# Analysis of complex differential gene expression experiments with multiple R/Bioconductor tools via a simple unified interface

Ryan C. Thompson

August 3, 2014

# Outline

- Comparing RNA-seq data to array data
- Modeling RNA-seq count data
- Modeling biological variation
- Normalization of RNA-seq counts
- Differential expression testing
- Giving limma a chance
- QuickDGE: Design experiment once, run all 3 methods

# Introduction

# RNA-seq is Like Microarrays

- We are measuring the same thing: RNA expression levels
- The data looks similar: a matrix of genes $\times$ samples
- We want to ask similar questions: Which genes are differentially expressed?

# RNA-seq is **Not** Like Microarrays

- RNA-seq measurements are discrete counts (not continuous intensity values)
- Each gene has exactly one count value (not an average of multiple probes)
- Counts are directly proportional to RNA abundances and lengths (not probe affinities)
  - Scaling normalization is appropriate
  - Quantile normalization is not needed

# RNA-seq is **Really** Not Like Microarrays

- Count measurements are interdependent (genes compete for limited "sequencing space")
  - Normalization must take this into account
  - Otherwise RPKM and counts-per-million are *not* comparable across samples
- Counts are not normally distributed
  - Not even if we pretend they are continuous
  - Not even if we take the logarithm
  - Not even if we cross our eyes and squint really hard
- So we can't shoehorn these counts into methods that assume a normal distribution, such as t-tests, ANOVA, least squares regression, etc.

# Modeling RNA-seq counts

# Where do these counts come from, anyway?

- RNA-seq doesn't produce counts, it produces read sequences!
- We align those reads to the genome (or transcriptome), then we count the number of reads in a sample that align unambiguously to each gene.
- We discard ambiguous reads, so each count is an exact integer
- There are more subtlties to counting reads. I'm not covering them here.

# Flipping coins

- How can we model discrete counts?
- By flipping (really biased) coins
- Suppose we have a coin with a probability **p** of landing on heads
- And we flip that coin **N** times
- Then the number of heads has a distribution **Binomial(N,p)**

# Binomial distribution

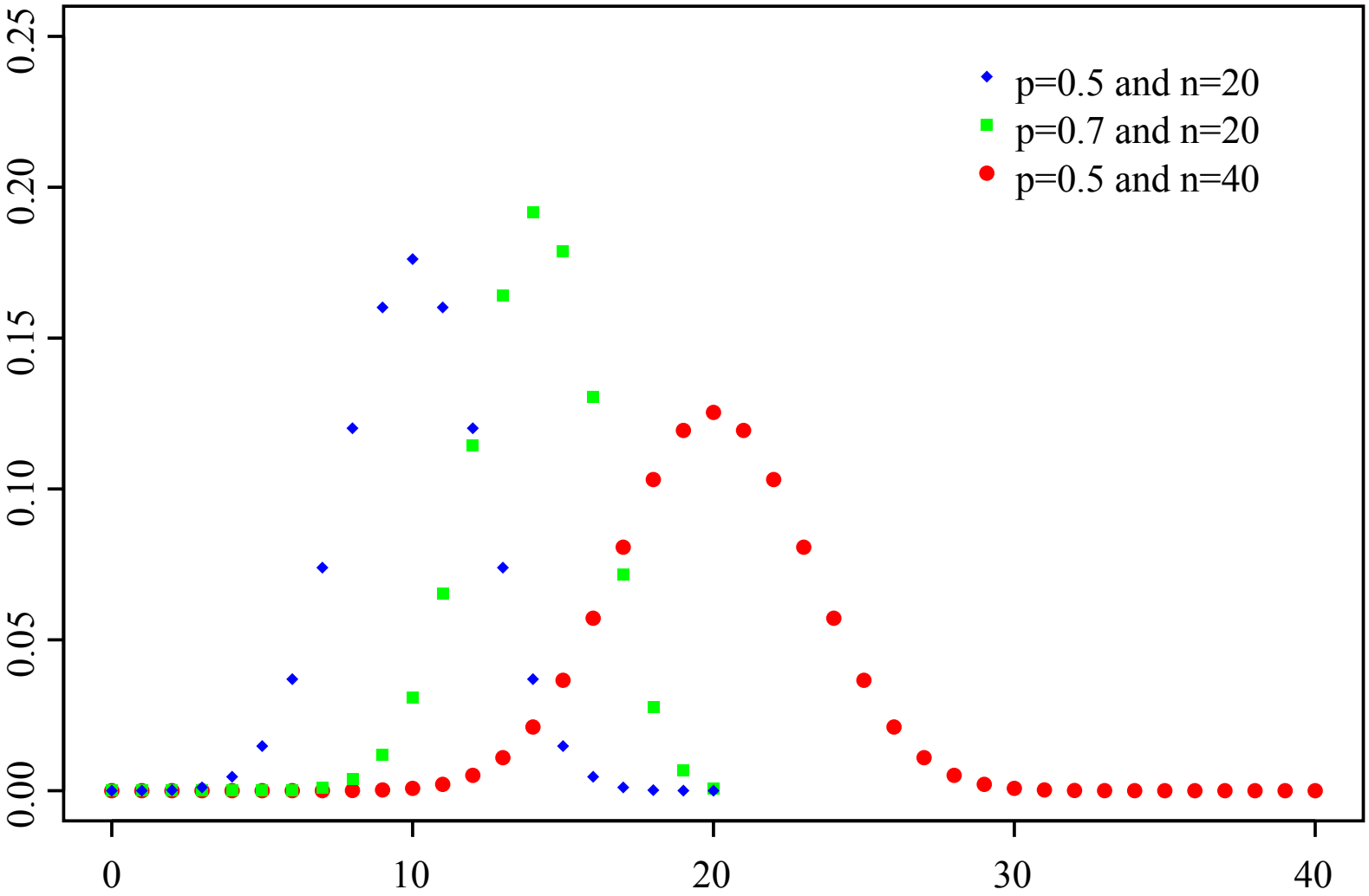


Figure 1: Binomial distribution examples

# Sampling fragments

- Suppose we have a gene that makes up a proportion **p** of the total RNA in the sample
- Then each fragment we sample has a probability **p** of coming from this gene
- If we sample **N** random fragments from the RNA sample, then the distribution of this gene's read count is **Binomial(N,P)**

# But factorials are hard!

- To compute Binomial probabilities, we need to compute **N!**.
  Binomial pmf equation:
  $\binom{n}{k} p^k (1-p)^{n-k}$

  Where
  $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
- Typical values of N are in the 10,000,000 range.
- Computing 10,000,000! is slow
- Maybe there's an easier way?

# Poisson Approximation to the Binomial

- As N becomes large and N*p remains small, we can approximate **Binomial(N,p)** with **Poisson($\lambda$)**, where $\lambda = N * p$
- The approximation is actually very very good
- Calculating Poisson probabilities is much easier (i.e. faster on a computer)
- Poisson distribution has only one parameter $\lambda$, which is equal to its mean *and* its variance
- This means that the variance of the Poisson distribution is constrained to be equal to the mean

# Technical variation in RNA-seq is Poisson-distributed

- Suppose we conduct many sequencing experiments on the same sample, with the same RNA proportions,
- The counts for each gene are sampled from a Poisson distribution with $\lambda = N * p$
- So, for technical replicates, variance = mean!
- But we're just using the Poisson as an approximation, right? Can we really rely on its properties like this?

# Technical replicates: Variance = Mean

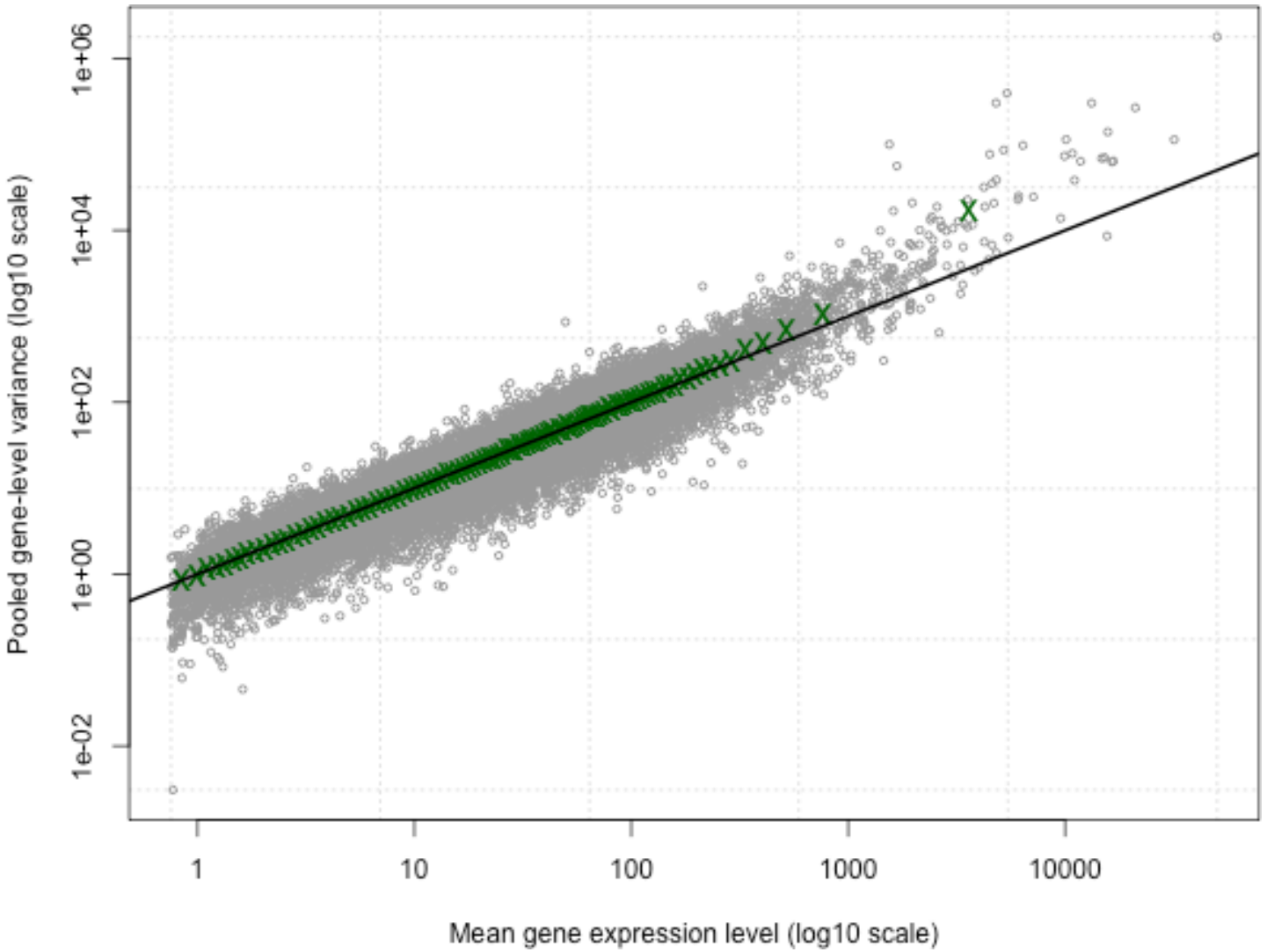


Figure 2: Variance vs Mean in Technical Replicates

# Biological replicates: Variance > Mean

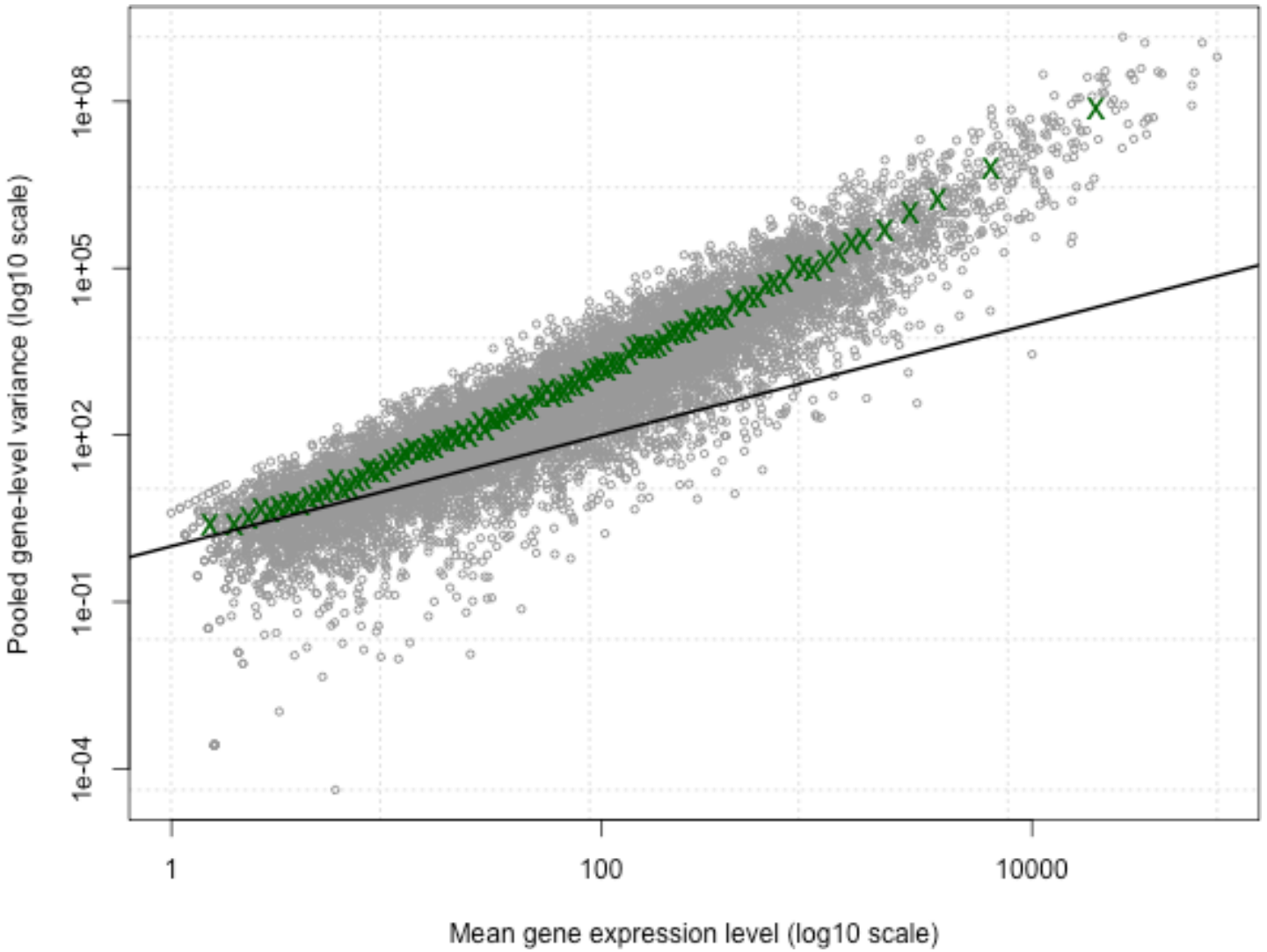


Figure 3: Variance vs Mean in Biological Replicates

# Modeling Biological Variation

# Biological replicates have "excess" variance

- The Poisson approximation holds when all samples have exactly the same RNA proportions
- So the only variation arises from the random process of sampling fragments from the same pool of RNA.
- When we draw many different RNA samples from a population, they will have different RNA proportions
- But we still have all the technical variation as well
- So biological replicates always have Poisson/techincal variance plus some more variance from biological variation
- How can we model this biological variation?

# A Biological-Technical mixture model

- We assume that for each gene, the population has a gamma distribution for that gene's RNA expression level
- So, to generate a sample for a gene, we first sample an abundance $\lambda$ from the population's gamma distribution and then we sample a count from corresponding Poisson distribution.
- In other words, the population's count distribution is a mixture of Poisson distributions, with the gamma distribution determining the mixing proportions.
- This defines a gamma-poisson mixture model, which just happens to be equivalent to the negative binomial distribution.

# The gamma distribution

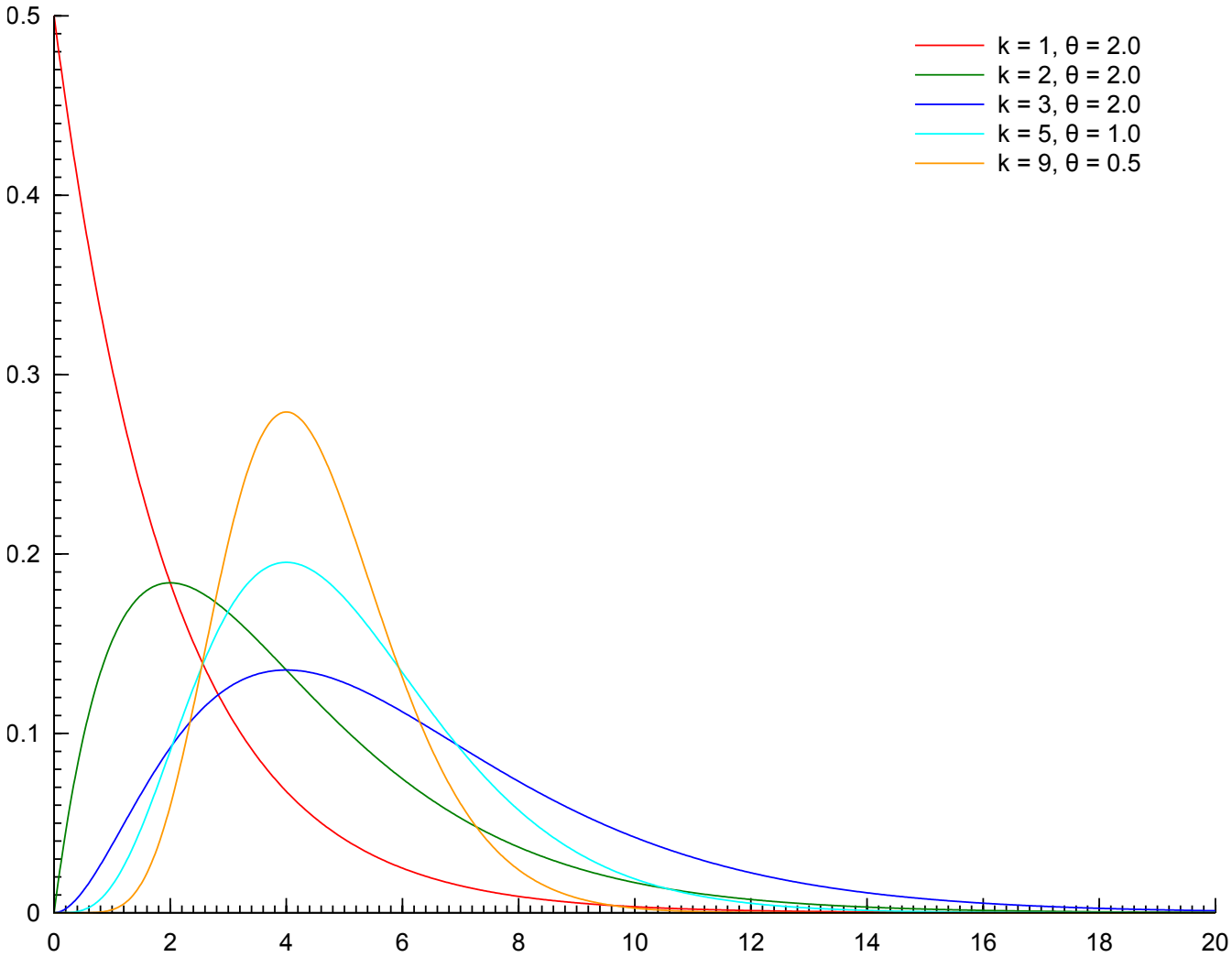


Figure 4: Gamma distribution examples

# Negative Binomial vs Poisson

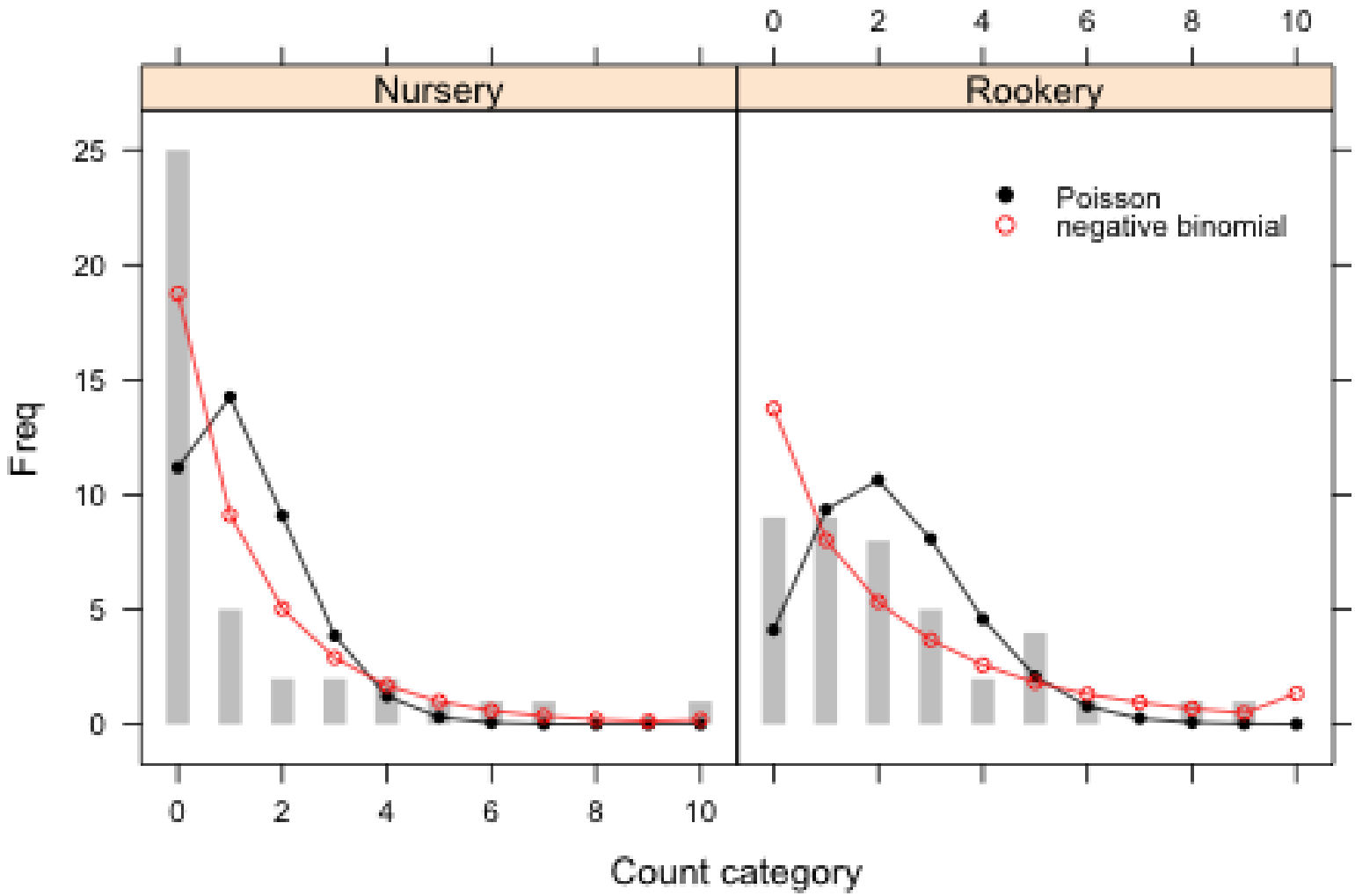


Figure 5: Comparison of NB and Poisson fits to data

# The Biological Coefficient of Variation

For NB distribution:

$$\sigma^2 = \mu + \phi\mu^2$$

Divide both sides by $\mu^2$ and note that $\frac{\sigma}{\mu}$ is the coefficient of variation (CV):

$$\text{CV}^2 = \frac{\sigma^2}{\mu^2} = \frac{1}{\mu} + \phi = \text{TCV}^2 + \text{BCV}^2$$

So we call $1/\mu$ the squared technical coefficient of variation, and $\phi$ the squared biological coefficient of variation, a.k.a. the *dispersion*. I.e. $\phi = \text{BCV}^2$.

- Think of BCV as the CV for just the biological component of the variability.

# The Biological Coefficient of Variation

- Because the technical variation is well understood ($\text{TCV}^2 = 1/\mu$), we can separate the variation from the read sampling process from variation in RNA abundances without technical replicates.
- Note that TCV varies inversely with sequencing depth, while BCV remains constant. This matches our expectation that TCV arises from the assay, while BCV arises from the inherent biological variability of the gene.
- TCV is easy to estimate; BCV is much harder. Efficient BCV estimation is of paramount importance to accurate differential expression results. It is a major research focus.

# Estimating the BCV

- Bad news: BCV is difficult to estimate and requires many replicates to achieve acceptable precision, so estimating BCV for each gene is not practical.
- We could precisely estimate one common BCV for all genes, but we don't really believe that all genes have the same BCV.
- We use a compromise: for each gene, take a weighted mean of the individual BCVs (accurate but unstable) and the common BCV (biased but stable) to achieve.
- Lastly, we observe that mean BCV seems to vary with abundance, so we fit a smooth spline trend instead of a single common BCV, then we combine the individual BCV with the trend.

# Estimating the BCV

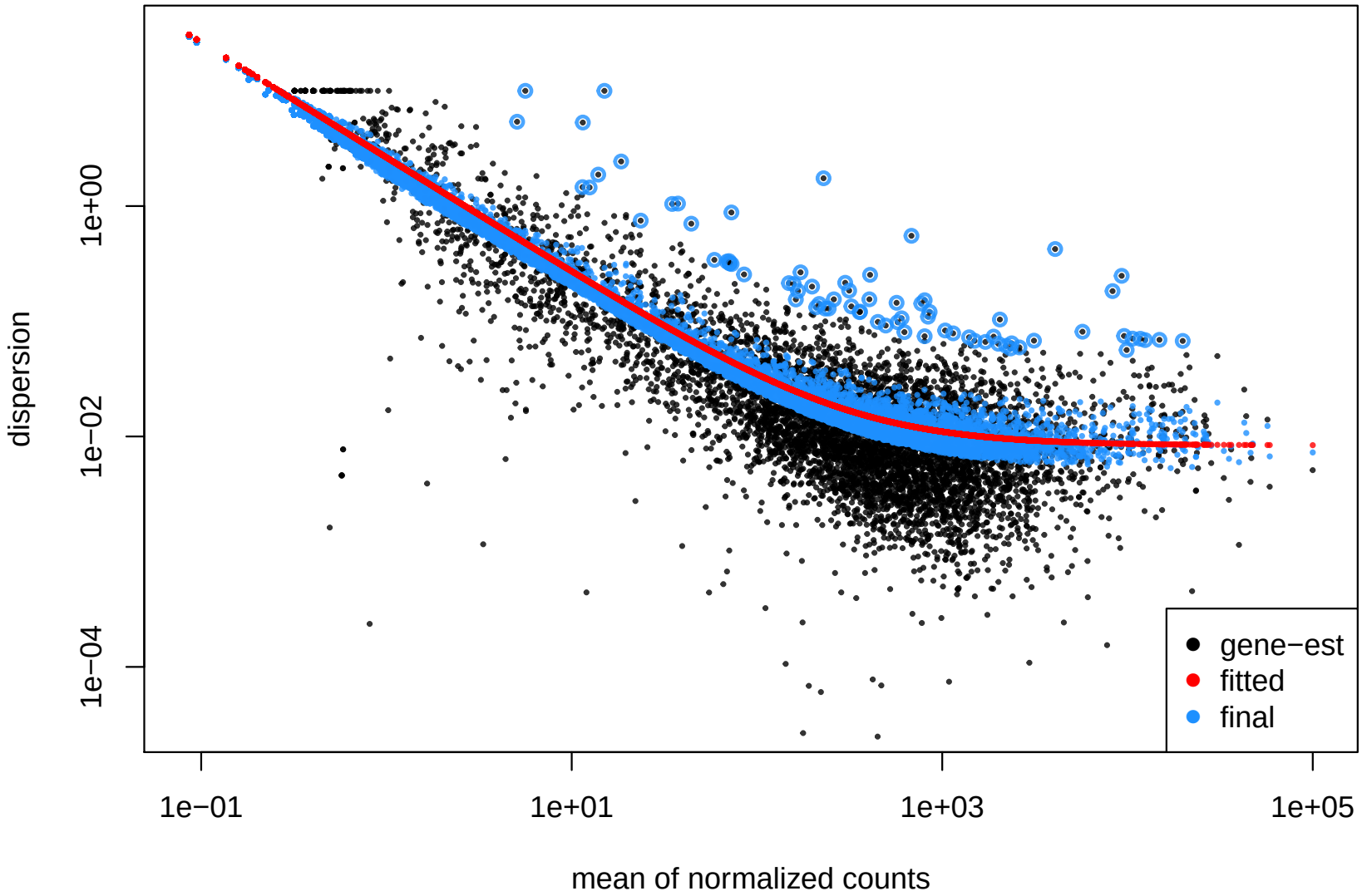


Figure 6: DESeq2 dispersion estimation

# Scaling Normalization for Count Data

# Adjusting for Compositional Bias

- Measures like RPKM and CPM account for sequencing depth
- But they do not account for compositional bias
- Because sequencing depth is limited and independent of the biology, genes compte for "sequencing space"
- If one gene goes up, all others go down
- When high-abundance genes change, they have a drastic effect on all others

# Example: CPM Fail

- Two RNA-seq samples, with a genome of 11 genes
  - In sample A, genes 1-10 have 20 counts each, and gene 11 has 0 counts
  - In sample B, genes 1-10 have 10 counts each, and gene 11 has 100 counts
- Both samples have 200 counts total
- It looks like all other genes went down 2-fold from 100,000 CPM to 50,000 CPM.
- But all that *really* happened was that gene 11 went way up
- We need a better normalization method than CPM

# Real world examples

- The previous example may seem like a pathological case, but consider:
  - Globin accounts for a very high and variable proportion of blood RNA
  - Viral RNA can account for a large fraction of all RNA in infected cells
- How can we avoid giving a few highly-abundant genes undue influence over all the others?

# A better normalization

- Instead of normalizing for total counts, normalize so that the average change is zero
- We have plenty of genes, so make the average robust by trimming the highest- and loest-abundnace genes.
- Also trim the highest and lowerst fold changes
- Result: "Trimmed mean of M-values", TMM, used by edgeR
- Alternatively, simply use the 75 percentile of M-values as a robust estimator: upper quartile normalization, used by DESeq
- In practice, both methods give extremely similar results, so it doesn't matter which one is used

# Example: TMM success

- Two RNA-seq samples, with a genome of 11 genes
  - In sample A, genes 1-10 have 20 counts each, and gene 11 has 0 counts
  - In sample B, genes 1-10 have 10 counts each, and gene 11 has 100 counts
- (This is the same example as before)
- We compute the M-values for each gene
- We throw out the top and bottom 2 M-values and top 2 highest and lowest-expressed genes (in particular, gene 11 gets thrown out on both counts)
- Take the mean of the remaining M-values
- Result is 2, so 20 counts in Sample A is equivalent to 10 counts in sample B
- DESeq normalization gives same result

# Real-world success: globin reduction

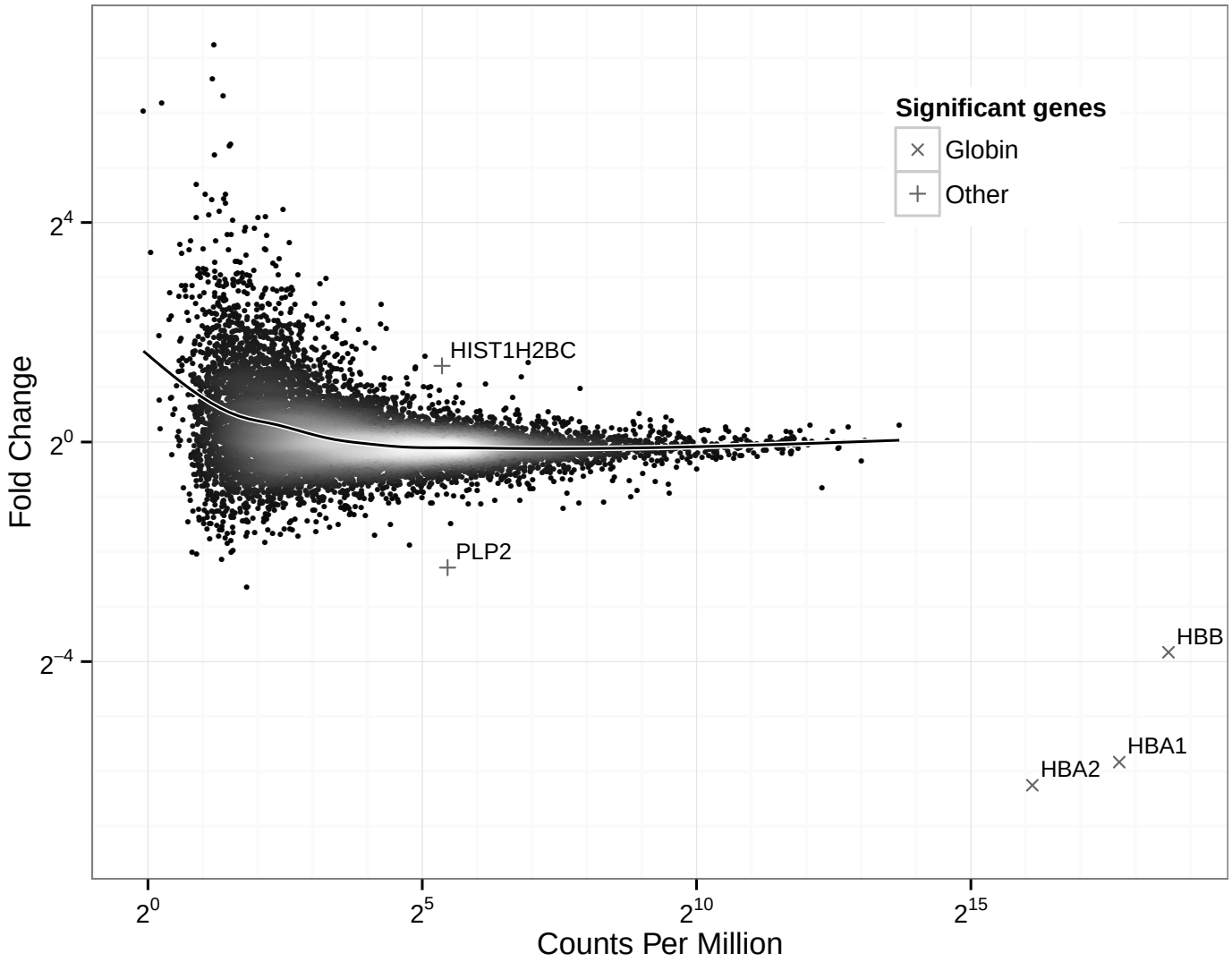


Figure 7: MA plot for GR vs no GR

# Testing for differential expression

# Generalized linear model

- Linear modeling is a generalization of t-test, ANOVA, linear regression, etc.; only works with normal distribution
  - Allows arbitrary covariates & batch effects in model
- Generalized linear modeling allows more distributions (any "exponential family" distribution)
- When BCV is held constant, NB is in the exponential family and qualifies for GLM
- Fitting GLM is done by iteratively reweighted least squares (IRLS)
  1. Fit coefficients by weighted least squares
  2. Recalculate weights based on fit
  3. Repeat until weights & coefficients converge
- edgeR and DESeq2 implement optimized IRLS algorithms tuned for fitting NB GLMs to NGS data (faster and better than base R glm implementation)

# Likelihood ratio test

To test for DE:

1. Fit the full model
2. Eliminate the coefficient(s) of interest and fit the reduced model
3. Compute the likelihood ratio (LR) between full and reduced models
4. Compute test statistic $D = 2\ln(\text{LR})$
5. Compute p-value by comparing D to $\chi^2(k)$ distribution, where k is the number of coefficients eliminated in the reduced model.

Note that edgeR also implements quasi-likelihood F-test, and DESeq2 also implements Wald test as alternatives to LRT.

# Voom: Shoehorning these counts into methods that assume a normal distribution

# RNA-seq *is* like microarrays?

Remember this?

> *"We can't shoehorn these counts into methods that assume a normal distribution" – Me, 20 minutes ago*

Well, let's forget all this negative binomial business and try to use limma anyway and see what happens.

# Why use a model we know is wrong?

- Theory: it is more important to account for the magnitude of the variance than the exact shape of the distribution.
- A linear model can be fit in one step, much more quickly than a GLM – no need to "iterate until convergence"
- Many more tools are available to work with linear models
- So, we will model the log of normalized counts with the normal distribution
- We will assume that the deviation from normality is not too bad
- However, we still have to deal with one major challenge

# Heteroskedasticity

- If counts have NB distribution, then variance is a function of the mean
- The same holds true for log counts
- Limma does empirical Bayes squeezing of all variances toward a common variance – assumes variance and mean expression are independent
- We can't ignore this problem – it gives bad results in practice
- We need to remove or account for this trend somehow

# First idea: gene-level variance modeling with a trend

- Simple idea: apply the trend idea from dispersion estimation to variance estimation here
- Fit a trend to the gene variances and squeeze toward the trend instead of a single value
- This works better than ignoring heteroskedasticity
- But what if some samples have far more sequencing depth?
- Counts will be very different for different observations in the same gene, so variances will be different too
- We want observation-level variance modeling

# Voom: variance modeling at the observation level

- Fit the same mean-variance trend to the gene-level variances as before
- Convert the trend's mean axis from CPM to a raw count scale
- Convert each fitted logCPM observation to a raw count scale
- Project each observation's fitted raw count onto the trend line to get its estimated variance
- Each observation gets the estimated variance for its fitted count
- Use precision (i.e. inverse variance) weights to counteract heteroskedasticity

# Voom: Perhaps a diagram will help

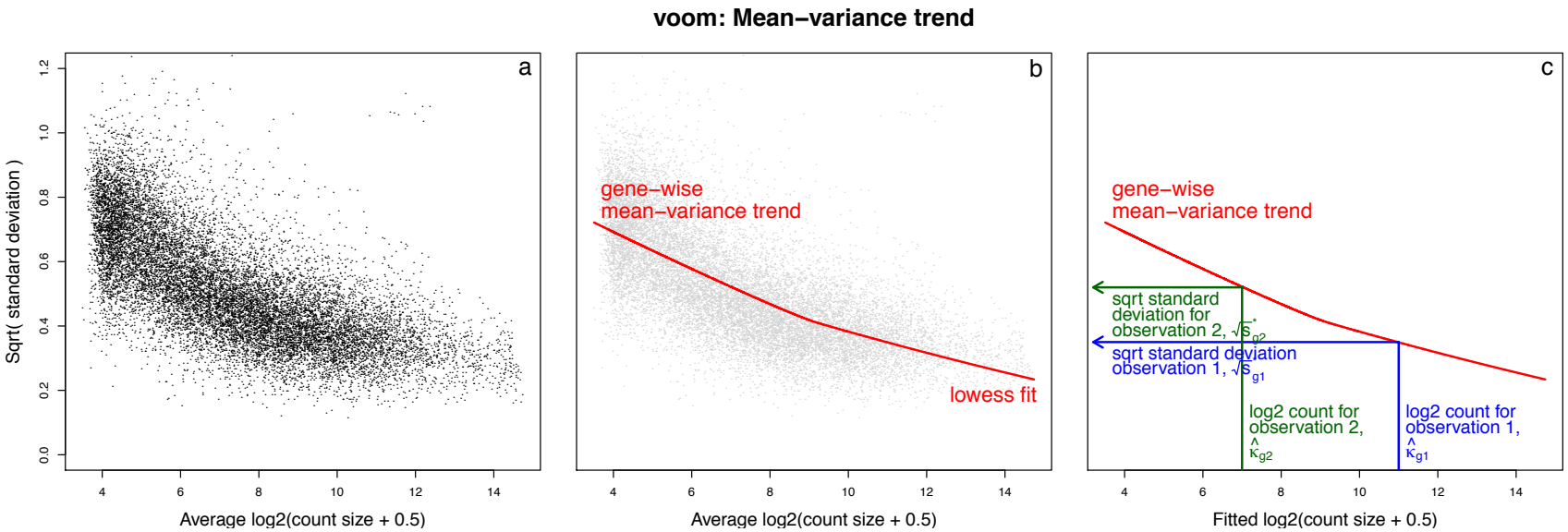


Figure 8: Diagram of voom method

# What's the difference?

- Voom sounds a lot like the gene-level trended variance modeling
- If all samples were sequenced to equal depth, voom is actually nearly equivalent
- However, when sequencing depths are very uneven, voom gives different variances to observations of differing sequencing depth
- In practice, observations from samples with greater sequencing depths will be given more weight
- This is good, because more sequencing depth yields more precise measurements

# QuickDGE: A simple but powerful interface to edgeR, DESeq, and limma for RNA-seq data

# What do these methods all have in common?

- All 3 methods:
  - operate on a matrix of counts
  - use a design matrix to specify parameters to be estimated
  - perform statistical tests of coefficients or contrasts within that design matrix
- So, to the extent that they are similar, couldn't they all be made available through a similar interface?
- Can't the code handle all the intermediate steps (normalization, BCV estimation, GLM fitting)?
- This is the design goal of QuickDGE: to provide streamlined and interchangable interfaces that still offer the full power and generality of these methods
- It's not EasyDGE: not limited to two-group comparisons, doesn't hide the complexity of experimental design, just the complexity of using these packages

# What does a QuickDGE analysis look like?

```
x <- do.edger.analysis(counts, design, tests)
y <- do.limma.analysis(counts, design, tests)
z <- do.deseq.analysis(counts, design, tests)
```

Figure 9: QuickDGE code examples

- Basic usage is identical for all three packages
- Method-specific feature are made available through optional arguments
- Results are returned in idendically-formatted tables
- Intermediate results in the analysis are also returned for auditing and exploratory analysis

# Hands-on example

Let's run through a simple example analysis using QuickDGE.

# Conclusions

- RNA-seq counts require special treatment
  - scaling normalization to asjust for compositional bias
  - Model integer counts with NB distribution
  - Estimate BCV robustly by sharing information between genes
  - Fit GLMs and LRT for hypothesis testing
- We can also estimate per-observation precision weights with voom and then analyze with standard linear modeling tools
- QuickDGE can be used to define your experimental design once and run your data through edgeR, DESeq, and limma

# Acknowledgements


- edgeR, limma, & DESeq authors for developing and documenting these methods
- The Bioconductor project for building an excellent base infrastructure for bioinformatic package development
- Sarah Lamere, Katie Podshivalova & Sunil Kurian for the use of their data
- Dan Salomon, my PI, for guidance & support

# Any Questions?