From CEL Files to Annotated Lists of Genes (Part 1)

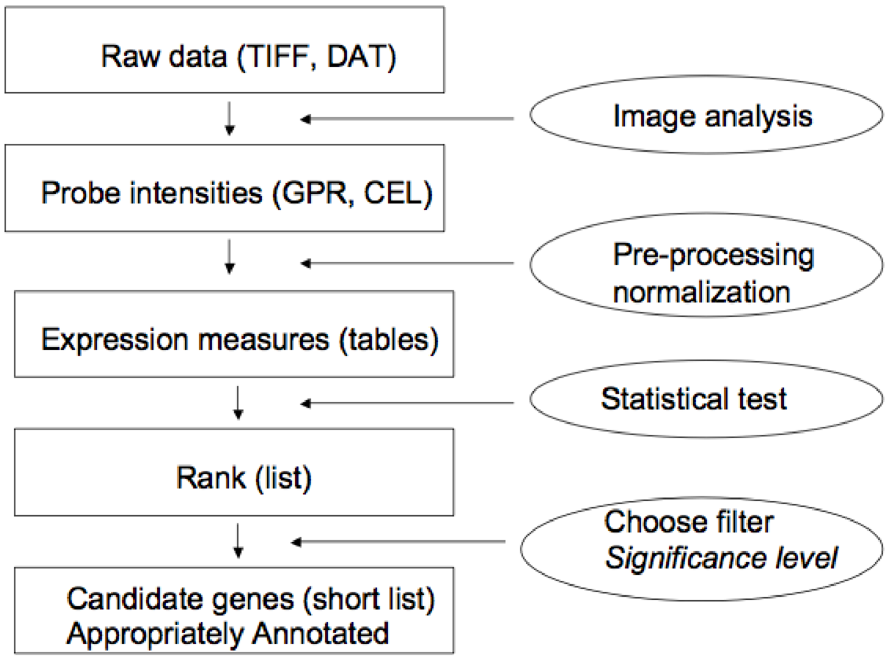
Héctor Corrada Bravo
based on slides developed by
Rafael A. Irizarry and Hao Wu

PASI, Guanajuato, México
May 3-4, 2010

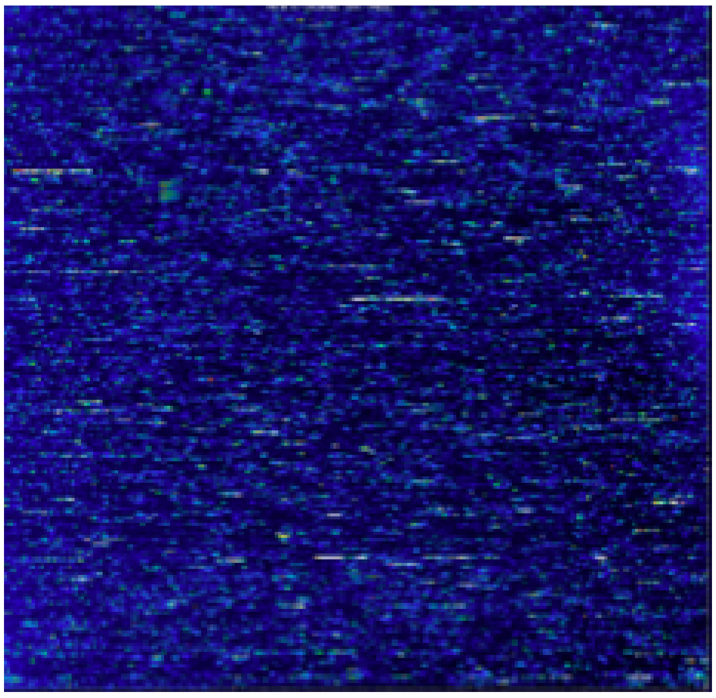
Running Example

Finding differentially expressed genes

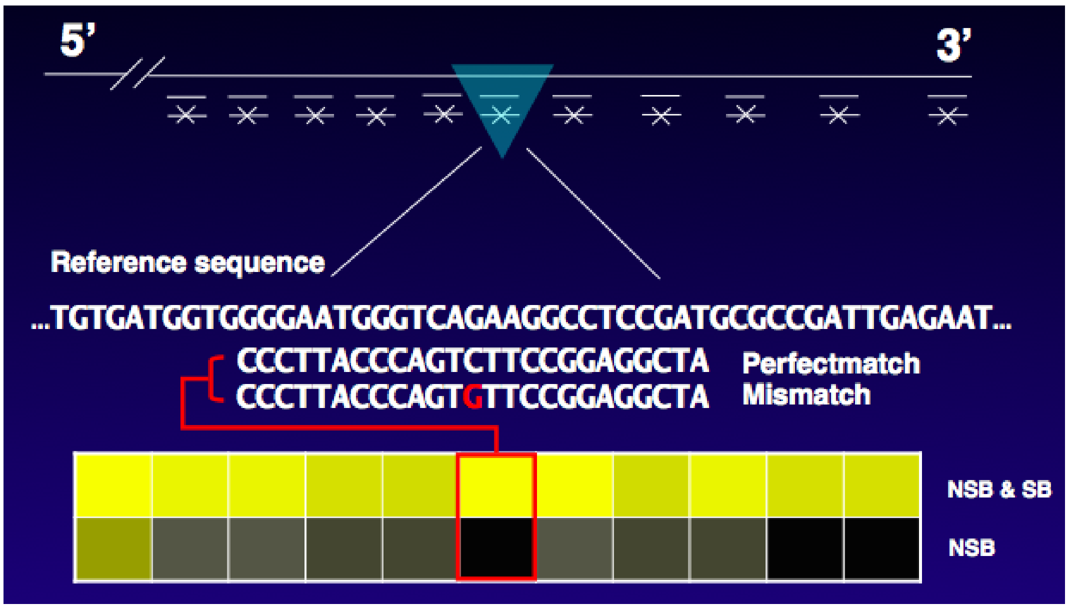
Workflow


Raw data (TIFF, DAT)
Image analysis
Probe intensities (GPR, CEL)
Pre-processing normalization
Expression measures (tables)
Statistical test
Rank (list)
Choose filter *Significance level*
Candidate genes (short list) Appropriately Annotated

Affymetrix GeneChip Arrays



Affymetrix GeneChip Design


5'
3'
Reference sequence
...TGTGATGGTGGGGAATGGGTCAGAAGGCCTCCGATGCGCCGATTGAGAAT...
CCCTTACCCAGTCTTCCGGAGGCTA Perfectmatch
CCCTTACCCAGTGTTCCGGAGGCTA Mismatch
NSB & SB
NSB

Terminology

- Each gene or portion of a gene is represented by 11 to 20 oligonucleotides of 25 base-pairs.
- Probe: an oligonucleotide of 25 base-pairs, i.e. a 25-mer
- Perfect Match (PM): A 25-mer complementary to a reference sequence of interest (e.g., part of a gene)
- Mismatch (MM): same as PM, but with a single homomeric base change for the middle (13th) base (transversion purine $\leftrightarrow$ pyrimidine, G $\leftrightarrow$ C, A $\leftrightarrow$ T.
- Probe-pair: A (PM,MM) pair
- Probe-pair set: a collection of probe-pairs (11 to 20) related to a common gene or fraction of a gene
- Affy ID: an identifier for a probe-pair set
- The purpose of the MM probe design is to measure non-specific binding and background noise

Affymetrix Files

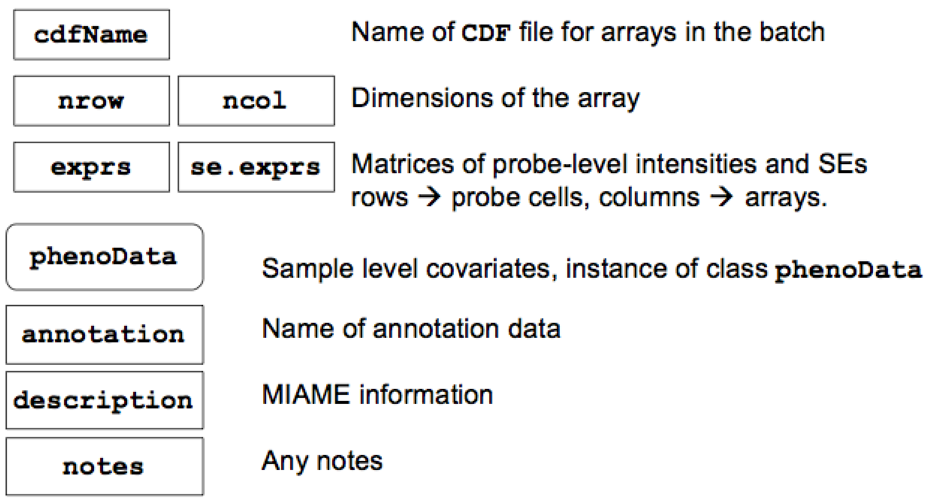
- Main software from Affymetrix company, Micro Array Suite - MAS, now version 5.
- `DAT` file: Image file, $\approx 10^7$ pixels, ≈ 50 MB.
- `CEL` file: Cell intensity file, probe level PM and MM values
- `CDF` file: Chip description file. Describes which probes go with which probe-pair sets (genes, gene fragments, ESTs).

`affy`: Pre-processing Affymetrix Data

- Class definitions for probe-level data: AffyBatch, ProbeSet, Cdf, Cel.
- Basic methods for manipulating microarray objects: printing, plotting, subsetting.
- Functions and widgets for input from `CEL` and `CDF` files, and automatic generation of microarray data objects.
- Diagnostic plots: 2D spatial images, density plots, boxplots, MA-plots, etc.

affy Classes: AffyBatch

Probe-level intensity data for a batch of arrays (same as CDF)



cdfName	Name of CDF file for arrays in the batch
nrow, ncol	Dimensions of the array
exprs, se.exprs	Matrices of probe-level intensities and SEs rows → probe cells, columns → arrays.
phenoData	Sample level covariates, instance of class phenoData
annotation	Name of annotation data
description	MIAME information
notes	Any notes

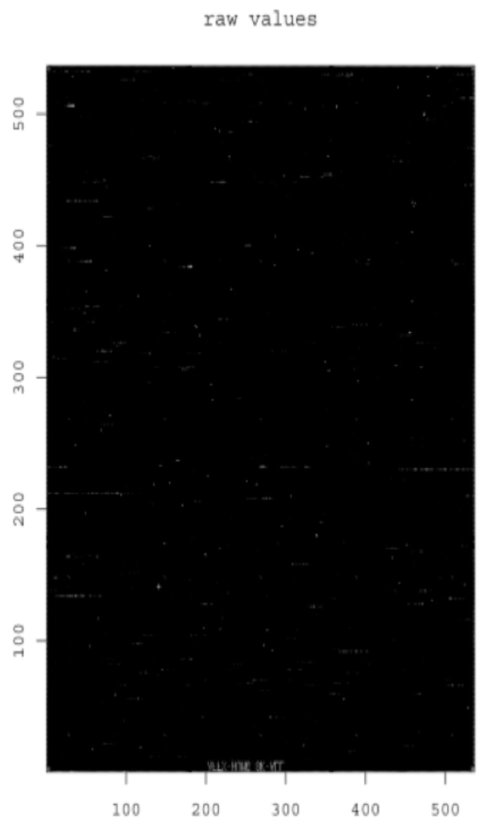
CDF Data Packages

- Data packages containing necessary CDF information are available at www.bioconductor.org
- Packages contain environment objects, which provide mappings between AffyIDs and matrices of probe locations, rows → probe-pairs, columns → PM, MM (e.g., 20×2 matrix for hu6800
- cdfName slot of AffyBatch
- HGU95Av2 and HGU133A provided by packages hgu95av2cdf and hgu133acdf respectively

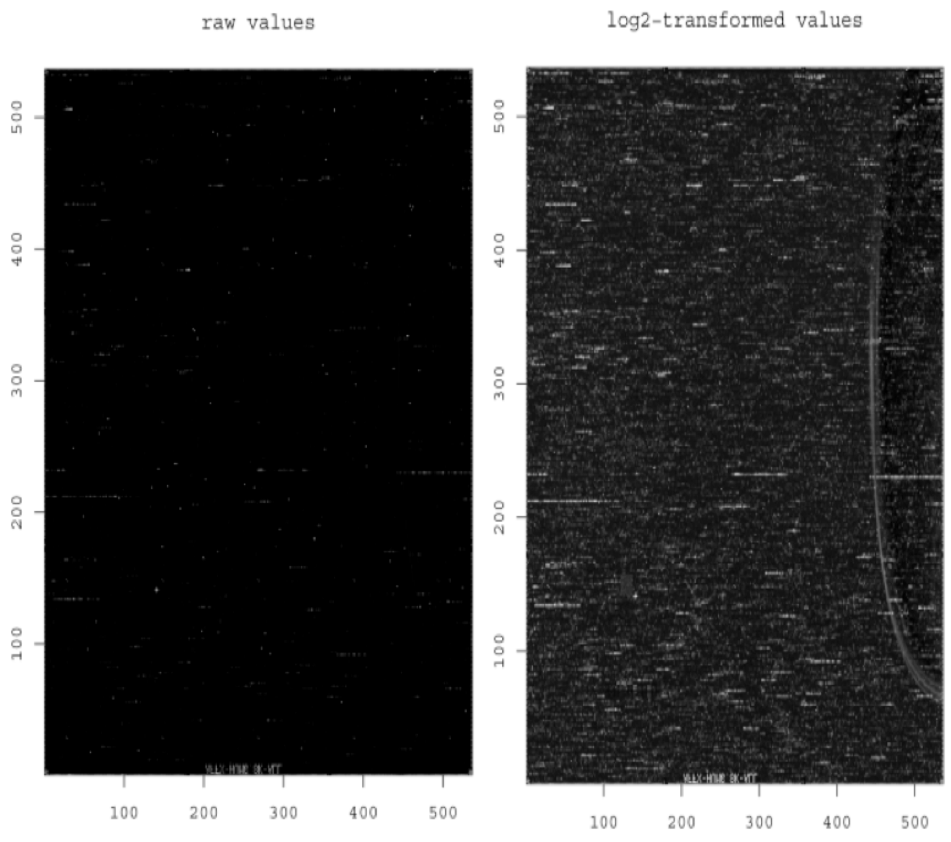
Why Keep Probe-Level Data?

- Quality Control
  - Spatial Effects
  - RNA degradation
- Detection of defective probes
- Transcript sequence “estimates” change
- Ways to reduce to expression measure keep improving

QC



QC



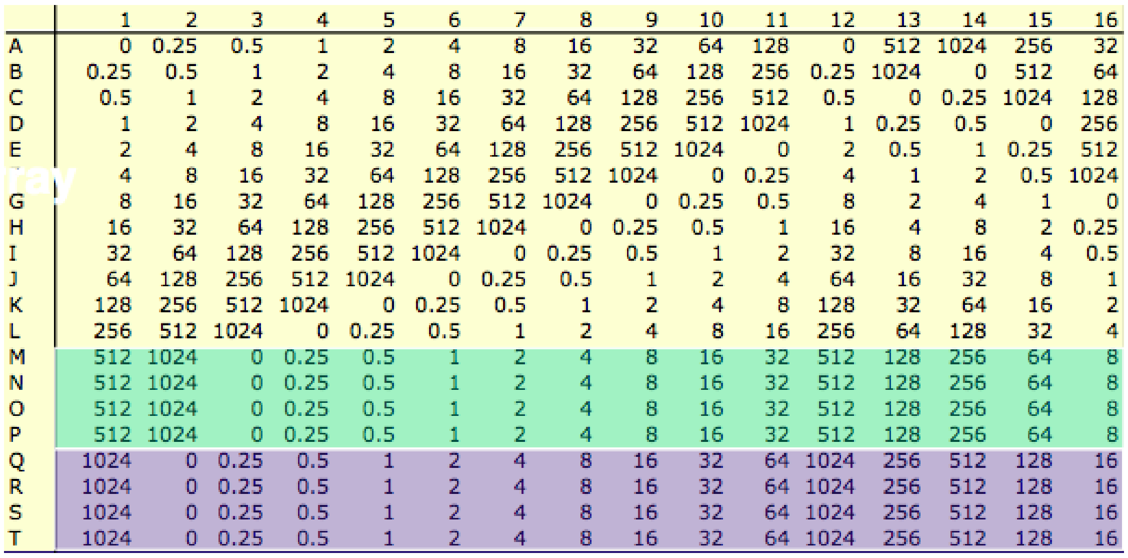
Expression Measures

- 10-20K genes represented by 11-20 pairs of probe intensities (PM & MM)
- Obtain expression measure for each gene on each array by summarizing these pairs
- Background adjustment and normalization are important issues
- There are many methods

Spike-in Experiment

- Throughout this presentation we will be using Data from Affymetrix's spike-in experiment
- Replicate RNA was hybridized to various arrays
- Some probesets were spiked-in at different concentrations across the different arrays
- This gives us a way to assess precision and accuray of expression measurements

Spike-in Experiment



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	0	0.25	0.5	1	2	4	8	16	32	64	128	0	512	1024	256	32
B	0.25	0.5	1	2	4	8	16	32	64	128	256	0.25	1024	0	512	64
C	0.5	1	2	4	8	16	32	64	128	256	512	0.5	0	0.25	1024	128
D	1	2	4	8	16	32	64	128	256	512	1024	1	0.25	0.5	0	256
E	2	4	8	16	32	64	128	256	512	1024	0	2	0.5	1	0.25	512
	4	8	16	32	64	128	256	512	1024	0	0.25	4	1	2	0.5	1024
G	8	16	32	64	128	256	512	1024	0	0.25	0.5	8	2	4	1	0
H	16	32	64	128	256	512	1024	0	0.25	0.5	1	16	4	8	2	0.25
I	32	64	128	256	512	1024	0	0.25	0.5	1	2	32	8	16	4	0.5
J	64	128	256	512	1024	0	0.25	0.5	1	2	4	64	16	32	8	1
K	128	256	512	1024	0	0.25	0.5	1	2	4	8	128	32	64	16	2
L	256	512	1024	0	0.25	0.5	1	2	4	8	16	256	64	128	32	4
M	512	1024	0	0.25	0.5	1	2	4	8	16	32	512	128	256	64	8
N	512	1024	0	0.25	0.5	1	2	4	8	16	32	512	128	256	64	8
O	512	1024	0	0.25	0.5	1	2	4	8	16	32	512	128	256	64	8
P	512	1024	0	0.25	0.5	1	2	4	8	16	32	512	128	256	64	8
Q	1024	0	0.25	0.5	1	2	4	8	16	32	64	1024	256	512	128	16
R	1024	0	0.25	0.5	1	2	4	8	16	32	64	1024	256	512	128	16
S	1024	0	0.25	0.5	1	2	4	8	16	32	64	1024	256	512	128	16
T	1024	0	0.25	0.5	1	2	4	8	16	32	64	1024	256	512	128	16

Setup

- Begin by loading the affy package

```
> library(affy)
```

- The spike-in data is available as a package as well:

```
> library(SpikeInSubset)
> data(spikein95)
```

- If it is not already installed, you might need to call

```
> biocLite("SpikeInSubset")
```

Setup

- Let's create some covariates for the spike-in data indicating which arrays are controls and which are treated

```
> pd <- data.frame(population = c(1,
+     1, 1, 2, 2, 2), replicate = c(1,
+     2, 3, 1, 2, 3))
> rownames(pd) <- sampleNames(spikein95)
> vl <- data.frame(labelDescription = c("1 is control, 2 is treat
+     "arbitrary numbering"))
> phenoData(spikein95) <- new("AnnotatedDataFrame",
+     data = pd, varMetadata = vl)
```

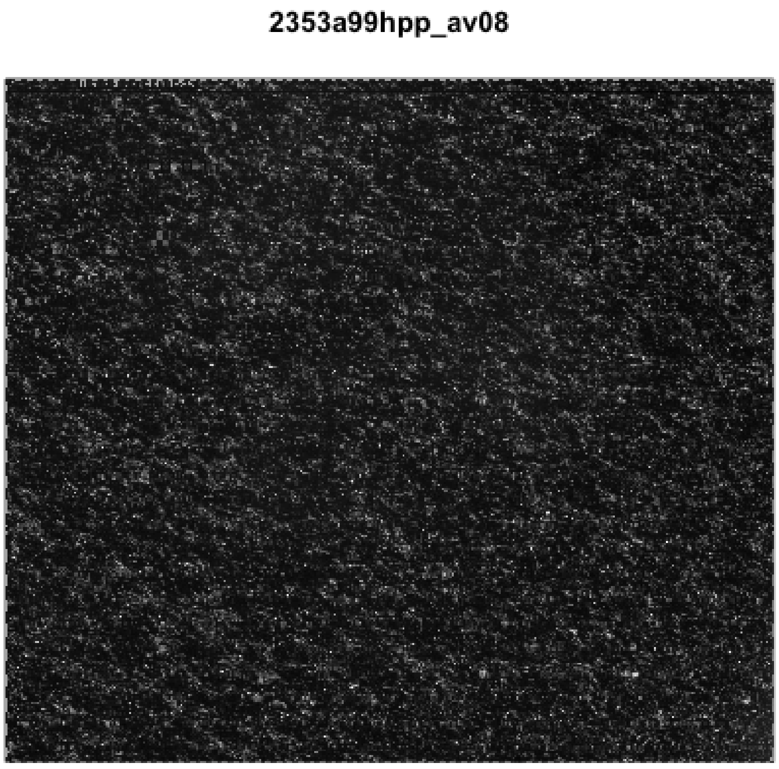
- Let's see how big the expression matrix is

```
> dim(exprs(spikein95))
[1] 409600      6
```

Visualization

You can visualize the expression matrix using the image function.

```
> image(spikein95[, 3])
```



Exploration

- You can get probeset ids (AffyIds). We will use these later to annotate interesting probesets (genes)

```
> ids <- geneNames(spikein95)
> ids[1:10]
 [1] "100_g_at"  "1000_at"   "1001_at"
 [4] "1002_f_at" "1003_s_at" "1004_at"
 [7] "1005_at"   "1006_at"   "1007_s_at"
[10] "1008_f_at"
```

- Find the average number of probes per probeset (gene)

```
> (nrow(exprs(spikein95))/2)/length(ids)
[1] 16.22050
```

- List covariate information for each sample

```
> pData(spikein95)
                 population replicate
1521a99hpp_av06           1         1
1532a99hpp_av04           1         2
2353a99hpp_av08           1         3
1521b99hpp_av06           2         1
1532b99hpp_av04           2         2
2353b99hpp_av08r          2         3
```

Preprocessing: MAS 5.0

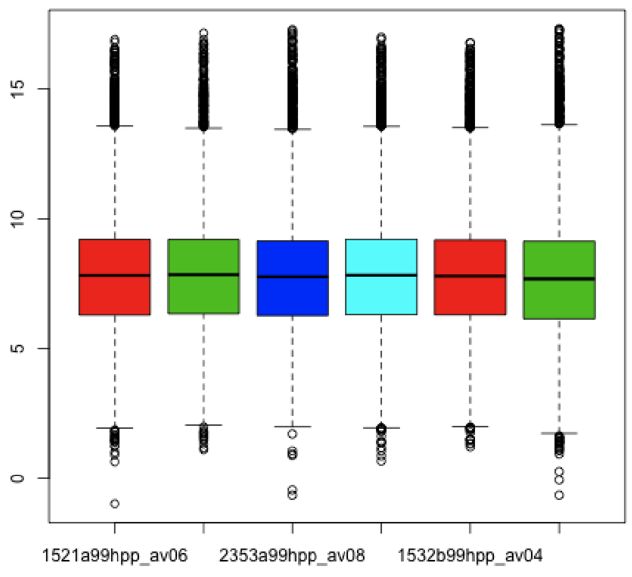
The affy package includes an implementation of the MAS 5.0 method

```
> mas5.eset <- mas5(spikein95)
> mas5.e <- log2(exprs(mas5.eset))
```

Preprocessing: MAS 5.0

We can summarize expression measurements by sample

```
> boxplot(mas5.e, col = 2:5)
```



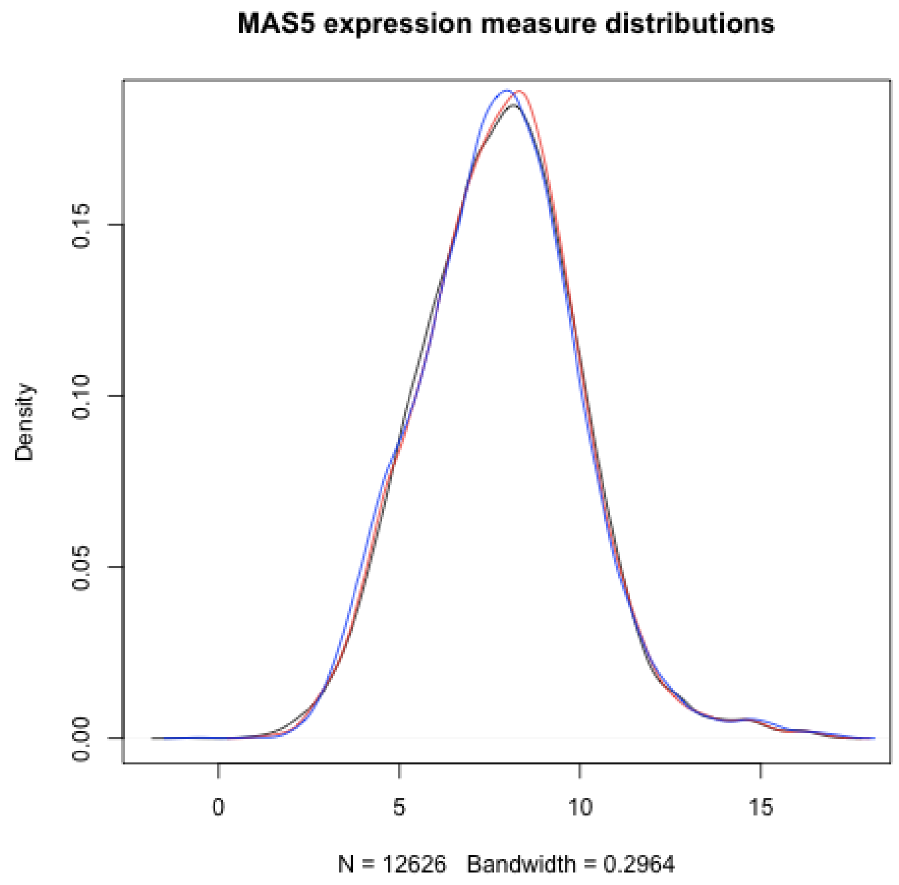
How many points in each boxplot?

Visualization

We can look at distributions of expression measurements for 3 samples

```
> density1 <- density(mas5.e[, 1])
> plot(density1, main = "MAS5 expression measure distributions")
> density2 <- density(mas5.e[, 2])
> lines(density2, col = "red")
> density3 <- density(mas5.e[, 3])
> lines(density3, col = "blue")
```

Visualization



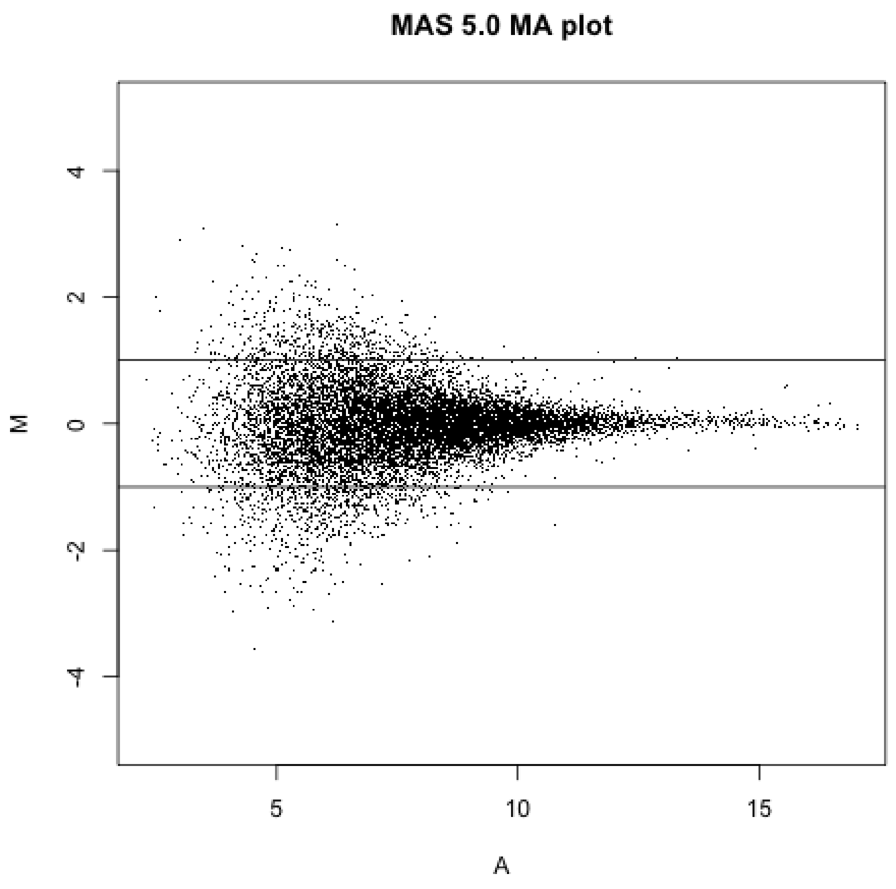
Exploration: The MA plot

- We are interested in genes with overall large fold-changes (spike-ins)
- Why not look at average log ratios?
- We can make MA plots:
  - M: difference in average log intensities
  - A: average log intensities

Exploration: The MA plot

```
> Index1 <- which(mas5.eset$population ==
+     1)
> Index2 <- which(mas5.eset$population ==
+     2)
> d <- rowMeans(mas5.e[, Index2]) -
+     rowMeans(e[, Index1])
> a <- rowMeans(mas5.e)
> plot(a, d, ylim = c(-5, 5), main = "MAS 5.0 MA plot",
+     xlab = "A", ylab = "M", pch = ".")
> abline(h = c(-1, 1))
```

Exploration: The MA plot



Exploration: The MA plot

- Let's look where the spiked-in probesets are in this plot
- Let's reload the spike-in AffyBatch object to get the original pData slot with spike-in information

```
> data(spikein95)
> pData(spikein95)[, 1:2]
                  37777_at 684_at
1521a99hpp_av06       0.00   0.25
1532a99hpp_av04       0.00   0.25
2353a99hpp_av08       0.00   0.25
1521b99hpp_av06       0.25   0.50
1532b99hpp_av04       0.25   0.50
2353b99hpp_av08r      0.25   0.50
```

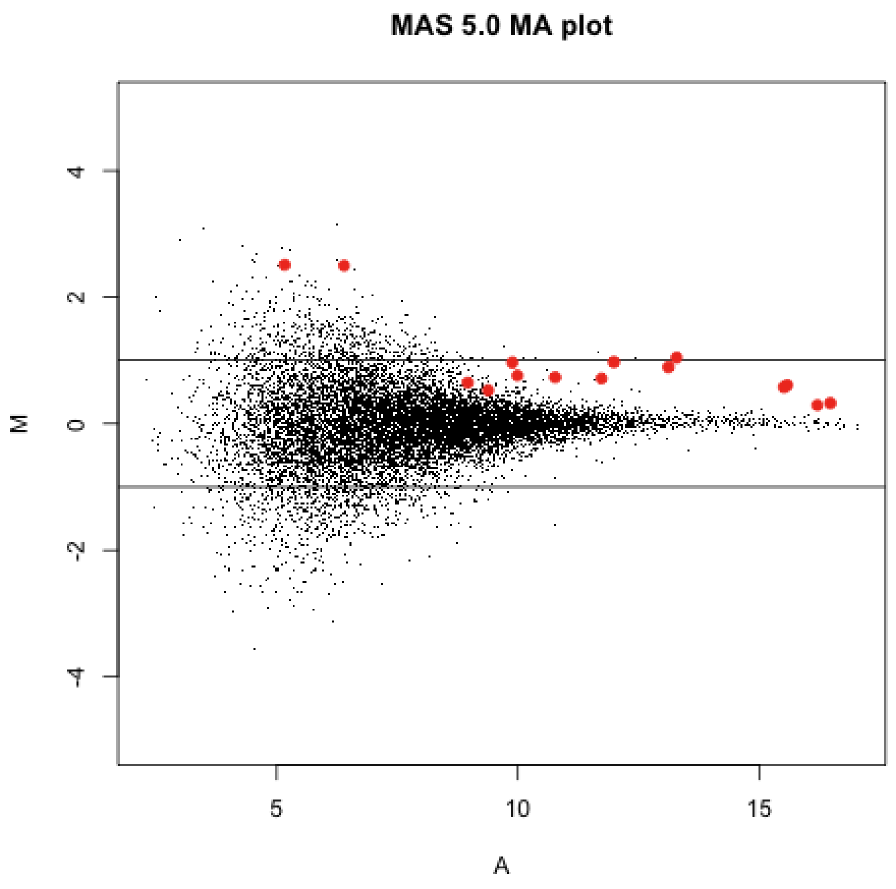
- Find the indices of the spiked-in probesets

```
> spikedin <- colnames(pData(spikein95))
> spikedIndex <- match(spikedin,
+      featureNames(mas5.eset))
```

- And add them to the plot

```
> points(a[spikedIndex], d[spikedIndex],
+      pch = 19, col = "red")
```

Exploration: The MA plot



Exploration: The MA plot

Let's see how the spike-in average log-ratios of expression rank among all the probes.

```
> mas5.ranks <- sort(rank(-abs(d))[spikedIndex])

> mas5.ranks
 1708_at 37777_at   407_at   1024_at 36311_at 36889_at 36202_at
       1        31       35      1134     1320     1340     1554
38734_at 39058_at   546_at    684_at 33818_at 36085_at  1597_at
    2055     2196     2273      2616     2890     3098     3481
1091_at 40322_at
   5433     5888
```

Preprocessing: RMA

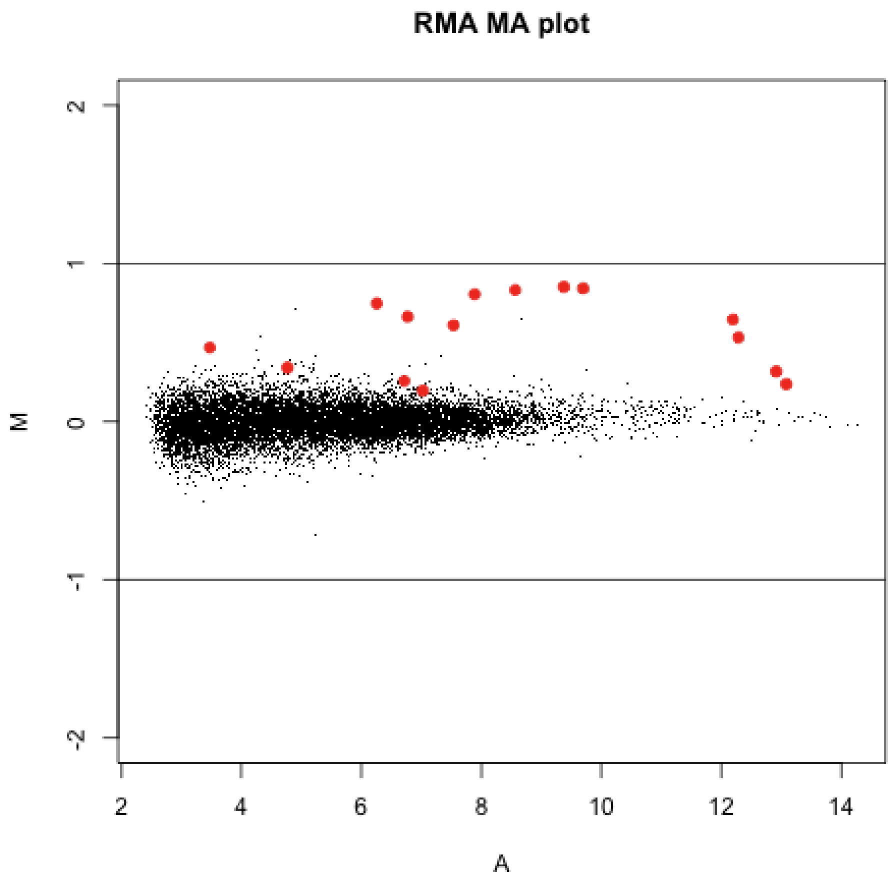
- Can we improve this? Let's try RMA

```
> rma.eset <- rma(spikein95)
> rma.e <- exprs(rma.eset)
```

- And make an MA plot

```
> d <- rowMeans(rma.e[, Index2] -
+      rma.e[, Index1])
> a <- rowMeans(rma.e)
> plot(a, d, ylim = c(-2, 2), main = "RMA MA plot",
+      xlab = "A", ylab = "M", pch = ".")
> abline(h = c(-1, 1))
> points(a[spikedIndex], d[spikedIndex],
+      pch = 19, col = "red")
```

Preprocessing: RMA



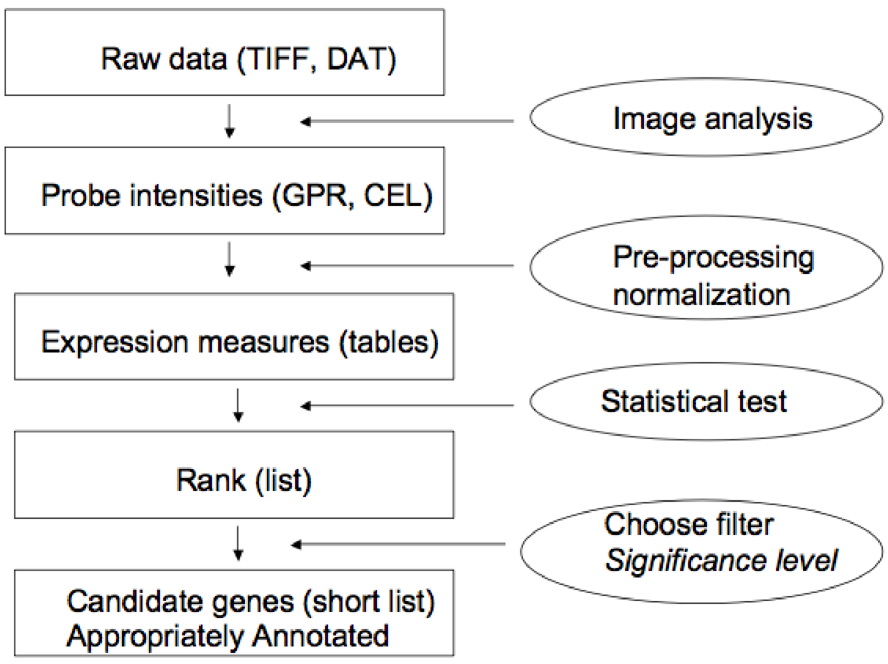
Proprocessing: RMA

How do the spike-ins rank now?

```
> rma.ranks <- sort(rank(-abs(d))[spikedIndex])
```

```
 1708_at 36202_at  1024_at 36311_at   546_at 38734_at 36889_at
       1        2        3        4        5        6        9
36085_at 39058_at 33818_at   407_at 37777_at 40322_at   684_at
      11       12       14       16       53       86      226
 1091_at  1597_at
     330      689
```

Workflow


Raw data (TIFF, DAT)
Image analysis
Probe intensities (GPR, CEL)
Pre-processing normalization
Expression measures (tables)
Statistical test
Rank (list)
Choose filter
Significance level
Candidate genes (short list) Appropriately Annotated

Back to basics

- Observations: $X_1, X_2, \ldots, X_M$ and $Y_1, Y_2, \ldots, Y_N$
- Averages:

$$\bar{X} = \frac{1}{M}\sum_{i=1}^{M} X_i \qquad \bar{Y} = \frac{1}{N}\sum_{i=1}^{N} Y_i$$

- Variances:

$$s_X^2 = \frac{1}{M-1}\sum_{i=1}^{M}(X_i - \bar{X})^2$$

$$s_Y^2 = \frac{1}{N-1}\sum_{i=1}^{N}(Y_i - \bar{Y})^2$$

Back to basics

The t-statistic:

$$\frac{\bar{Y} - \bar{X}}{\sqrt{\frac{s_Y^2}{N} + \frac{s_X^2}{M}}}$$

Back to basics

- If N and M are large, then the t-statistic is normally distributed with mean 0 and SD of 1
- If the observed data is normally distributed then the t-statistic follows a t-distribution, regardless of N and M
- Regardless, the square of the t-test is proportional to the ratio of across group variance to within group variance:
- t-statistic squared (if $M = N$):

$$N \times \frac{\left(\bar{Y} - \bar{X}\right)^2}{s_Y^2 + s_X^2}$$

Another useful plot

- We can use the t-statistic to check for significant difference in mean (average log ratio of expression, fold-change)
- This takes variation into account
- The volcano plot shows, for a particular test, negative log p-value against the effect size (M)
- How do we get p-values?

The volcano plot

- Package genefilter is very efficient at computing *t*-statistics and *p*-values for all probesets (rows) in a matrix

```
> library("genefilter")
```

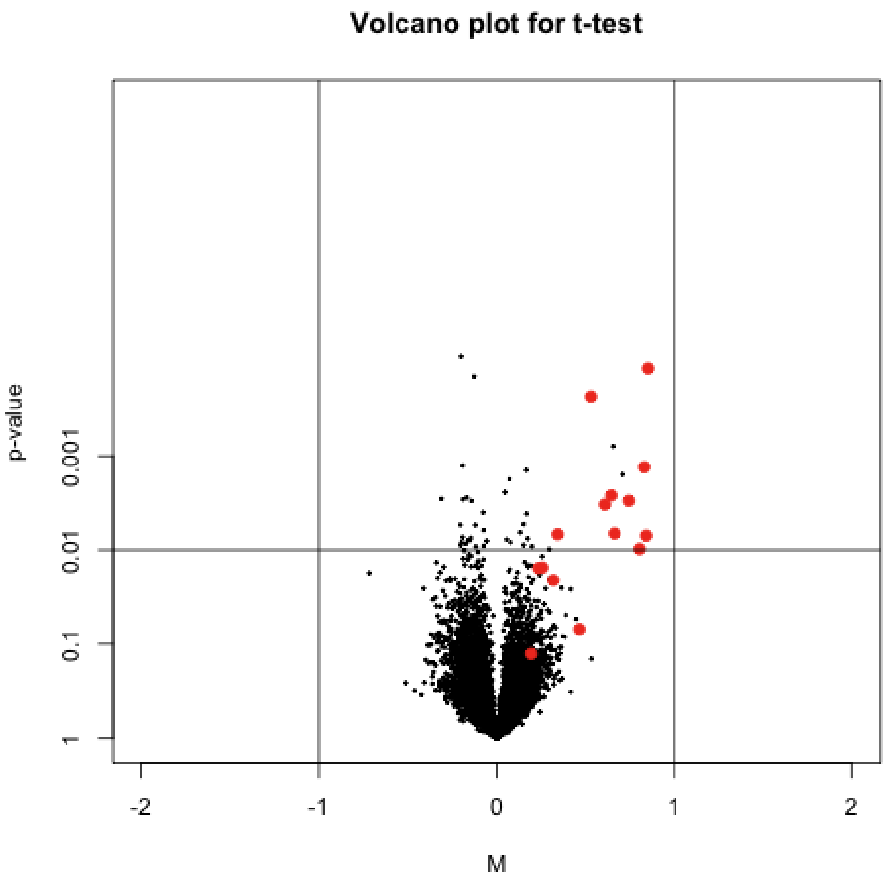
- A little nuiscance: we have to get our covariate data into the RMA ExpressionSet object

```
> pData(rma.eset) <- pData(mas5.eset)
> tt <- rowttests(rma.e, factor(rma.eset$population))
> lod <- -log10(tt$p.value)
```

- Now, make the volcano plot

```
> plot(d, lod, cex = 0.25, main = "Volcano plot for t-test",
+      xlim = c(-2, 2), xlab = "M",
+      ylab = "p-value", yaxt = "n")
> axis(2, at = seq(0, 3, by = 1),
+      labels = 10^(-seq(0, 3, by = 1)))
> points(d[spikedIndex], lod[spikedIndex],
+      pch = 19, col = "red")
> abline(h = 2, v = c(-1, 1))
```

The volcano plot



The *t*-test

Let's see how the spike-ins rank according to the *t*-statistic

```
> ttest.ranks <- rank(-abs(tt$statistic))[spikedIndex]
> names(ttest.ranks) <- colnames(pData(spikein95))
> ttest.ranks <- sort(ttest.ranks)
```

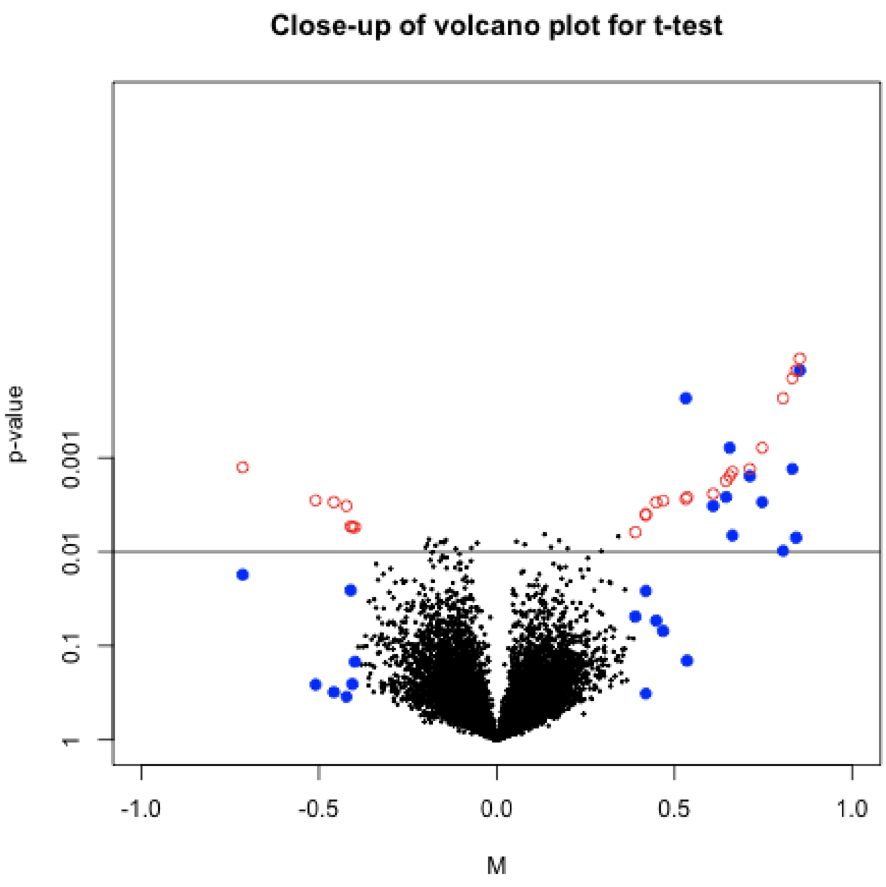
```
 1708_at 36202_at 33818_at 36311_at 36085_at 38734_at 39058_at
       1        3        5        8       13       17       19
36889_at 37777_at  1024_at   546_at   684_at  1091_at 40322_at
      27       28       29       45       70       72       92
  407_at  1597_at
     391      900
```

The volcano plot

Let's do another volcano plot and label points according to their effect-size and *t*-statistic rank

```
> o1 <- order(abs(d), decreasing = TRUE)[1:25]
> o2 <- order(abs(tt$statistic),
+     decreasing = TRUE)[1:25]
> o <- union(o1, o2)
> plot(d[-o], lod[-o], cex = 0.25,
+     xlim = c(-1, 1), ylim = range(lod),
+     main = "Close-up of volcano plot for t-test",
+     xlab = "M", ylab = "p-value",
+     yaxt = "n")
> axis(2, at = seq(0, 3, by = 1),
+     labels = 10^(-seq(0, 3, by = 1)))
> abline(h = 2)
> points(d[o1], lod[o1], pch = 19,
+     col = "blue")
> points(d[o1], lod[o2], pch = 1,
+     col = "red")
```

The volcano plot



Estimating the variance

- If different genes (or probes) have different variation, then it is not a good idea to use average log ratios even if we do care about significance
- Under a random model, we need to estimate SD
- The *t*-test divides by SD
- But, with few replicates, estimates of SD are not stable
- This explains why the *t*-test is not powerful
- There are many proposals for estimating variation
- Many borrow strength across genes
- Empirical Bayes approaches are popular
- SAM, an ad-hoc procedure, is even more popular

Some examples of tests

- Notation:
  - T is average log expression in treatment
  - C is average log expression in control
  - S is SD
- Tests:
  - Average log fold-change: $(T - C)$
  - t-statistic: $(T - C)/S$
  - SAM shrunken statistic: $(T - C)/(S + S_0)$
  - Bayesian posteriors: $(T - C)/\sqrt{(S^2 + K^2)}$
  - Wilcoxon: rank test
  - Ad-hoc pairwise comparison: no formula
- Many of these are in the limma package, SAM is in the siggenes package. Also look at the EBayes package.

Limma

- Let's use the moderated t-test from limma

```
> library(limma)
> design <- model.matrix(~factor(rma.eset$population))
> fit <- lmFit(rma.eset, design)
> ebayes <- eBayes(fit)
```

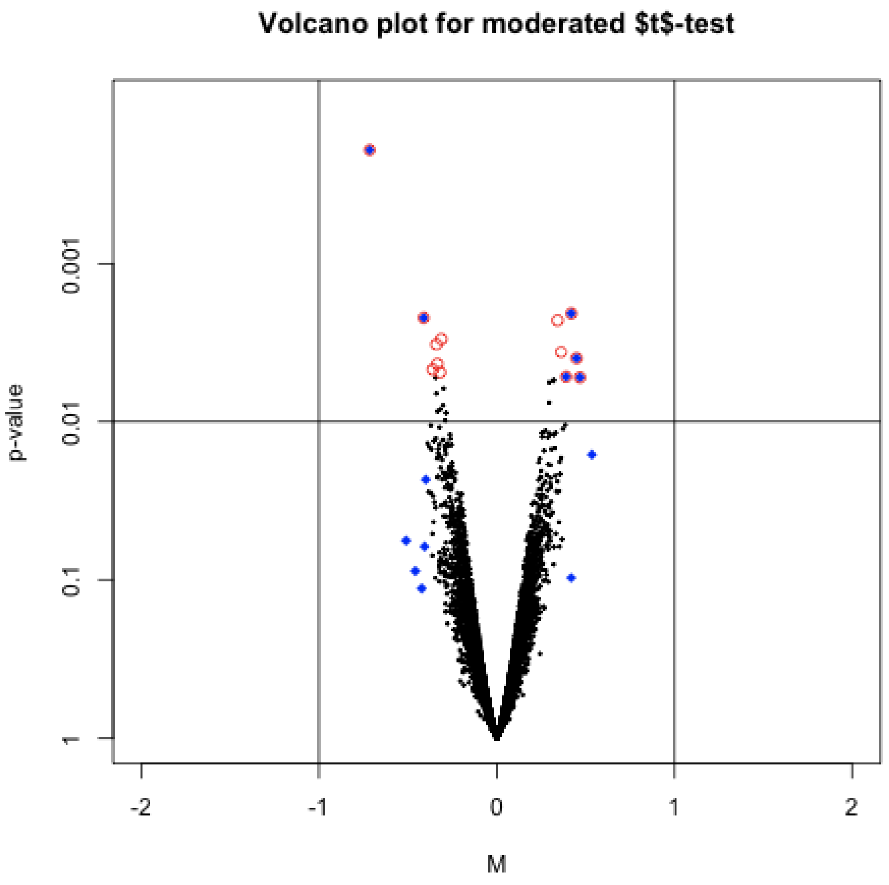
- Do it's volcano plot

```
> lod <- -log10(ebayes$p.value[,
+     2])
> mtstat <- ebayes$t[, 2]
> o1 <- order(abs(d), decreasing = TRUE)[1:25]
> o2 <- order(abs(mtstat), decreasing = TRUE)[1:25]
> o <- union(o1, o2)
> plot(d[-o], lod[-o], cex = 0.25,
+     xlim = c(-2, 2), ylim = c(0,
+         4), main = "Volcano plot for moderated $t$-test",
+     xlab = "M", ylab = "p-value",
+     yaxt = "n")
> axis(2, at = seq(0, 3, by = 1),
```

Limma

```
+     labels = 10^(-seq(0, 3, by = 1)))
> abline(h = 2, v = c(-1, 1))
> points(d[o1], lod[o1], pch = 18,
+     col = "blue")
> points(d[o2], lod[o2], pch = 1,
+     col = "red")
```

Limma



Limma

Let's see how the spike-ins rank now

> mt.rank <- sort(rank(-abs(mtstat))[spikedIndex])

```
 1708_at 36202_at 36311_at 38734_at  1024_at 36085_at 33818_at
       1        2        3        6        7        8        9
39058_at   546_at 36889_at 37777_at   407_at 40322_at   684_at
      10       11       12       16       25       28       48
 1091_at  1597_at
      77      465
```

For next time...

- We will look at a breast cancer dataset, dowload from here: `http://www.biostat.jhsph.edu/~hcorrada/PASI_2010/chang03.rda`
- We will need a few more packages, you can install with biocLite now if you want to save time: hgu95av2.db, XML, annotate, KEGG.db, GO.db, annaffy
- We will also do a little bit of analysis of second-generation sequencing data, download a dataset from here: `http://www.biostat.jhsph.edu/~hcorrada/PASI_2010/seqdata.zip`
- We will also need a few more packages for sequence analysis: ShortRead, BSgenome.Scerevisiae.UCSC.sacCer1, yeast2probe

The R Workspace

- Like most programming environments the startup of R can be controlled by your environment (e.g. environment variables, or startup files in your home directory).
- The most important environment variable is *R_LIBS*. This environment variable dictates where packages are installed, so if you are on a shared system, or a system where you do not have admin rights then you want to use this variable to control where packages are installed.
- This variable should be set in your .bashrc file (or in your environment variables widget in windows).
- You can check where R will check for packages by using the .libPaths function. This function additionally allows you to add directories to the search path.
- .Platform, .Machine
- Additionally, you have a file called .Rprofile which can be used to set up some initial code.

```
> dirname(.libPaths())
```

The R Workspace

```
[1] "/Library/Frameworks/R.framework/Resources"

> basename(.libPaths())

[1] "library"
```

Examining the R session

- Often we want to know what packages / capabilities / options R is using. There are a number of relevant functions for examining the R session.

```
> sessionInfo()
```

```
R version 2.11.0 (2010-04-22)
x86_64-apple-darwin9.8.0

locale:
[1] en_US.utf-8/en_US.utf-8/C/C/en_US.utf-8/en_US.utf-8

attached base packages:
[1] stats     graphics  grDevices
[4] utils     datasets  methods
[7] base

other attached packages:
[1] hgu95acdf_2.6.0
```

Examining the R session

```
[2] SpikeInSubset_1.2.8
[3] affy_1.26.0
[4] Biobase_2.8.0
[5] RColorBrewer_1.0-2

loaded via a namespace (and not attached):
[1] affyio_1.16.0
[2] preprocessCore_1.10.0
[3] tools_2.11.0
```

```
> capabilities()
```

```
    jpeg      png     tiff    tcltk
    TRUE     TRUE     TRUE     TRUE
     X11     aqua http/ftp  sockets
   FALSE     TRUE     TRUE     TRUE
  libxml     fifo   cledit    iconv
    TRUE     TRUE    FALSE     TRUE
     NLS  profmem    cairo
    TRUE     TRUE     TRUE
```

Examining the R session

```
> options()[c("pkgType", "device")]

$pkgType
[1] "mac.binary.leopard"

$device
function (file = ifelse(onefile, "Rplots.pdf", "Rplot%03d.pdf"),
    width, height, onefile, family, title, fonts, version, paper,
    encoding, bg, fg, pointsize, pagecentre, colormodel, useDingbats,
    useKerning, fillOddEven, maxRasters)
{
    initPSandPDFfonts()
    new <- list()
    if (!missing(width))
        new$width <- width
    if (!missing(height))
        new$height <- height
    if (!missing(onefile))
        new$onefile <- onefile
```

Examining the R session

```
    if (!missing(title))
        new$title <- title
    if (!missing(fonts))
        new$fonts <- fonts
    if (!missing(version))
        new$version <- version
    if (!missing(paper))
        new$paper <- paper
    if (!missing(encoding))
        new$encoding <- encoding
    if (!missing(bg))
        new$bg <- bg
    if (!missing(fg))
        new$fg <- fg
    if (!missing(pointsize))
        new$pointsize <- pointsize
    if (!missing(pagecentre))
        new$pagecentre <- pagecentre
    if (!missing(colormodel))
```

Examining the R session

```
        new$colormodel <- colormodel
    if (!missing(useDingbats))
        new$useDingbats <- useDingbats
    if (!missing(useKerning))
        new$useKerning <- useKerning
    if (!missing(fillOddEven))
        new$fillOddEven <- fillOddEven
    if (!missing(maxRasters))
        new$maxRasters <- maxRasters
    old <- check.options(new, name.opt = ".PDF.Options", envir = .PSe
    if (!missing(family) && (inherits(family, "Type1Font") ||
        inherits(family, "CIDFont"))) {
        enc <- family$encoding
        if (inherits(family, "Type1Font") && !is.null(enc) &&
            enc != "default" && (is.null(old$encoding) || old$encodin
            "default"))
            old$encoding <- enc
        family <- family$metrics
    }
```

Examining the R session

```
    if (is.null(old$encoding) || old$encoding == "default")
        old$encoding <- guessEncoding()
    if (!missing(family)) {
        if (length(family) == 4L) {
            family <- c(family, "Symbol.afm")
        }
        else if (length(family) == 5L) {
        }
        else if (length(family) == 1L) {
            pf <- pdfFonts(family)[[1L]]
            if (is.null(pf))
                stop(gettextf("unknown family '%s'", family),
                  domain = NA)
            matchFont(pf, old$encoding)
        }
        else stop("invalid 'family' argument")
        old$family <- family
    }
    version <- old$version
```

Examining the R session

```
        versions <- c("1.1", "1.2", "1.3", "1.4", "1.5", "1.6")
        if (version %in% versions)
            version <- as.integer(strsplit(version, "[.]")[[1L]])
        else stop("invalid PDF version")
        onefile <- old$onefile
        if (!checkIntFormat(file))
            stop("invalid 'file'")
        .External(PDF, file, old$paper, old$family, old$encoding,
            old$bg, old$fg, old$width, old$height, old$pointsize,
            onefile, old$pagecentre, old$title, old$fonts, version[1L],
            version[2L], old$colormodel, old$useDingbats, old$useKerning,
            old$fillOddEven, old$maxRasters)
        invisible()
    }
    <environment: namespace:grDevices>

    > R.version
```

Examining the R session

```
                   _
    platform       x86_64-apple-darwin9.8.0
    arch           x86_64
    os             darwin9.8.0
    system         x86_64, darwin9.8.0
    status
    major          2
    minor          11.0
    year           2010
    month          04
    day            22
    svn rev        51801
    language       R
    version.string R version 2.11.0 (2010-04-22)
```