Plotting with Rcell (Version 1.1-5)

Alan Bush

October 27, 2011

1 Introduction

Rcell uses the functions of the **ggplots2** package to create the plots. This package created by Wickham implements the ideas of Wilkinson (2005) of a "grammar of graphics", i.e. a formal description of statistical graphics. I recommend to read the book on the ggplot2 package. If you are already familiar with this package you can skip this document and read the help on `cplot`. **Rcell** plotting functions return objects of class `ggplot`, and therefore can be combined with other functions of the **ggplot2** package in a transparent manner. In this document I will present and explain different plots that can be made with the **Rcell** plotting functions. If you haven't done so, read the "Getting Started with Rcell" vignette before proceeding.

```
> vignette("Rcell")
```

2 A Grammar of Graphics

There exist an enormous variety of statistical plots, as a few example we can mention scatter plots, bar plots, histograms, piecharts, box plots, etc. The grammar of graphics attempts to describe all these different plots in a simple, coherent language. It defines the following elements to describe a plot:

data: the dataset to be plotted

geom: the geometrical object used to represent the data (e.g. points, bars, lines, etc)

aesthetic mapping: defines what variable of the dataset will be mapped to each aesthetic attribute of the geometrical object

stat: statistical transformations to be performed before plotting (e.g. calculate mean, bin the data for a histogram, etc)

position: minnor position adjustments of the geoms

scale: defines the scale for the mapping between a variable and a aesthetic attribute (e.g. lineal, log)

coordinate: the coordinate system to be used. This is normally cartesian coordinates, but can also be polar coordinates for example.

faceting: describes how the plot should be facet into a series of subplots.

ggplot2 implements a "layered grammar of graphics", meaning that several layers containing the mentioned elements can be added to a plot. In the following sections I will try to make this clear by a series of explained examples.

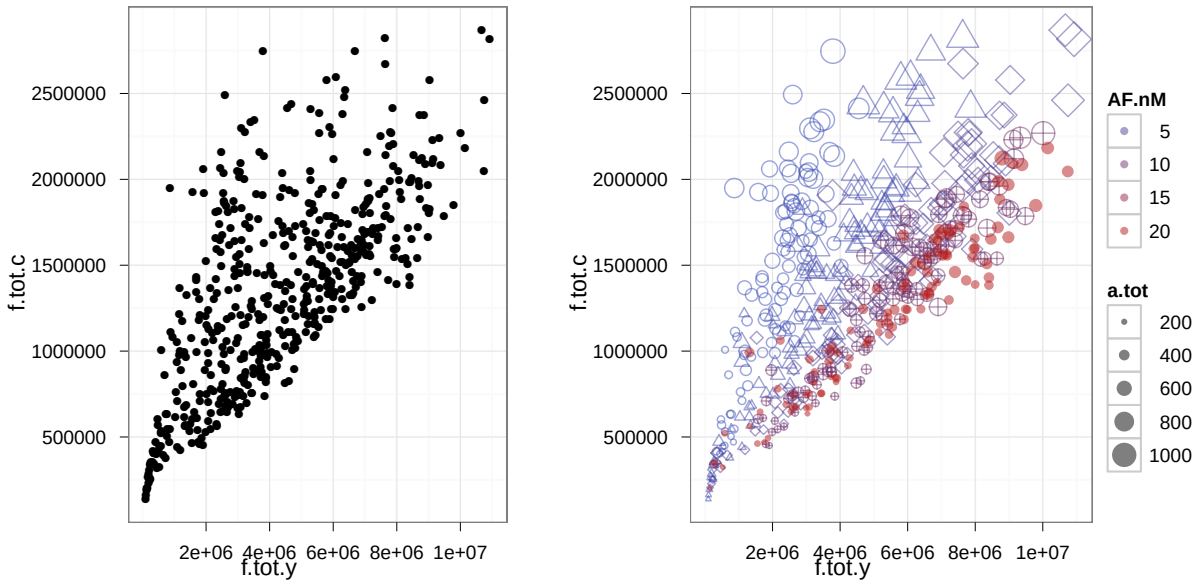

Figure 1: Left: Scatter plot of total CFP (f.tot.c) vs YFP (f.tot.y) fluorescence. Right: same plot with shape mapped to treatment (AF.nM), and size mapped to cell area (a.tot).

3 Scatter Plots

You can create a scatter plot of YFP vs CFP total fluorescence (Figure 1 left) by typing [1]

```
> cplot(X, x = f.tot.y, y = f.tot.c, subset = t.frame == 13)
```

The first argument of the `cplot` function is the `cell.data` object, i.e. the *data* element of the plot. The second and third argument specify the *aesthetic mapping*; *f.tot.y* variable of the dataset should be mapped to the x position aesthetic, and *f.tot.c* to the y position aesthetic (the `subset` argument just indicates we don't want to plot the entire dataset, just some registers). As you can see not all elements of the grammar where specified. The ones that are not are set to default values. For example *geom* is set to `"point"`, meaning that a point is going to be used to represent the data. The aesthetic attributes available depend on the selected *geom*. For example a point *geom* has *x*, *y*, *size*, *shape*, *color*, *fill* and *alpha* (transparency). To see all aesthetic attributes and default values of the available *geom* go to `http://had.co.nz/ggplot2`. There are many attributes that can be mapped to variables, therefor increasing the amount of information a plot contains. You can specify the *aesthetic mapping* of these attributes as we did before.

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

We have mapped the *size* aesthetic to the *a.tot* variable, and the *color* and *shape* aesthetics to the *AF.nM* variable (Figure 1 right). This mapping redundancy can be useful to present the data to color blind persons or if the plot is going to be presented in a black and white device (e.g. paper). Note that the *alpha* (transparency) aesthetic has been mapped to a constat value and not a variable. Making objects semi-transparents can reduce over-plotting.

[1] load the example dataset if you haven't done so with data(ACL394)

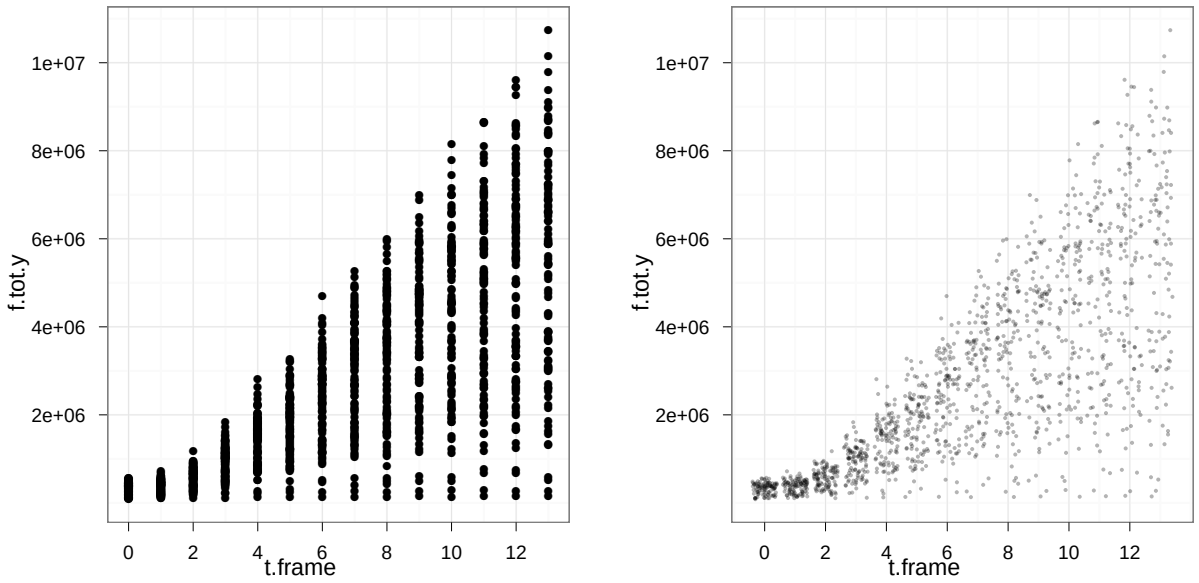

Figure 2: Left: Time course of YFP fluorescence (f.tot.y) for the 20nM dose. Right: same plot with smaller, semitransparent and jittered points to reduce over-plotting.

4 Time Courses and Over-plotting

A time course plot is basically a scatter plot in which the x axis is time. To get a time course of YFP fluorescence (Figure 2 left) type in

```
> cplot(X, x = t.frame, y = f.tot.y, subset = AF.nM == 20)
```

The same plot can be obtained using a formula notation of the form `y~x` as the second argument in the call. When using this notation you can use the generic function `plot` instead of `cplot`. The next two lines result in the same plot (Figure 2 left).

```
> cplot(X, f.tot.y ~ t.frame, subset = AF.nM == 20)
> plot(X, f.tot.y ~ t.frame, subset = AF.nM == 20)
```

You can see that the *t.frame* variable takes discrete levels, thus causing overplotting; to many points fall in the same region and you can no longer estimate the density (Figure 2 left). The plot is "saturated". There are several strategies to avoid overplotting, for example one could reduce the *size* of the *geom* and use *alpha* blending. Another strategy is to use the *position* adjustment. By default this is set to `"identity"`, meaning that the points fall where the data says so. If you change this to `"jitter"`, the points are jittered (a small white noise is added to the x position) to avoid overplotting (Figure 2 right).

```
> cplot(X, f.tot.y ~ t.frame, size = 1, alpha = 0.3, position = "jitter", subset = AF.nM ==
+     20)
```

5 Statistical Transformations

Another way to avoid overplotting is by using other statistical plots, for example a box and whisker plot (Figure 3 left).

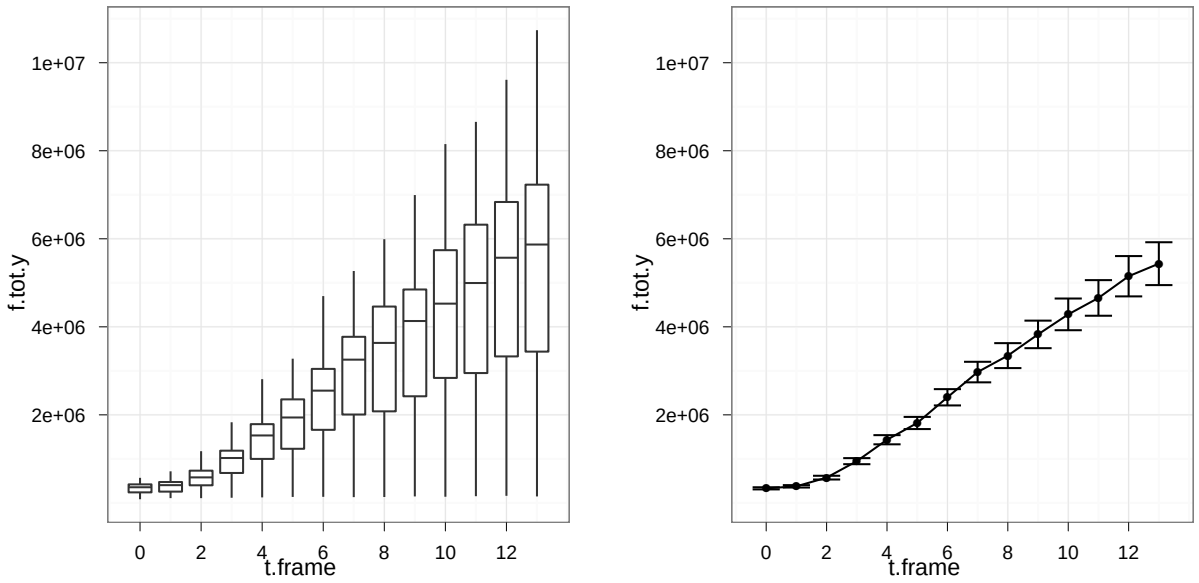

Figure 3: Left: box-whisker plot for the time course data of f.tot.y. Right: means and 95% confidence intervals of the mean for the same data.

```
> cplot(X, f.tot.y ~ t.frame, group = t.frame, geom = "boxplot", subset = AF.nM ==
+     20)
```

Setting the *geom* to `"boxplot"` creates the desired plot. If you think the steps required to produce this plot, you will realize that some preprocessing of the data has to occur before the plot is actually done. For instance the 25%, 50% and 75% quantiles have to be calculated for each group (defined by the *group* argument). This means that a "statistical transformation" (*stat*) of the data has to be done. The *stat* that does this calculation is called `"boxplot"`. When we selected the `"boxplot"` *geom*, the *stat* was set by default to `"boxplot"`. Every *geom* has a default *stat*. For example *geom* `"point"` has the default *stat* `"identity"` that does no transformation to the data, exactly what we need to do scatter plots.

Another way to show this data is to plot the mean and a confidence interval for the mean, for each time. We can do this selecting *stat* `"summary"` and specifying which function is to be used to calculate the summary statistics by the `fun.data` argument.

```
> cplot(X, f.tot.y ~ t.frame, subset = AF.nM == 20, stat = "summary", fun.data = "mean_cl_normal")
```

Because plotting the mean is a fairly common thing to do, **Rcell** provides the convenience function `cplotmean` that sets *stat* to `"summary"` and `fun.data` to `"mean_cl_normal"`. The same plot (Figure 3 right) can be done with

```
> cplotmean(X, f.tot.y ~ t.frame, subset = AF.nM == 20)
```

Note that the plotting range used for this plot is the data range. You probably want to zoom-in the y axis to show only the range of the means. To do this you can use the `yzoom` argument, for example `yzoom=c(0,6e6)`. The `ylim` argument is also available, but it filters the data BEFORE the statistical transformation, thus the calculated mean will depended on the plotting range! A warning is issued if you use `ylim` in a call to `cplotmean`.

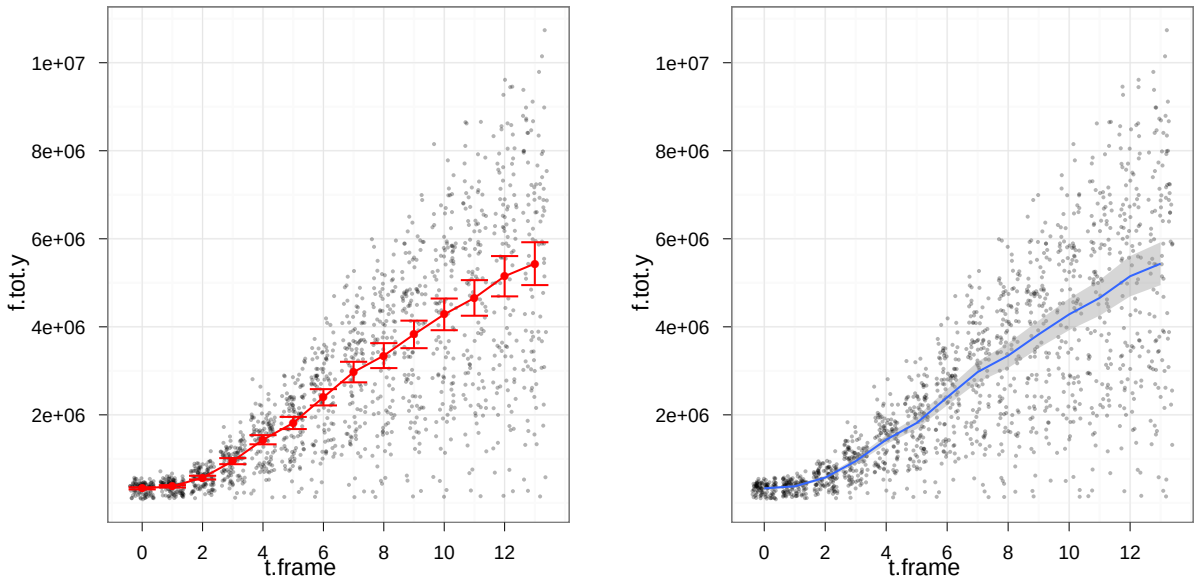

Figure 4: Left: means and confidence intervals layer added to the scatter plot. Right: Using the "smooth" geom to represent the mean and CI.

6 Adding Layers to a Plot

Many time plots with several layers are required. For example we could want to add a layer with the mean and confidence interval for the mean, to a scatter plot. To create such a multilayer plot we first create a plot and assign it to a variable [2].

```
> p <- cplot(X, f.tot.y ~ t.frame, size = 1, alpha = 0.3, position = "jitter",
+     subset = AF.nM == 20)
```

We can then add a layer with the mean using the `clayermean` function (Figure 4 left).

```
> p + clayermean(color = "red")
```

When used with no `cell.data` or mapping arguments, the added layer uses the same *data* and *aesthetic mapping* as the plot to which it is being added. You can specify the layer to use another dataset or aesthetic mapping by passing arguments to `clayermean`. The default geom for the layer can also be changed, for example to the `"smooth"` *geom* (Figure 4 right).

```
> p + clayermean(geom = "smooth")
```

Only layers can be added to plots, you can't add plots to plots nor layers to layers. Create the layers with `clayer` and `clayermean`.

7 Faceting Plots

Faceting creates many subplots showing different subsets of the data. You can facet a plot by providing a *facets* argument. This argument should be a formula. If only the right term is provided (`~myVar`) then

[2] The plotting functions return a object of class ggplot

facet_wrap function of **ggplot2** is used. This function creates a one dimensional strip of subplots "wrapped" into 2D to save space. If you provide both terms to the *facets* formula, a grid of subplots is created by facet_grid function. An example of each is shown in Figure 5 and 6.

```
> cplot(X, f.tot.c ~ f.tot.y, facets = ~t.frame, size = 0.5, alpha = 0.5)
```

```
> cplot(X, f.tot.c ~ f.tot.y, facets = AF.nM ~ t.frame, subset = t.frame %in%
+     c(1, 4, 7, 10, 13), size = 0.5, alpha = 0.5)
```

8 Histograms

Histograms are automatically created if you specify the x, but not the y aesthetic to cplot (Figure 7 left). This sets *stat* to "bin" and *geom* to "bar".

```
> cplot(X, ~f.tot.y, subset = t.frame == 13)
```

As with other plots, you can map other aesthetics to variables. In *geom* "bar" the *color* aesthetic refers to the border of the bar. The *fill* aesthetic specifies the filling color (Figure 7 right).

```
> cplot(X, ~f.tot.y, subset = t.frame == 13, fill = factor(AF.nM))
```

Note that we transformed the numerical variable AF.nM to a factor. This was done because the color scale applied to factors makes it easier to differentiate between levels of the variable. You can see that the title of the legend contains is "factor(AF.nM)". To avoid the inclusion of "factor" in this legend you can use the *as.factor* argument (see next example). When specifying the filling color, the *position* adjustment is set to "stack", which means that bars of different colors are stacked one over the other (the outline of the histogram doesn't change). This behavior can be changed by using other *position* adjustments, for example "dodge" and "fill" are shown in Figure 8, left and right respectively.

If you use *position* "identity" the histograms are plotted one over an other, so you will probably just see the last one to be plotted. You can avoid this by using *geom* "step" (Figure 9 left). If you do so you have to specify the *stat* to "bin" as this is not the default.

```
> cplot(X, x = f.tot.y, subset = t.frame == 7, color = AF.nM, as.factor = "AF.nM",
+     geom = "step", stat = "bin")
```

A smoothed version of this plot is obtained with a density plot, as shown in Figure 9 right.

```
> cplot(X, x = f.tot.y, subset = t.frame == 7, geom = "density", color = factor(AF.nM),
+     yzoom = c(0, 1e-06))
```

References

Leland Wilkinson The Grammar of Graphics *Statistics and Computation. Springer*, 2005

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.

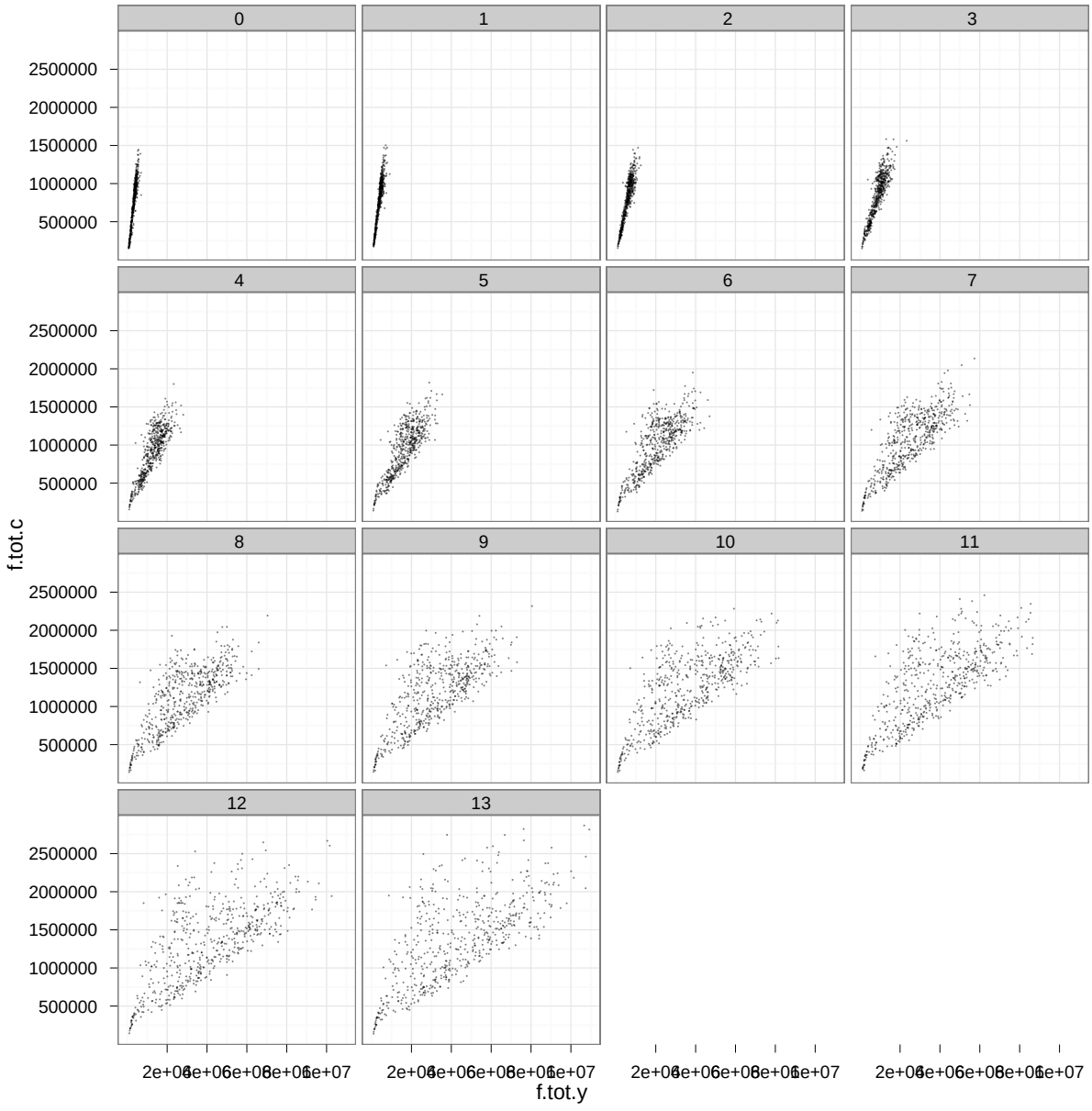

Figure 5: CFP vs YFP scatter plot, facetted by time frame.

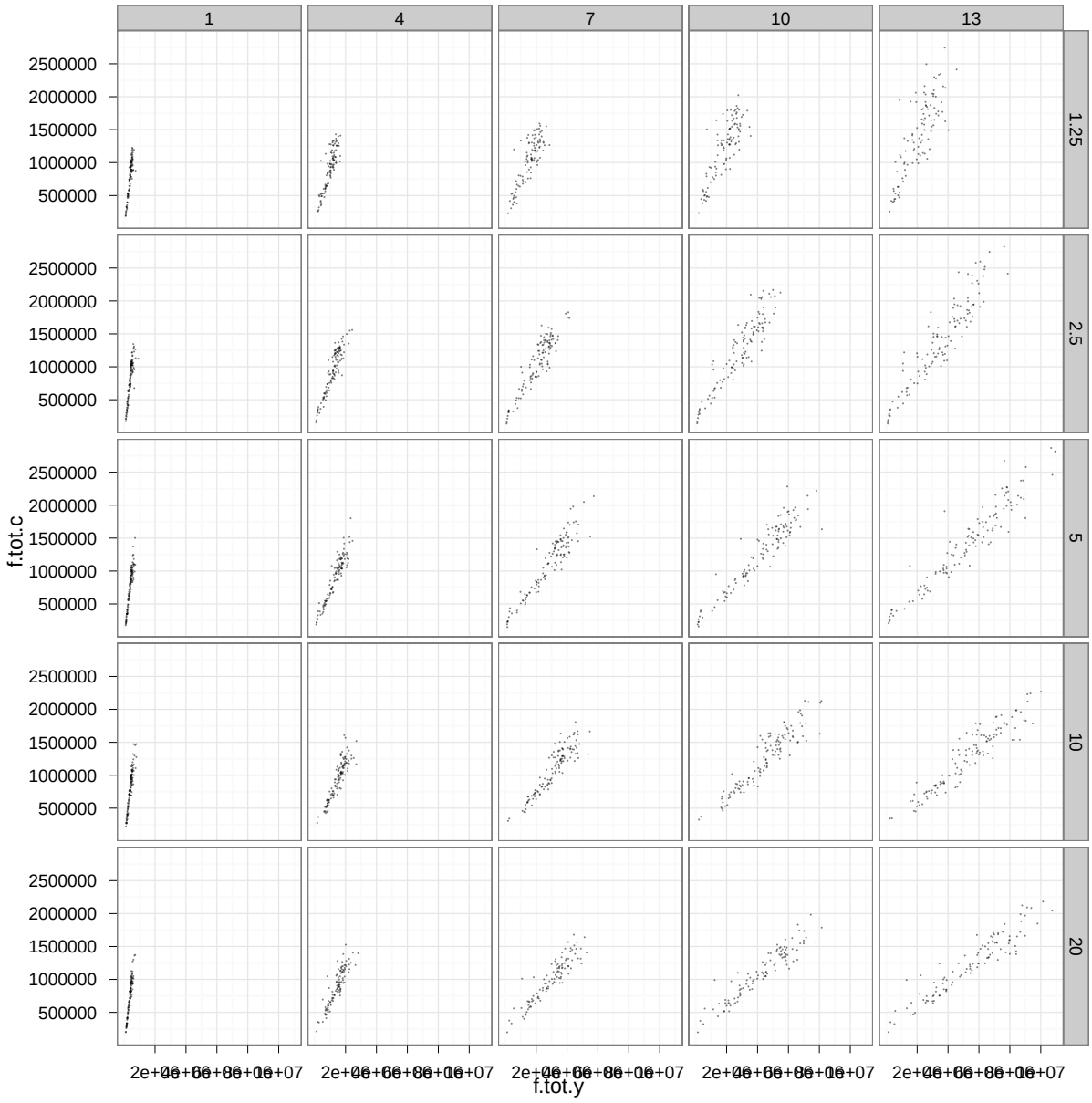

Figure 6: CFP vs YFP scatter plot, facetted by time frame and pheromone dose.

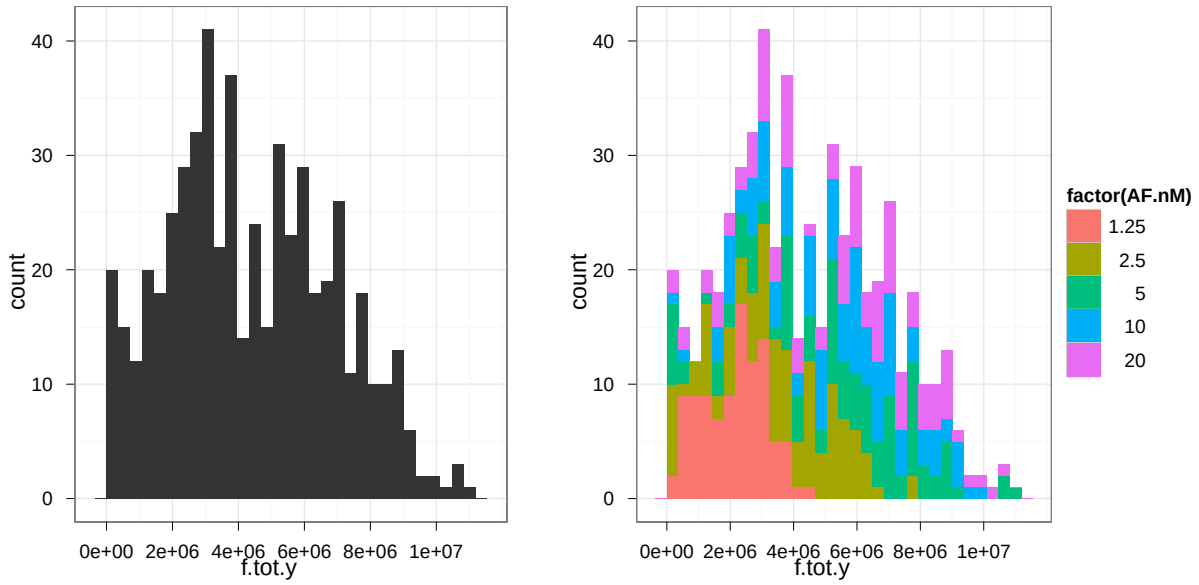

Figure 7: Left: histogram for total YFP fluorescence. Right: Same histogram, with fill aesthetic mapped to pheromone dose (AF.nM)

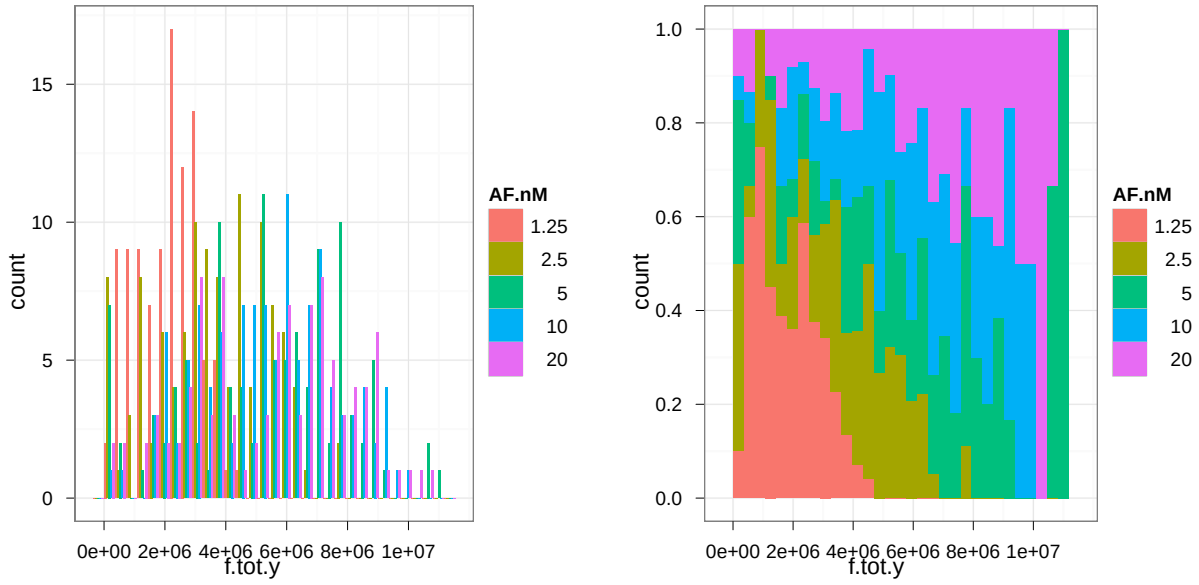

Figure 8: Left: histogram for total YFP fluorescence, using position "dodge". Right: Same histogram, with position "fill"

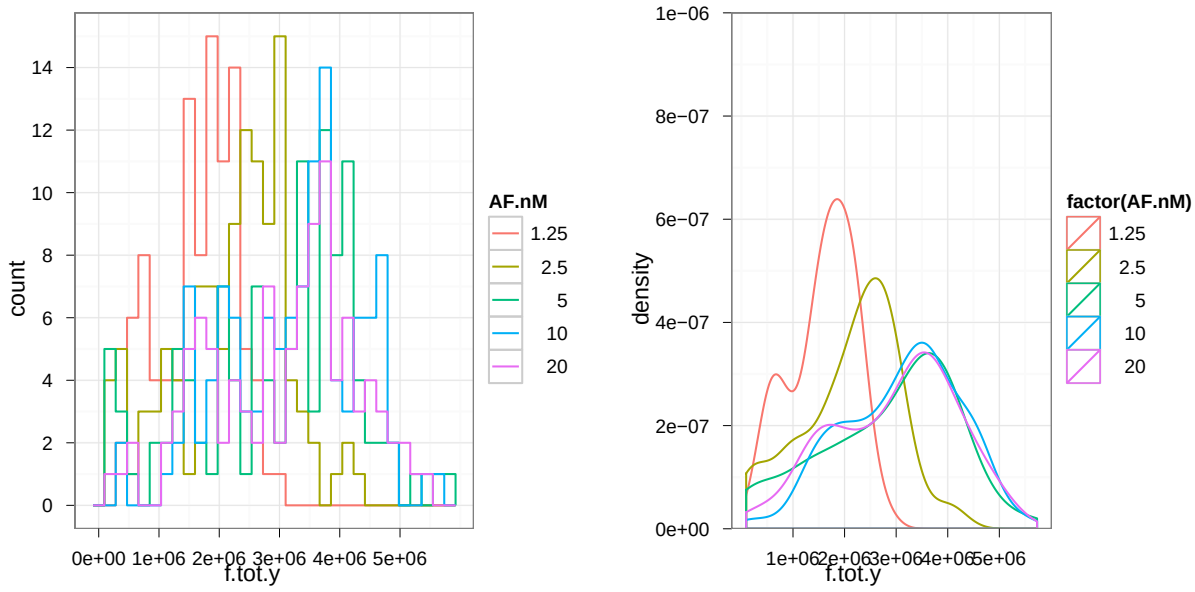

Figure 9: Left: histogram for total YFP fluorescence, using geom "step" and stat "bin". Right: Same histogram, with geom "density"