The pepStat user guide

Raphael Gottardo[*], Greg Imholte[†], Mike Jiang[‡] and Renan Sauteraud[§]

May 8, 2013

A step-by-step guide in the analysis of peptide microarray antibody binding

Contents

[*] rgottard@fhcrc.org
[†] gimholte@uw.edu
[‡] wjiang2@fhcrc.org
[§] rsautera@fhcrc.org

1 Introduction

The pepStat package offers a complete analytical framework for the analysis of peptide microarray data. It includes a novel normalizatio n method to remove non-specific peptide binding activity of antibodies, a data smoothing reducing step to reduce background noise, and subject -specific positivity calls.

1.1 Installing the package

The pepStat package requires GSL, an open source scientific computing library. This library is freely available at http://www.gnu.org/software/gsl/.

1.2 Loading the package

As with any R package, it should first be loaded in the session.

```
library(pepStat)

## Loading required package:  Biobase
## Loading required package:  BiocGenerics
##
## Attaching package:  'BiocGenerics'
## The following object(s) are masked from 'package:stats':
##
## xtabs
## The following object(s) are masked from 'package:base':
##
## anyDuplicated, cbind, colnames, duplicated, eval, Filter,
## Find, get, intersect, lapply, Map, mapply, mget, order, paste,
## pmax, pmax.int, pmin, pmin.int, Position, rbind, Reduce,
## rep.int, rownames, sapply, setdiff, table, tapply, union,
## unique
## Welcome to Bioconductor
##
## Vignettes contain introductory material; view with
## 'browseVignettes()'.  To cite Bioconductor, see
## 'citation("Biobase")', and for packages 'citation("pkgname")'.
## Loading required package:  IRanges
```

2 Generating a peptideSet

2.1 Reading in .gpr files

The reading function takes a path as its argument and parses all the *.gpr* files in the given directory. Alternatively, one may specify a character vector of paths to individual *.gpr* files.

Optionally, one may provide a path to a *mapping file* giving annotation data for each slide, such as treatment status or patient information. If provided, the data set **must** be a `.csv` file and **must** include columns labeled `filename`, `ptid`, and `visit`. Elements in column `filename` must correspond to the filenames of slides to be read in, without the `.gpr` extension. Column `ptid` is a subject or slide identifier. Column `visit` indicates a case or control condition, such as pre/post vaccination, pre/post infection, or healthy/infected status. Control conditions must be labelled *pre*, while case conditions must be labelled *post*. Alternatively, one may input a `data.frame` satisfying the same requirements.

By default channels F635 Median and B635 Median are collected, and the `'normexp'` method of the `backgroundCorrect` function in the `limma` package corrects probe intensities for background fluorescence. Other methods may be selected, see documentation.

```
mapFile <- system.file("extdata/mapping.csv", package = "pepStat")
map <- read.csv(mapFile)
map
```

```
##   filename ptid visit treatment
## 1     f1_1    1   Pre   PLACEBO
## 2     f1_2    1  Post   PLACEBO
## 3     f2_1    2   Pre   PLACEBO
## 4     f2_2    2  Post   PLACEBO
## 5     f3_1    3   Pre   VACCINE
## 6     f3_2    3  Post   VACCINE
## 7     f4_1    4   Pre   VACCINE
## 8     f4_2    4  Post   VACCINE
```

```
dirToParse <- system.file("extdata/RVV", package = "pepStat")
list.files(dirToParse)
```

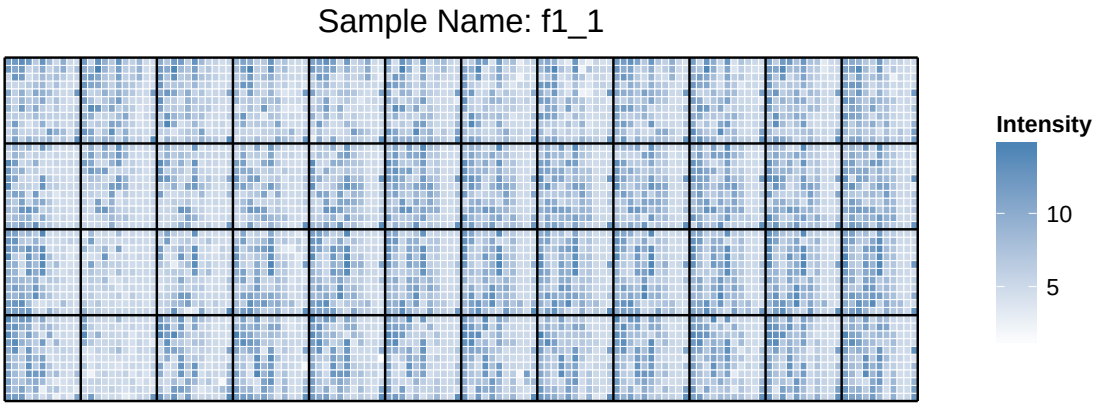
```
## [1] "f1_1.gpr" "f1_2.gpr" "f2_1.gpr" "f2_2.gpr" "f3_1.gpr" "f3_2.gpr"
## [7] "f4_1.gpr" "f4_2.gpr"
```

```
pSet <- makePeptideSet(files = NULL, path = dirToParse, mapping.file = mapFile,
    log = TRUE)
```

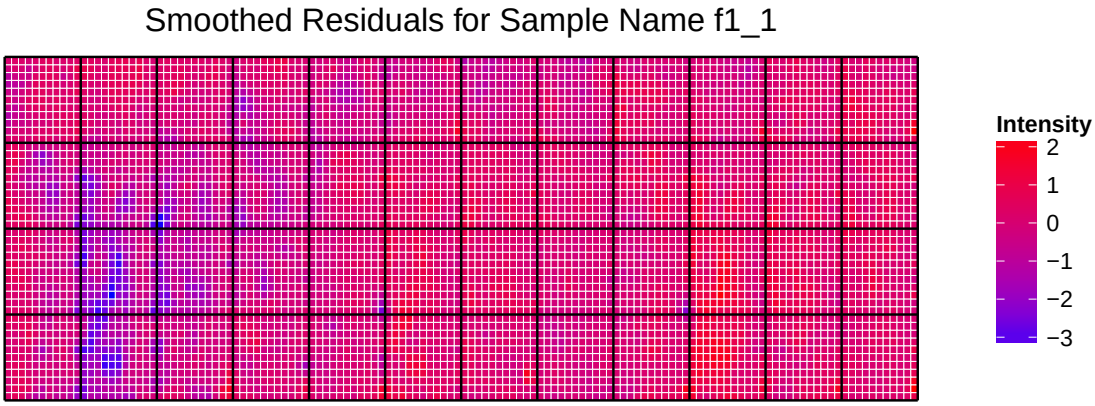
2.2 Visualize slides

We include a plotting function to detect spatial slide artifacts.

```
plotArrayImage(pSet, array.index = 1)
```




```
plotArrayResiduals(pSet, array.index = 1, smooth = TRUE)
```



2.3 Accessing `peptideSet` elements

`makePeptideSet` returns an object of class `peptideSet`, the base structure used in `pepStat`. It contains the sequence and ID of the peptides, measured intensities, annotations added through a mapping file, and probe slide position information. Various accessor functions can extract these values.

```
# peptide intensities
exprs(pSet)[1:5, 1:4]

##    f1_1   f1_2   f2_1   f2_2
## 1 11.89 10.356 11.770 11.444
## 2 10.61  9.485 10.665 10.872
```

```
## 3 11.25 10.356 11.366 11.639
## 4 11.10 10.007 11.270 11.278
## 5 10.79  9.033  9.658  9.291

# probe information
head(values(ranges(pSet))[[1]])

## DataFrame with 6 rows and 5 columns
##         featureID         peptide    block      row   column
##       <character>     <character> <factor> <factor> <factor>
## 1 WVTVYYGVPVWKDAE WVTVYYGVPVWKDAE        1        1        1
## 2 VYYGVPVWKEATTTL VYYGVPVWKEATTTL        1        1        2
## 3 VYYGVPVWRDAETTL VYYGVPVWRDAETTL        1        1        3
## 4 GVPVWRDADTTLFCA GVPVWRDADTTLFCA        1        1        4
## 5 VWKEAKTTLFCASDA VWKEAKTTLFCASDA        1        1        5
## 6 DAETTLFCASDAKAY DAETTLFCASDAKAY        1        1        6

# same as 'peptide' and 'featureID' columns above
head(peptide(pSet), 4)

## [1] "WVTVYYGVPVWKDAE" "VYYGVPVWKEATTTL" "VYYGVPVWRDAETTL" "GVPVWRDADTTLFCA"

head(featureID(pSet), 4)

## [1] "WVTVYYGVPVWKDAE" "VYYGVPVWKEATTTL" "VYYGVPVWRDAETTL" "GVPVWRDADTTLFCA"

# mapping file slide annotations
head(pData(pSet))

##      ptid visit treatment
## f1_1    1   pre   PLACEBO
## f1_2    1  post   PLACEBO
## f2_1    2   pre   PLACEBO
## f2_2    2  post   PLACEBO
## f3_1    3   pre   VACCINE
## f3_2    3  post   VACCINE
```

`preproc(pSet)` stores additional information such as slide layout, background correction methods, normalization, transformation, etc .

2.4 Summarizing within-slide replicates

The function `summarizePeptides` summarizes within-slide replicates by either their mean or median. Additional peptide sequence and/or annotation information may be incorporated with a `RangedData` object from the `IRanges` package. In this example, we use pep_hxb2 available in the `PEP.db` package.

```
library(PEP.db)
data(pep_hxb2)
psSet <- summarizePeptides(pSet, summary = "mean", position = pep_hxb2)

## Some peptides have no match in the RangedData object rownames and are removed from
the peptideSet!
```

pep_hxb2 gives information regarding the position of each peptide, their z-scores, the clades they belong to and the alignment with the reference sequence HXB2.

3 Normalizing the peptideSet

The primary goal of the data normalization step is to remove non-biological source of bias and increase the comparability of true positive signal intensities across slides. The method developped for this package uses physiochemical properties of individual peptides to model non-specific antibody binding to arrays.

```
pnSet <- NormalizeArray(psSet)
```

An object of class `peptideSet` containing the corrected peptides intensities is returned.

4 Data smoothing

The optional data smoothing step takes advantage of the overlapping nature of the peptides on the array to remove background noise caused by experimental variation. It is likely that two overlapping peptides will share common binding signal, when present. `pepStat` use a sliding mean technique technique to borrow strength across neighboring peptides and to reduce signal variability. This statistic increases detection of binding *hotspots* that noisy signals might otherwise obscure. Peptides are smoothed according to their sequence alignment position, taken from `position(psSet)`.

```
psmSet <- slidingMean(pnSet, width = 9)
```

5 Making calls

The final step is to make the positivity calls. The function `makeCalls` automatically uses information provided in the mapping file, accessed via `pData(pSet)`. It detects whether samples are paired or not. If samples are paired, POST intensities are subtracted from PRE intensities, then thresholded. Otherwise, PRE samples are averaged, and then subtracted from POST intensities. These corrected POST intensities are thresholded.

The `freq` argument controls whether we return the percentage of responders against each peptide, or a matrix of subject specific call . When `freq` is `TRUE`, we may supply a `group` variable from `pData(psmSet)` on which we split the frequency calculation.

```
V_calls <- makeCalls(psmSet, freq = TRUE, group = "treatment", cutoff = 0.1,
    method = "FDR", verbose = TRUE)

## You have paired PRE/POST samples
```