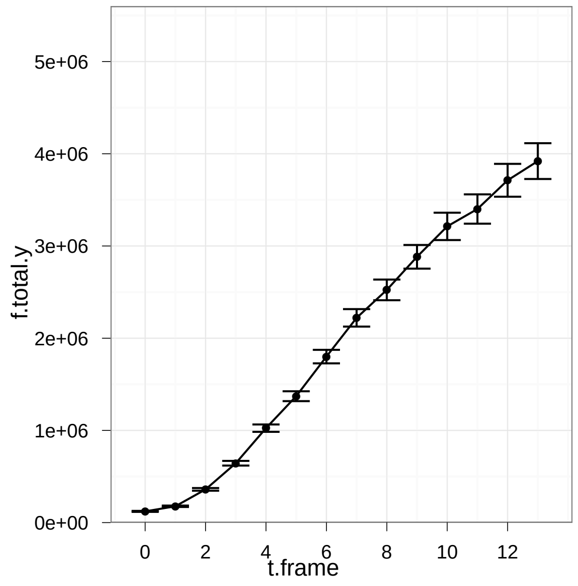

Figure 5: mean f.tot.y vs t.frame

```
> X <- update.n.tot(X)
> X <- QC.filter(X, n.tot == 14)
```

```
cumulative row exclusion: 21.5%
```

If you are not sure on the cut to apply you can plot a histogram of *n.tot* If you don't like a filter you just applied you can undo it with `QC.undo`, or you can reset all filters with `QC.reset`.

## 6 Plotting the data

In the previous sections we've seen how to do scatter plots and histograms, and how to facet the plot. You can do a lot of different plots. Here I will show a couple of useful examples. First of all we will plot the mean YFP fluorescence against time, differentiating the pheromone doses by color (Figure 5).

```
> cplotmean(X, f.total.y ~ t.frame, color = AF.nM, yzoom = c(0, 5600000))
```

The function `cplotmean` calculates the mean and standard error of the mean for "y", at each level of the "x". Note that to assign a color to each level of *AF.nM* we just have to assign this variable to the argument `color`. This can also be done with `cplot`. By default `cplotmean` uses the entire range of the data for the plot. To zoom in a region you can use the `yzoom` and `xzoom` arguments as shown in the example.

We can also study the correlation between YFP, the pheromone reporter gene, and CFP, constitutive under the Act1 promoter (Figure 6).

```
> cplot(X, f.tot.y ~ f.tot.c, color = AF.nM, size = a.tot, alpha = 0.5, subset = t.frame ==
+     13)
```

Note that we mapped `color` to *AF.nM* and `size` to *a.tot*. We also use semi-transparency with `alpha=0.5` to avoid over-plotting (not available in all devices). With the `subset` argument we select the last time frame.

For more details on `cplot` and other plotting functions read the cplot vignette.

```
> vignette("cplot")
```

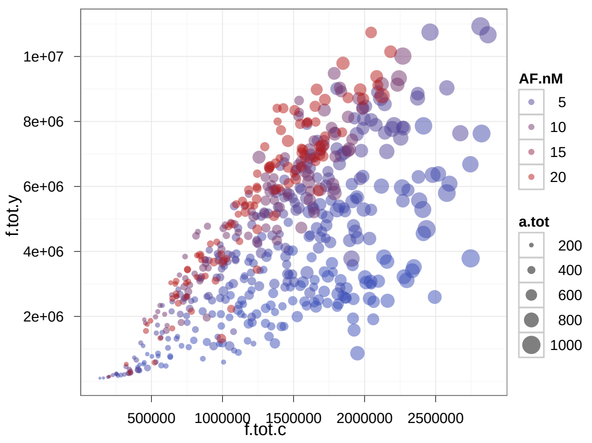

Figure 6: f.tot.y vs t.frame, colored by dose of AF and size proportional to cell area

## References

Hadley Wickham. ggplot2: Elegant graphics for Data Analysis *Springer* 2009

Colman-Lerner, Gordon et al. (2005). Regulated cell-to-cell variation in a cell-fate decision system. *Nature*, 437(7059):699-706.

Chernomoretz, Bush et al. (2008). Using Cell-ID 1.4 with R for Microscope-Based Cytometry *Curr Protoc Mol Biol.*, Chapter 14:Unit 14.18.