

## BioConductor Cheat Sheet

---

BioConductor (<http://www.bioconductor.org>) is a large set of libraries for analyzing bioinformatics data in R; the focus is on microarray data. BioConductor is big, really really big, and complex. We could run a whole other course just for BioConductor. Having said that, apparently everyone and their horse wants to use R for microarray analysis, so Bioconductor's the way to go.

BioConductor introduces many new data structures (objects) into R; the two we'll be talking about are **AffyBatch** and **exprsSet** objects.

### Loading Affymetrix CEL files into R

---

The raw probe intensity data from Affymetrix GeneChip trials is stored in a CEL file. To read CEL files into AffyBatch objects, you'll need the “**affy**” library installed and loaded. The main function is “**ReadAffy**”.

AffyBatch objects contain the raw probe data for one or more Affymetrix GeneChip trials; these trials are expected to be part of a single experiment. AffyBatch objects also contain a table of phenotypic data (phenoData) indicating what phenotypic groups each trial is part of, for example “malignant” or “nonmalignant.” Please note that AffyBatch objects do not contain any analyzed data—a list of genes, for example—just probes and measured intensities. Further processing is needed to extract meaningful data from an AffyBatch.

### Cleaning and calculating gene expression

---

After loading the raw probe intensity data into an AffyBatch object, preliminary data “cleaning” is done to compensate for experimental anomalies and irrelevant differences between trials. In BioConductor, this cleaning comprises multiple steps:

- Background adjustment: adjust probes against all probes on same chip.
- Normalization: adjust probe against same probes on all other chips.
- PM correction: adjust PM against MM values.
- Summarization: combine probes representing the same gene or expressed sequence.

Each of these steps is optional, although you'll almost certainly want to do some kind of summarization. You may use one of several predefined methods for each step, including implementations of Affymetrix's algorithms, or you

can always create your own. After summarization, a list of gene expression levels is produced. These gene expression levels are stored in an `exprsSet` object.

There is a single all-in-one function that does all of the cleaning and produces an `exprsSet` object from a `AffyBatch` object: “**expresso**”. It is defined in the “**affy**” library. The `exprsSet` object is not easy to get expression data out of manually; to get a matrix of expression levels from an `exprsSet` object, use the “**exprs**” function.

## **Analysis of differential expression**

---

Having derived our expression data, it seems like a pretty safe bet that we probably want to analyze it. The most common goal is to find genes that are differentially expressed between two groups in the experiment. The “**limma**” package (linear models on microarrays) provides tools to do this, in stages:

- The “**lmFit**” function fits expression to a linear model for each gene and also calculates the residual error.
- The “**eBayes**” function calculates the likelihood that a residual error would be seen by chance, and hence the likelihood that the gene is expressed differentially.
- The “**topTable**” function produces a table of the top N genes (10 by default) that appear to be differentially expressed.

Each of these functions takes the result of the previous as its input, so you’d write something like (and this is just a sketch):

```
fitted <- lmFit(exprData, ...)
calculated <- eBayes(fitted)
top10 <- topTable(calculated, ...)
```

The top table produced by “**topTable**” is just a data frame and can be examined normally. To get human-readable descriptions of the Affymetrix gene IDs seen in the table, use the “**aafDescription**” function found in the “**annaffy**” package.

## **GO analysis**

---

Another useful technique is to see if the differentially expressed genes have anything in common. We can do this by finding what Gene Ontology (GO) categories the genes are in and seeing if any categories are overrepresented in our sample of genes. The functions for this are in the “**GOstats**” package.

To find overexpressed GO categories, we must follow the following steps:

- Convert Affymetrix IDs to LocusLink IDs using the “**aafLocusLink**” function from the “**annaffy**” package.
- Use these IDs with the “**GOHyperG**” function from the “**GOstats**” package. This function decides what GO categories are present and overrepresented.
- You can see the assigned p-values for each identified category by getting the “**pvalues**” column from the result of the GOHyperG function, as in the example below:

```
X <- GOHyperG(locusLinkIDs, ...)  
X$pvalues
```

This should be enough information to get you started on using BioConductor.  
Good luck!