Quantifying Proteins 2: Lysate Arrays

Keith Baggerly

June 16, 2006

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".

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`.

Getting Started

```
> options(width = 60)
> 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` (for 40 cell lines), 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.

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
 Min.   :1.00   Min.   : 1.0   Min.   :1.00
 1st Qu.:1.75   1st Qu.: 3.0   1st Qu.:1.75
 Median :2.50   Median : 5.5   Median :2.50
 Mean   :2.50   Mean   : 5.5   Mean   :2.50
 3rd Qu.:3.25   3rd Qu.: 8.0   3rd Qu.:3.25
 Max.   :4.00   Max.   :10.0   Max.   :4.00

    Sub.Col           Sample       Mean.Net
 Min.   :1.00   sample 39: 16   Min.   :  279
```

```
1st Qu.:1.75   sample 40: 16   1st Qu.: 2506
Median :2.50   sample1  : 16   Median : 6585
Mean   :2.50   sample10 : 16   Mean   : 8022
3rd Qu.:3.25   sample11 : 16   3rd Qu.:13042
Max.   :4.00   sample12 : 16   Max.   :22090
               (Other)  :544
  Mean.Total      Median.Net       Vol.Bkg
Min.   :  783   Min.   :  210   Min.   : 23520
1st Qu.: 4176   1st Qu.: 2455   1st Qu.: 66936
Median : 8916   Median : 7119   Median : 88560
Mean   :10210   Mean   : 8525   Mean   :104179
3rd Qu.:15522   3rd Qu.:14078   3rd Qu.:119856
Max.   :25409   Max.   :23008   Max.   :594624

   Vol.Dust
Min.   :0
1st Qu.:0
Median :0
Mean   :0
3rd Qu.:0
Max.   :0
```

```
> names(aktTake2@data)
```

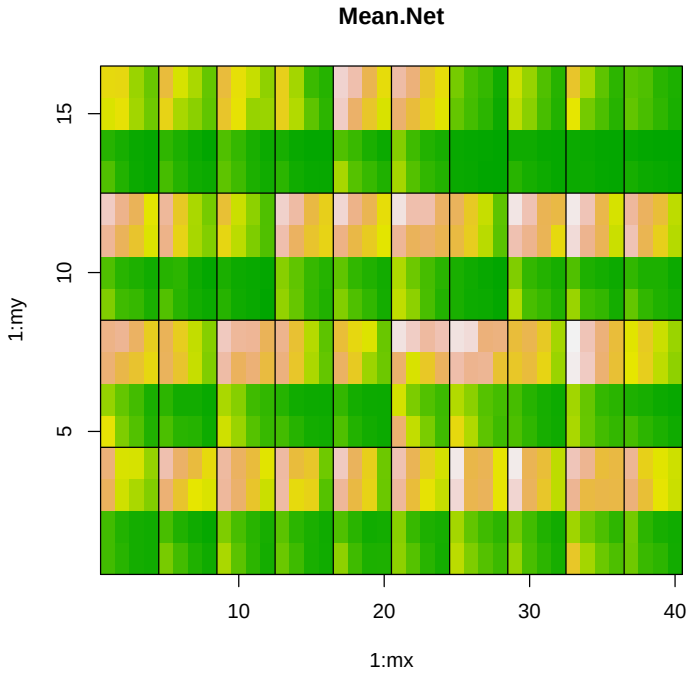
```
[1] "Main.Row"   "Main.Col"   "Sub.Row"    "Sub.Col"
[5] "Sample"     "Mean.Net"   "Mean.Total" "Median.Net"
[9] "Vol.Bkg"    "Vol.Dust"
```

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. The package vignette provides more details here.

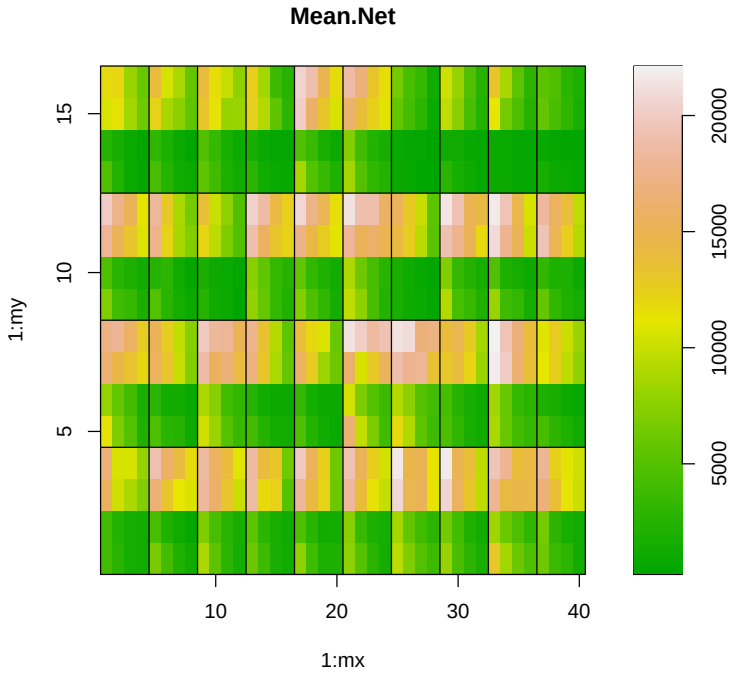
Given an `RPPA`, one of the first things we can do is simply *look* at the data, by checking heatmap images of the quantifications spatially arranged to match their positioning on the array.

```
> 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.Total or 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 appears to be 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.

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"
[6] "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 measurements within a sample, if desired. (This can be useful as a check for consistency, particularly if replicate series are printed in different patches on the array.) 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
1         1        1       1       1 sample1   3.5
2         1        1       1       2 sample1   2.5
3         1        1       1       3 sample1   1.5
4         1        1       1       4 sample1   0.5
5         1        1       2       1 sample1   3.5
6         1        1       2       2 sample1   2.5
7         1        1       2       3 sample1   1.5
8         1        1       2       4 sample1   0.5
9         1        1       3       1 sample1  -0.5
10        1        1       3       2 sample1  -1.5
11        1        1       3       3 sample1  -2.5
12        1        1       3       4 sample1  -3.5
13        1        1       4       1 sample1  -0.5
14        1        1       4       2 sample1  -1.5
15        1        1       4       3 sample1  -2.5
16        1        1       4       4 sample1  -3.5
17        1        2       1       1 sample2   3.5
         Series
1  sample1.Rep1
2  sample1.Rep1
3  sample1.Rep1
4  sample1.Rep1
5  sample1.Rep2
6  sample1.Rep2
7  sample1.Rep2
8  sample1.Rep2
9  sample1.Rep1
10 sample1.Rep1
11 sample1.Rep1
12 sample1.Rep1
13 sample1.Rep2
14 sample1.Rep2
15 sample1.Rep2
16 sample1.Rep2
17 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. I wouldn't try entering the code block below, but rather just skim it to see if the structure makes sense.

```
> 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.

RPPAFit Details

Given an RPPADesign and an RPPA, we can use a model to estimate the protein concentrations, producing an RPPAFit.

For our fitting, 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.

The fitting function `RPPAFit` 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"
 [4] "measure"        "method"         "coefficients"
 [7] "concentrations" "lower"          "upper"
[10] "conf.width"     "intensities"    "linear.p"
[13] "slope"          "p.values"       "ss.ratio"
[16] "warn"           "version"
```

```
> aktFit@call
```

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

```
> aktFit@version
```

```
[1] "0.93"
```

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 the 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))
```

Q: Is the variance stable as a function of the mean? This version of the package does not address this issue.

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 when we looked at the `RPPA` 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)
```

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 (proportional to the standard deviation) 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
> series <- akt@data$Sample
> design40Sample <- RPPADesign(akt, steps = steps,
+     series = series)
```

Now, we simply rerun the fit

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

More Generic Goals

Using the `design40Sample` design object, fit `akt`, `c.erk2`, and `ctnnb1`. Extract the estimated sample concentrations and bind them into a matrix. Try this for both Mean.Total and Mean.Net. Describe the pros and cons of using `Mean.Net` and `Mean.Total` for `c.erk2`.

Cluster the samples using `hclust`. Now, add 10 to all of the `akt` measurements and subtract 10 from all of the `ctnnb1` measurements. Do your cluster results change? We only know the values for each protein up to an additive offset; we do not have absolute values. Any metric we use for clustering should give results that are invariant with respect to these types of changes. This means that different metrics are required for clustering samples and clustering proteins.

When we estimate the concentrations on the dilution curve, estimates where many of the spots are at extremely low (background) or extremely high (saturation) levels are very unstable. One way to correct for this is to add upper and lower bounds to the values that will be used. For example, if we extract the coefficients for the fitted logistic model, we can identify the concentration such that the fitted intensity is more than $\alpha + 0.99\beta$. If the estimated concentration

at the midpoint of the dilution series is above this cutoff (i.e., more than half of the observations are saturated), reset the estimate so that the midpoint is at the threshold noted. We can apply a similar heuristic for concentrations below $\alpha + 0.01\beta$. How would this rule alter the results for `akt`?

We've also included a second small data set, `rppaTumor`. Try fitting the arrays in `rppaTumor`. Note that the arrays used here have a different design. Do you see how the design could be assembled? Note that in this design, the `controls` slot of the design is not empty. What does it contain? Why does this matter?

We are extremely aware of the limitations of the current version of the package (it's not robust enough with respect to extreme outliers, for example). What functionality should we add to the package? Is there any we should remove?

There is one important issue that we don't know how to resolve within R. Specifically, we have found it vital to look at the grayscale TIFF images, and not just the quantifications, when trying to develop intuition about how the data works. Due to the nature of TIFF files, R does not have a TIFF reader. Eventually we'll get there! In the meantime, however, find an image viewer for your machine that you can be happy with.