Beginning Lysate Array Doc

Keith Baggerly

June 27, 2006

Contents

1 Introduction

The *SuperCurve* package provides tools for the analysis of reverse-phase protein arrays (RPPAs), which are also known as “tissue lysate arrays” or simply “lysate arrays”.

1.1 The Biology of RPPAs

RPPAs resulted from an attempt to extend the microarray approach to the measurement of proteins. A microarray is “forward-phase” in that it simultaneously measures the expression levels of many genes in one biological sample. An RPPA is “reverse-phase” in that it simultaneously measures the expression levels of one protein in many biological samples.

The biological samples of interest are lysed, producing a homogeneous mixture (lysate), and these lysates are printed onto an array according to a dilution series. The arrays are typically glass with a nitrocellulose membrane on one side; the lysates are printed on the nitrocellulose.

In order to measure the protein of interest, the array is first interrogated with an antibody specific to the protein of interest (the primary antibody, typically derived from a mouse, rabbit or goat). This is allowed to bind, and loose material is washed away. The array is then interrogated with a labeled antibody (a secondary antibody, such as anti-goat immunoglobulin) which recognizes the primary antibody. This is allowed to bind, and loose material is washed away. In the most common labeling approach, the secondary antibody is linked to an enzyme, as with enzyme-linked immunosorbent assays (ELISAs). The enzyme substrate is then introduced. The enzyme reacts with its substrate, causing precipitate to build up near the site of the reaction: more of the protein of interest at a spot means more enzyme should bind and more precipitate should form. After a short period, the loose substrate is washed away. After drying, the array is then imaged, typically with a flatbed scanner, producing a TIF image file. (Ideally, the TIF file should be 16-bit grayscale. We have encountered cases where the files were exported as 24-bit truecolor, and then converted to grayscale afterwards. Depending on the software used, this latter step can introduce substantial distortions.) The printed spots visible in the image file are then quantified using software developed for cDNA microarrays.

Several other methods of labeling the secondary antibody have been tried, including fluorescent dyes and quantum dots, but all methods still yield a TIF image file which is then quantified.

1.2 The SuperCurve Model

A key distinction between reverse-phase and forward-phase assays is that for reverse-phase assays the hybridization kinetics should be the same at every spot, as all samples are being queried for the same protein. Thus, in the case of lysate arrays, we expect there to be a single common dose-response curve, instead of a separate one for each sample. In particular, we can borrow strength across samples for the estimation of baseline and saturation intensity.

We assume that the observed intensity for sample i, dilution step j, replicate k can be fit as

$$y_{ijk} = \alpha + \beta * g(\gamma(\delta_i + x_{ij})) + \epsilon_{ijk},$$

where $g(x) = e^x/(1 + e^x)$. The shape paramters of the logistic response curve, α, β, and γ are common for all samples. The x_{ij} are known offsets from the level of interest, such as the undiluted or "neat" state. We typically use $\log_2$ units for x_{ij}, letting the adjustment to base e be subsumed into γ. The δ_i terms represent the unknown true protein concentration at the reference level for sample i. Finally, ϵ_{ijk} is taken to be white noise.

We fit the above model using `nls` iteratively, alternating between fitting the shape parameters and the sample concentrations.

1.3 The Classes

There are 3 key classes in the SuperCurve package:

- `RPPA`, representing the quantification of an array,
- `RPPADesign`, specifying where each sample/dilution step combination is printed on the array, and

- `RPPAFit`, the results of fitting the SuperCurve model to an `RPPA`, `RPPADesign` combination.

More details can be found in the documentation for `RPPA`, `RPPA-class`, `RPPADesign`, `RPPADesign-class`, `RPPAFit`, and `RPPAFit-class`, but we will attempt to cover the high points below.

2 A Detailed Example: Breast Cancer Cell Lines

We begin with an example where we were attempting to measure the relative abundances of several proteins within a panel of 40 breast cancer cell lines. The proteins were chosen largely from the PI3K pathway, which is frequently disregulated in tumors. We're going to focus on 3 arrays from this study, corresponding to the proteins AKT, β-catenin, and ERK2. These data are included as the `rppaCell` data file, described as `rppaCellData`.

2.1 Getting Started

```
> library(SuperCurve)
```

```
Loading required package: MASS
```

```
> data(rppaCell)
> ls()
```

```
[1] "akt"      "c.erk2"   "ctnnb1"   "design40"
```

There are two types of objects in the data file. First, we have an `RPPADesign` object, `design40`, giving the layout of the array. For each spot, this specifies what sample has been printed, and what the dilution level offset of that spot is (in $\log_2$ units) relative to the reference level we're trying to estimate. Second, we have several `RPPA` objects, `akt`, `c.erk2`, and `ctnnb1`, each containing the spot-level quantifications supplied by the microarray image quantification software. These arrays were quantified using MicroVigene.

2.2 RPPA Details

If you'd like to take a look at the initial text files with the quantifications, feel free; these are also included. These also allow us to illustrate how an `RPPA` object can be constructed from a MicroVigene txt file:

```
> superHome <- system.file(package = "SuperCurve")
> aktHome <- "rppaCellData/Akt.txt"
> aktTake2 <- RPPA(file = aktHome, path = superHome)
> class(aktTake2)
```

```
[1] "RPPA"
attr(,"package")
[1] "SuperCurve"
```

This RPPA object is the same as akt. Now, there's really not that much to an RPPA object:

```
> slotNames(aktTake2)

[1] "data" "file"

> aktTake2@file

[1] "rppaCellData/Akt.txt"
```

just the data frame containing the quantifications and the file the data was acquired from.

Now, we've seen what the file slot contains, but what about data? As mentioned above, it's simply a data frame, with named components corresponding for the most part to measurements associated with each spot. Several of the functions that act on RPPA objects have a "measure" argument, and they're simply looking for the name of the appropriate component. What do we have here?

```
> summary(aktTake2)

An RPPA object loaded from rppaCellData/Akt.txt

    Main.Row        Main.Col        Sub.Row         Sub.Col              Sample   
 Min.   :1.00   Min.   : 1.0   Min.   :1.00   Min.   :1.00   sample 39: 16  
 1st Qu.:1.75   1st Qu.: 3.0   1st Qu.:1.75   1st Qu.:1.75   sample 40: 16  
 Median :2.50   Median : 5.5   Median :2.50   Median :2.50   sample1  : 16  
 Mean   :2.50   Mean   : 5.5   Mean   :2.50   Mean   :2.50   sample10 : 16  
 3rd Qu.:3.25   3rd Qu.: 8.0   3rd Qu.:3.25   3rd Qu.:3.25   sample11 : 16  
 Max.   :4.00   Max.   :10.0   Max.   :4.00   Max.   :4.00   sample12 : 16  
                                                             (Other)  :544  
    Mean.Net        Mean.Total      Median.Net        Vol.Bkg          Vol.Dust
 Min.   :  279   Min.   :  783   Min.   :  210   Min.   : 23520   Min.   :0  
 1st Qu.: 2506   1st Qu.: 4176   1st Qu.: 2455   1st Qu.: 66936   1st Qu.:0  
 Median : 6585   Median : 8916   Median : 7119   Median : 88560   Median :0  
 Mean   : 8022   Mean   :10210   Mean   : 8525   Mean   :104179   Mean   :0  
 3rd Qu.:13042   3rd Qu.:15522   3rd Qu.:14078   3rd Qu.:119856   3rd Qu.:0  
 Max.   :22090   Max.   :25409   Max.   :23008   Max.   :594624   Max.   :0  

> names(aktTake2@data)

 [1] "Main.Row"   "Main.Col"   "Sub.Row"    "Sub.Col"    "Sample"    
 [6] "Mean.Net"   "Mean.Total" "Median.Net" "Vol.Bkg"    "Vol.Dust"  
```

Most of the columns have ready interpretations, but the first 6 here are special. An `RPPA` object must have `Main.Row`, `Main.Col`, `Sub.Row`, and `Sub.Col` to specify the position of the spot, `Sample` to tell us what the software thinks was printed, and `Mean.Net` as a background-corrected measure of spot intensity.

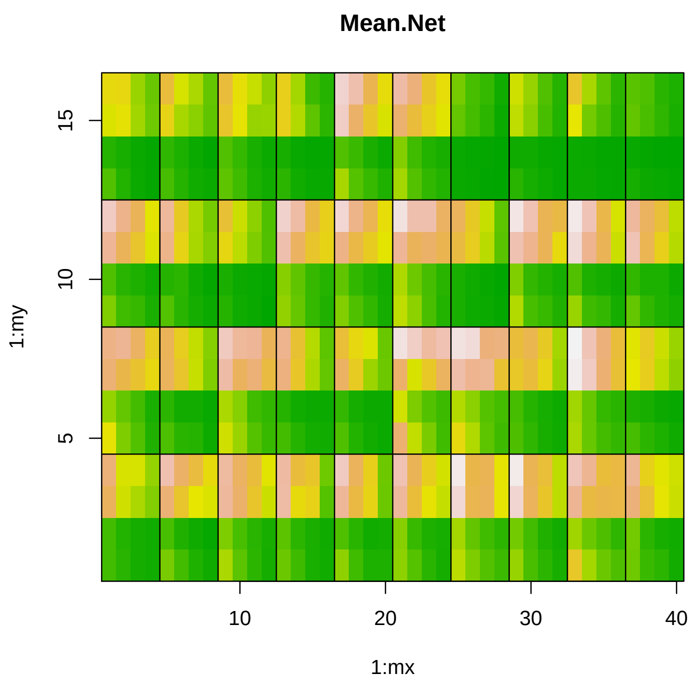
The constructor function is tuned for MicroVigene files at present. If we have other types of data files, we might want to assemble an `RPPA` object more directly.

```
> aktRead <- paste(superHome, aktHome, sep = "/")
> aktTemp <- read.table(file = aktRead, header = T, sep = "\t",
+     skip = 4, fill = T)
> names(aktTemp)[5] <- "Sample"
> names(aktTemp)[6] <- "Mean.Net"
> aktTake3 <- new("RPPA", data = aktTemp, file = aktRead)
```

again, this `RPPA` object is just the same as the others that we have assembled for AKT.

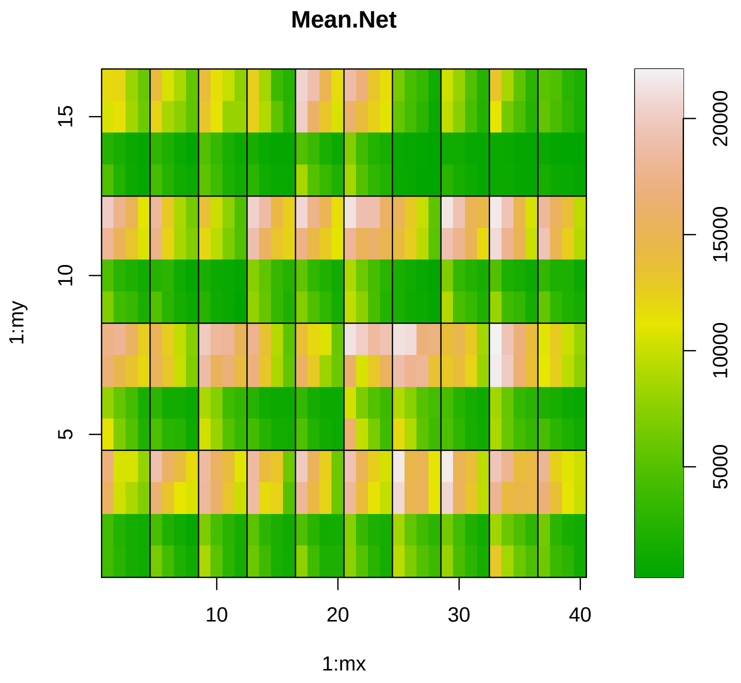
Given an `RPPA`, one of the first things we can do is simply *look* at the data, by checking heatmap images of the quantifications spatially arranged to match their positioning on the array.

```
> image(akt)
```



By default, image will use "Mean.Net" as the measure. Different spots on the array are separated by white lines, and different subgrids (patches) are separated by black lines. Here, we can see that the top 2 rows in each patch are different than the bottom 2 rows. To see how big the difference is, let's add a colorbar:

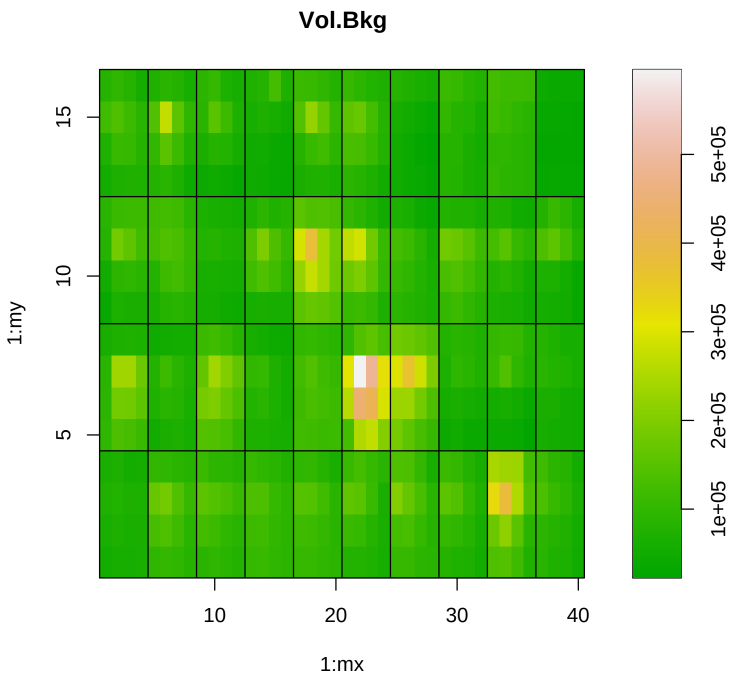
```
> image(akt, "Mean.Net", colorbar = TRUE)
```



The top 2 rows are decidedly brighter. This pattern is due to the design of the arrays used in this experiment. Every patch corresponds to a different sample, and the spots within a patch comprise two replicates of an 8-step dilution series (protein concentration decreases moving from left to right here). The top 2 rows are replicates of the first 4 dilution steps, and the bottom 2 rows are replicates of the last 4. We'll return to the design shortly.

In addition to measures of foreground intensity such as Mean.Net, measures of background (such as "Vol.Bkg") can often suggest potential problems. For example,

```
> image(akt, "Vol.Bkg", colorbar = TRUE)
```

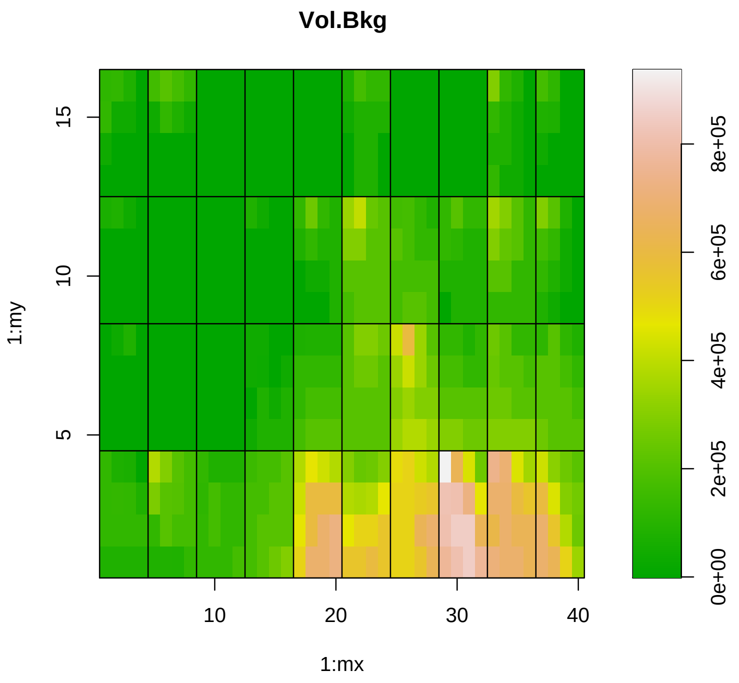


shows that something odd may have happened in the patch at Main.Row 3, Main.Col 6 (try calling `image` with "Main.Row" or "Main.Col" as the measure to get a better feel for the coordinates, if needed). Thus, we'll pay special attention to the results from this patch. We note in passing that the numbers for Vol.Bkg correspond to the estimated background at the spot times the number of pixels in the spot, and as such aren't directly comparable to the values from Mean.Net (especially as the size of the spot can vary). The scaled background values correspond to the difference between Mean.Total and Mean.Net. If desired, we can explore the exact relationship further:

```
> attach(akt@data)
> plot(Vol.Bkg/(Mean.Total - Mean.Net))
> detach("akt@data")
```

Similar checks of the background for the other `RPPA` objects show that there was a problem affecting the bottom edge of the β-catenin slide, `ctnnb1`.

```
> image(ctnnb1, "Vol.Bkg", colorbar = TRUE)
```



Estimates for samples in these regions are decidedly suspect.

2.3 RPPADesign Details

Now, the `RPPA` objects are only part of the story; we can't proceed to quantify the different samples unless we know the layout of the samples on the array. This information is contained in an `RPPADesign` object, here `design40`. Let's take a closer look at that.

```
> class(design40)

[1] "RPPADesign"
attr(,"package")
[1] "SuperCurve"

> slotNames(design40)

[1] "layout"    "alias"     "sampleMap" "controls"

> class(design40@layout)

[1] "data.frame"
```

The RPPADesign object has 4 slots: layout, alias, sampleMap, and controls. Of these, the most important is layout, which is a data frame specifying what is printed at each spot. If we take a look at the contents of the layout,

```
> names(design40@layout)
```

```
[1] "Main.Row" "Main.Col" "Sub.Row"  "Sub.Col"  "Sample"   "Steps"    "Series"
```

most of the terms look familiar. Main.Row, Main.Col, Sub.Row and Sub.Col specify the position of the spot on the array. Sample gives the name of the biological sample printed at the spot. Steps gives the dilution step of the spot within the sample in terms of a $\log_2$ offset relative to a reference point within the series. Finally, Series lets us subset the measurments within a sample, if desired. To make this clearer, let's take a look at the first few entries.

```
> design40@layout[1:17, ]
```

```
   Main.Row Main.Col Sub.Row Sub.Col  Sample Steps       Series
1         1        1       1       1 sample1   3.5 sample1.Rep1
2         1        1       1       2 sample1   2.5 sample1.Rep1
3         1        1       1       3 sample1   1.5 sample1.Rep1
4         1        1       1       4 sample1   0.5 sample1.Rep1
5         1        1       2       1 sample1   3.5 sample1.Rep2
6         1        1       2       2 sample1   2.5 sample1.Rep2
7         1        1       2       3 sample1   1.5 sample1.Rep2
8         1        1       2       4 sample1   0.5 sample1.Rep2
9         1        1       3       1 sample1  -0.5 sample1.Rep1
10        1        1       3       2 sample1  -1.5 sample1.Rep1
11        1        1       3       3 sample1  -2.5 sample1.Rep1
12        1        1       3       4 sample1  -3.5 sample1.Rep1
13        1        1       4       1 sample1  -0.5 sample1.Rep2
14        1        1       4       2 sample1  -1.5 sample1.Rep2
15        1        1       4       3 sample1  -2.5 sample1.Rep2
16        1        1       4       4 sample1  -3.5 sample1.Rep2
17        1        2       1       1 sample2   3.5 sample2.Rep1
```

For this array, the first 16 spots comprise the patch in the upper left hand corner, and all of these spots are derived from sample 1. The reference point was taken to be midway through the 8-step dilution series, so the most intense (undiluted) spots have a step value of 3.5. The replicate measurements for this sample have been arbitrarily grouped into two distinct "series", in part as a consistency check. When we estimate protein concentrations, we produce an estimate for each series; series from the same sample should yield similar values. The pattern shown here is repeated for the other patches on the array.

The trickiest part of constructing an RPPADesign object is often the specification of the dilution pattern and the grouping into series. For example,

```
> steps <- rep(c(rep(8:5, 2), rep(4:1, 2)), 40) - 4.5
> rep.temp <- factor(paste("Rep", rep(rep(1:2, each = 4), 80),
+     sep = ""))
> series <- factor(paste(as.character(akt@data$Sample), as.character(rep.temp),
+     sep = "."))
> design40Take2 <- RPPADesign(akt, steps = steps, series = series)
```

This version of the constructor extracts some of the needed layout information from the RPPA object itself. Now, we can construct an RPPADesign using a more low-level approach, such as reading the Sample information (and possibly the dilution levels as well) from a separate file.

```
> superHome <- system.file(package = "SuperCurve")
> aktHome <- "rppaCellData/Akt.txt"
> aktRead <- paste(superHome, aktHome, sep = "/")
> aktTemp <- read.table(file = aktRead, header = T, sep = "\t",
+     skip = 4, fill = T)
> names(aktTemp)[5] <- "Sample"
> aktTemp <- aktTemp[, names(aktTemp) %in% c("Main.Row", "Main.Col",
+     "Sub.Row", "Sub.Col", "Sample")]
> steps <- rep(c(rep(8:5, 2), rep(4:1, 2)), 40) - 4.5
> rep.temp <- factor(paste("Rep", rep(rep(1:2, each = 4), 80),
+     sep = ""))
> series <- factor(paste(as.character(aktTemp$Sample), as.character(rep.temp),
+     sep = "."))
> aktLayout <- data.frame(aktTemp, Steps = steps, Series = series)
> aktNames <- levels(aktLayout$Sample)
> aktAlias <- data.frame(Alias = aktNames, Sample = aktNames)
> aktSampleMap <- as.vector(tapply(as.character(aktLayout$Sample),
+     list(series), function(x) x[[1]]))
> names(aktSampleMap) <- levels(aktLayout$Series)
> design40Take3 <- new("RPPADesign", layout = aktLayout, alias = aktAlias,
+     sampleMap = aktSampleMap)
```

Rather ornate, but it pulls things together.

2.4 RPPAFit Details

Given an RPPADesign and an RPPA, we can use a model to estimate the protein concentrations, producing an RPPAFit. The fitting function requires that we specify the measure to be used, in addition to the RPPA and RPPADesign.

```
> aktFit <- RPPAFit(akt, design40, "Mean.Net")
> class(aktFit)
```

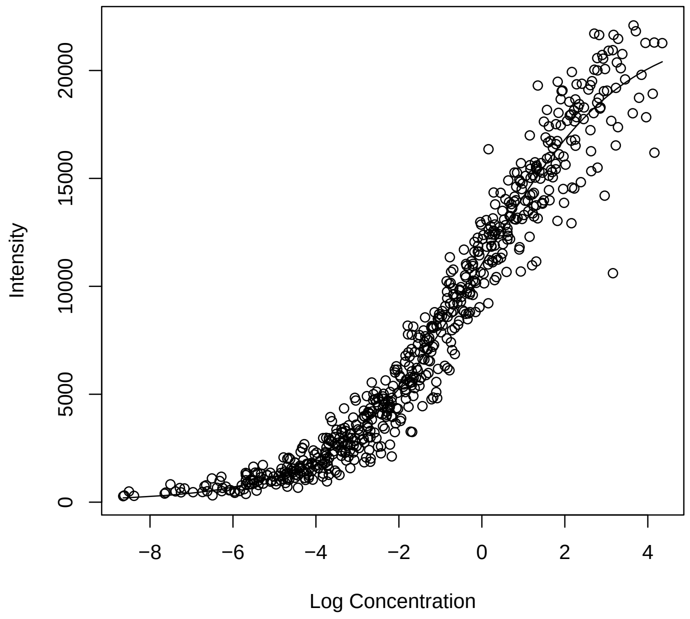
```
[1] "RPPAFit"
attr(,"package")
[1] "SuperCurve"
```

```
> slotNames(aktFit)

 [1] "call"           "rppa"           "design"         "measure"
 [5] "method"         "coefficients"   "concentrations" "lower"
 [9] "upper"          "conf.width"     "intensities"    "linear.p"
[13] "slope"          "p.values"       "ss.ratio"       "warn"
[17] "version"

> aktFit@call

RPPAFit(rppa = akt, design = design40, measure = "Mean.Net")

> aktFit@version

[1] "0.931"
```

This fits the basic SuperCurve model: a logistic dose response curve common to all samples, with a separate offset term for each series (not sample).

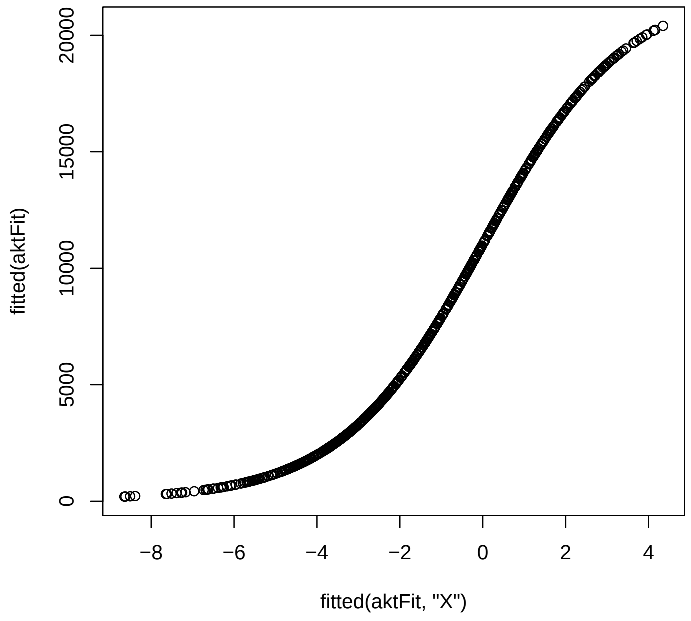
We can get a quick feel for the shape of the curve by looking at a “cloud” plot showing the observed intensity and estimated log protein concentration for each spot:

```
> plot(aktFit)
```



The data appears to follow a curve pretty well here. We can get a better idea of how good the fit is by decomposing the observed values into "fitted" + "residuals". In looking at the fitted values, we need to keep in mind that the model we are using results in fitted values for the observations in terms of both intensity (the default, or "Y") and log concentration ("X").

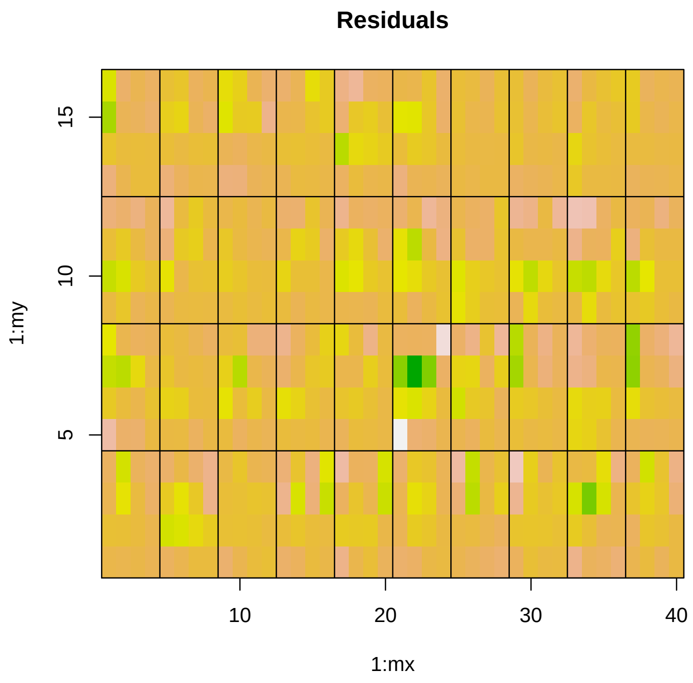
```
> plot(fitted(aktFit, "X"), fitted(aktFit))
> plot(fitted(aktFit, "X"), resid(aktFit))
```



There is clearly an increase in the variability of the residuals as a function of the estimated log concentrations. This version of the package does not address this issue.

In any event, there is at least one clear outlier in the residuals. We may be able to understand this better if we look at the residuals arranged spatially:

```
> image(aktFit)
```



The most extreme residuals are located in the patch in Main.Row 3, Main.Col 6, which we had already flagged for attention based on odd behavior in the background.

We can also look at fits of each individual series to the underlying response curve:

```
> oldAsk <- par(ask = TRUE)
> plot(aktFit, type = "individual")
> par(oldAsk)
```

As this produces a lot of figures, we have chosen to not reproduce them here.

So, what was the estimated concentration for the first series? We can find out in a few different ways.

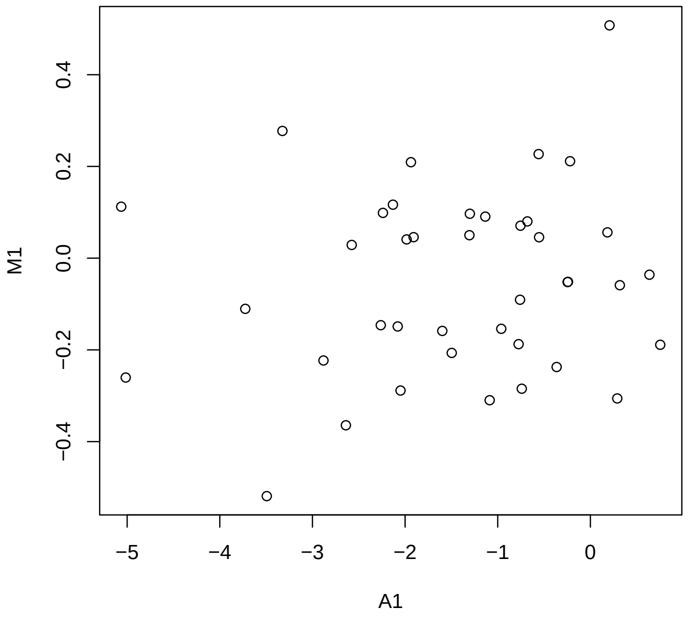
```
> aktFit@concentrations[1]

sample1.Rep1
   -2.590578

> aktFit@concentrations["sample1.Rep1"]

sample1.Rep1
   -2.590578
```

Individual elements are named, so we can readily extract information about the samples of interest. Similarly, we can get the parameters for the fitted model in a few different ways.

```
> coefficients(aktFit)

 alpha.alpha    beta.beta        gamma
   60.077188 21944.853236     0.584724

> coef(aktFit)

 alpha.alpha    beta.beta        gamma
   60.077188 21944.853236     0.584724

> aktFit@coefficients

 alpha.alpha    beta.beta        gamma
   60.077188 21944.853236     0.584724
```

Now, at this point we've computed fits by series. We can use this to give us some idea of the stability of the results for a sample by doing the equivalent of an MA-plot, plotting the difference in replicates as a function of their average.

```
> M1 <- (aktFit@concentrations[seq(2, 80, 2)] - aktFit@concentrations[seq(1,
+     80, 2)])
> A1 <- (aktFit@concentrations[seq(2, 80, 2)] + aktFit@concentrations[seq(1,
+     80, 2)])/2
> plot(A1, M1)
```



Eyeballing the fit suggests a standard deviation of about 0.2 (in $\log_2$ units). Given the x-range, this is acceptable. Of course, if we know that replicates agree fairly well, we'd like to go back and fit the results for each sample without splitting things up!

The main thing this requires is a slightly different `RPPADesign`, grouping all spots from a single sample together.

```
> steps <- rep(c(rep(8:5, 2), rep(4:1, 2)), 40) - 4.5
> rep.temp <- factor(paste("Rep", rep(rep(1:2, each = 4), 80),
+     sep = ""))
> series <- akt@data$Sample
> design40Sample <- RPPADesign(akt, steps = steps, series = series)
```

Now, we simply rerun the fit

```
> aktFitSample <- RPPAFit(akt, design40Sample, "Mean.Net")
```