RNA-Seq

Cavan Reilly

November 13, 2019

Table of contents

RNA-seq

As was mentioned previously, we can use next generation sequencing to generate sequences from samples of RNA.

The transcriptome is the collection of all transcripts that can be generated from a genome.

Current estimates indicate that 92-94% of human transcripts with more than one exon have alternatively spliced isoforms (Wang, Sandberg, Luo, et al. *Nature*, 2008).

This means that the same gene can give rise to multiple mRNA molecules.

Some will differ because they have have different exons assembled into an mRNA, but other possibilities include differences in use of the transcription start site (TSS), and differences in the 5' and 3' untranslated regions (UTRs).

Aligning reads to the transcriptome

Differences in exon usage will result in mRNAs from the same gene leading to different proteins, hence we speak of differential coding DNA sequences (CDS).

One of the differences that arises due to differential 3'UTR usage is alternative polyadenelation patterns.

These differences in 5' and 3' UTR usage lead to mRNA molecules that have differential affinity for proteins that bind to the mature mRNA molecule and we are just starting to understand how biological processes are regulated by these subtle mechanisms.

RNA-seq

Note that there is a difference in aligning a set of reads from an RNA sample and a DNA sample: the RNA molecules will have undergone removal of introns hence they won't align directly to the genome.

If there is an annotated transcriptome for an organism you should start out trying to use this to analyze your data.

However most investigations find evidence for transcripts that are not currently annotated, even for well studied organisms like mice and humans.

Duplicates

One of the differences between using next generation sequencing for resequencing and profiling RNA levels is that with libraries prepared from RNA, one often finds duplicate reads.

In fact it is not unusual to find that 60-80% of the reads are duplicates.

Duplicates

In a resequencing study the chances of finding duplicates is quite low: for example, the human haploid genome is about 3 billion bps so the probability of observing a duplicate is about 1 in 3 billion, so with 20-40 million reads, duplicate reads seem unlikely.

Duplicate reads can arise as a PCR artifact, hence in resequencing studies duplicate reads should probably be treated as artifacts and removed.

There is a SAMtools function called `rmdup` that can be used to do this, but read the manual on its use: for example, it is designed for paired end reads by default.

Library complexity

However, when we consider RNA-seq libraries, duplicates might arise just due to low complexity of our sample.

By low complexity we mean there are not many distinct RNA molecules.

The median length of a human transcript is 2186 nucleotides (according to the UCSC hg19 annotation), so if one has a library with only 1 type of transcript and generates 20 million reads from it one is guaranteed to find duplicates.

So for analysis of RNA-seq experiments it is probably unwise to remove duplicates.

Alignment algorithms

There are a number of algorithms that are designed to align a set of short reads to the transcriptome, here is a short non-exhaustive list.

1. TopHat
2. GSNAP
3. MapSplice
4. SpliceMap
5. HMMSplicer
6. HISAT2

However it is not clear at this point if any of these really dominate the others in terms of performance.

However, TopHat and its extensions are popular, and as these build off of Bowtie and its extensions, we will use this program in this course.

TopHat

The basic idea behind TopHat is that if 2 segments of a read map to different locations on the genome then they are likely mapping to 2 different exons.

To use TopHat you need fastq files for all of our samples and the genome sequence for the organism.

Note that sometimes the data for a single sample will be in more than one fastq file.

While we can use Bowtie to align short reads to a genome, reads that span an intron, or genes that span a *splice junction*, will not align directly to the genome.

TopHat

If a read contains an internal segment that should map to an exon, Bowtie will fail to map this read to its correct genomic location as it won't match the genome outside of this exon.

For this reason, Tophat initially splits the reads up into 25 base units, does the read mapping, then reassembles reads.

Note: if you are using a set of reads that are shorter than 50 bases you should instruct TopHat to break the reads into fragments that are about half the length of your reads.

For example, if you have a set of Illumina reads that are 36 bases you should instruct TopHat to break them into fragments of length 18 (using the `--segment-length` options, like `--segment-length 18`), otherwise it may not detect any splice junctions.

TopHat

TopHat starts out by mapping the reads to the genome using Bowtie and sets aside those reads that don't map.

It then constructs a set of disjoint covered regions (it no longer uses the program Maq as it did at the time of the publication): these are taken to be the initial set of exons observed in that sample.

The program determines a consensus sequence using the reads, however in low coverage regions it will use the reference genome to determine the sequence.

Also, as the ends of exons are likely only mapped by reads that span the splice junction, the disjoint regions covered by reads are expanded (45 bases by default) using the reference sequence.

TopHat

For genes that are transcribed at low levels there could be gaps in the exons after the assembly step, hence exons that are very close are merged into a single exon.

Introns of length less than 70 bases are rare in mammalian genomes, hence by default we combine exons that are separated by less than this many bases (this can be adjusted).

The resulting set of putative exons (with their extensions from the reference genome) we will call *islands*.

The 5' end of an intron is called a *donor site* and the 3' end is called the *acceptor site*.

TopHat

This donor site almost always has the 2 bases GU at the end while the acceptor site is almost always AG, but these are sometimes GC-AG and AU-AC so current versions of TopHat also look for these acceptor and donor sites.

TopHat uses these sequence characteristics to find potential donor and acceptor site pairs in the island sequences (not just adjacent ones, but nearby ones to allow for alternative splicing).

It then looks at the set of reads that were initially unmapped and examines if any of these map to the sequence that spans the splice junction.

To conduct this mapping TopHat uses the fact that most introns are between 70 and 500,000 bases (the maximum differs from the publication): these intron sizes are based on mammals, so the user should specify values for these if using a different organism.

TopHat

It also requires that at least one read span at least 8 bases on each side of the splice junction by default, but this value can be adjusted.

It also only uses the first 28 bases from the 5' end of the read to ensure that the read is of high quality, but this too can be adjusted.

Finally the program reports a splice junction only if it occurs in more than 15% of the depth of coverage of the flanking exons (the 15% value comes from the Wang et al. 2008, *Nature* paper), but this is also adjustable as this is based on human data.

If one supplies a transcriptome to TopHat via a GTF file, the program will first map reads to the transcriptome then try to identify splice junctions for the unmapped reads.

TopHat

One can disable the subsequent search for splice junctions by specifying the -T option if you have provided a transcriptome.

If you are going to use an annotated transcriptome then TopHat will need to construct an index for this and if you are processing multiple samples TopHat will redo exactly the same calculation for each.

To save time, you should have TopHat do these calculations for the first sample and save the results.

You can do this using the `--transcriptome-index` option. This option expects a directory and a name to use in this directory.

TopHat

The exact syntax is like this: note that the `--transcriptome-index` option must be used right after the `-G` option.

```
tophat2 -o out_sample1 -G known_genes.gtf \
--transcriptome-index=transcriptome_data/known \
hg19 sample1_1.fq.z
```

Then for processing the other samples we just tell TopHat where these indices are and don't specify the -G option

```
tophat2 -o out_sample2 \
--transcriptome-index=transcriptome_data/known \
hg19 sample2_1.fq.z
```

TopHat

To use the program first get the latest version

```
[user0001@boris bin]$ wget
```

```
http://ccb.jhu.edu/software/tophat/downloads/tophat-2.1.0.Linux_x86_64.tar.gz
```

Then uncompress the binaries, enter the directory that is generated and copy the executables to your path.

```
[user0001@boris bin]$ tar zxvf tophat-2.1.0.Linux_x86_64.tar.gz
[user0001@boris tophat-2.1.0.Linux_x86_64]$ cp tophat2 $HOME/bin/tophat2
[user0001@boris tophat-2.1.0.Linux_x86_64]$ cp * $HOME/bin/
```

TopHat

Then you should test the installation, so get some test data.

```
[user0001@boris tophat-2.1.0.Linux_x86_64]$ wget
http://ccb.jhu.edu/software/tophat/downloads/test_data.tar.gz
```

Then uncompress, go into that directory and run a test

```
[user0001@boris tophat-2.1.0.Linux_x86_64]$ tar zxvf test_data.tar.gz
[user0001@boris tophat-2.1.0.Linux_x86_64]$ cd test_data
```

Then test by issuing the command (the `-r 20` option indicates that we think there are 20 bases between read pairs)

```
[user0001@boris test_data]$ tophat2 -r 20 test_ref reads_1.fq reads_2.fq
```

TopHat

and if it is working correctly this yields the following

```
[2015-11-04 11:54:03] Beginning TopHat run (v2.1.0)
-----------------------------------------------
[2015-11-04 11:54:03] Checking for Bowtie
                  Bowtie version:       2.2.6.0
[2015-11-04 11:54:03] Checking for Bowtie index files
(genome)..
       Found both Bowtie1 and Bowtie2 indexes.
...
-----------------------------------------------
[2015-11-04 11:54:08] Run complete:  00:00:05 elapsed
```

Cufflinks

While TopHat is useful for aligning a set of reads to the transcriptome, one must use some other program to generate transcript level summaries of gene expression and test for differences between groups.

One such program that is supported by the same group of researchers that develop TopHat is Cufflinks.

Cufflinks uses the output from TopHat and we will see that one can use functionality of the R package cummeRbund to examine the output and generate reports and publication quality graphs.

It is designed for paired end reads but can be used for unpaired reads.

Cufflinks

Cufflinks uses an expression for the likelihood of a set of paired end reads that depends on the length of the fragments: with paired end reads one can estimate the distribution of the lengths.

Note that there is a fundamental identifiability problem with transcript assembly that is only mitigated by the presence of reads that span splice junctions.

Suppose there are 2 exons and some reads map to both exons but we are unable to identify any reads that span splice junctions.

If there are 2 exons there are 3 possible transcripts (2 with one of each exon and a third that uses both) so if we allow for all 3 transcripts there are an infinite collection of combinations of the 3 transcript levels that are consistent with observed read counts that hit both exons.

Cufflinks

For this reason one must enforce a constraint on the transcripts: for example, we attempt to find the minimal set of transcripts that explain the exon counts.

Cufflinks quantifies the level of expression of a transcript via the number of fragments per kilobase of exon per million fragments mapped (FPKM).

This attempts to account for the fact that longer transcripts will generate more reads than shorter transcripts due to their length.

Cuffdiff is a program that is bundled with Cufflinks that allows one to test for differences between experimental conditions.

Cufflinks

The log of the ratio of FPKM values are approximately normal random variables: this allows one to define a test statistic once one determines an approximation to the standard error of this log ratio.

This standard error depends on the number of isoforms for the transcript: if there is a single isoform there is no uncertainty in the expression estimate produced by the algorithm and the only source of variability is from biological replicates.

If there are no replicates within conditions the standard error is computed by examining the variance across the 2 conditions.

If there are not many transcripts that differ in their level of expression this will provide a reasonable standard error, but if there are replicates one can obtain better estimates of the variance of the log ratio by using the negative binomial distribution.

Cufflinks

If there is only a single isoform then an approach similar to that used in the R package DESeq is used, but if there are multiple isoforms then uncertainty in the isoform expression estimates must be taken into account: this is done using a generalization of the negative binomial distribution called the beta negative binomial distribution.

Some useful details:

While both BAM and SAM files can be used for input the program requires that if chromosome names are present in the @SQ records in the header of the file then the reads must be sorted in the same order.

Make sure your chromosomes have exactly the same names in all of your indices and GTF files.

RNA-seq using TopHat and Cufflinks

We've already installed and tested the latest version of TopHat, so now let's install Cufflinks using what is now the familiar strategy: download the tarball, unpack it and write the executables to our path by copying them to bin.

```
[user0001@boris bin]$ wget
http://cole-trapnell-lab.github.io/cufflinks/assets/downloads/cufflinks-2.2.1.Linux_x86_64.tar.gz
[user0001@boris bin]$ tar zxvf cufflinks-2.2.1.Linux_x86_64.tar.gz
[user0001@boris bin]$ cd cufflinks-2.2.1.Linux_x86_64
[user0001@boris cufflinks-2.2.1.Linux_x86_64]$ cp * $HOME/bin/
```

RNA-seq using TopHat and Cufflinks

Then test the installation using some test data, so download and run.

Note that the link from the cufflinks site is broken, so download the cufflinks test data from the course website.

```
[user0001@boris cufflinks-2.2.1.Linux_x86_64]$ cufflinks test_data.sam
```

and you should see

RNA-seq using TopHat and Cufflinks

```
You are using Cufflinks v2.2.1, which is the most
recent release.
[bam_header_read] EOF marker is absent.
[bam_header_read] invalid BAM binary header (this is
not a BAM file).
File test_data.sam doesn't appear to be a valid BAM
file, trying SAM...
...
[11:36:11] Assembling transcripts and estimating
abundances.
> Processed 1 loci.  [*************************] 100%
```

Which indicates that this is working properly. This generates a GTF file called transcripts.gtf which has information on transcripts that were identified and their level of expression.

RNA-seq using TopHat and Cufflinks

As an example of using the TopHat/Cufflinks programs we will analyze the data that was used to illustrate the protocol developed by the authors of the programs (this is from the Nature Protocols paper from March 2012).

The data is already loaded on the server and we've installed all of the software, so we are almost ready to go.

We also need a set of Bowtie2 indexes and if we want to use an annotated transcriptome we need a GTF file with that information.

RNA-seq using TopHat and Cufflinks

These have already been set up in the subdirectory `/export/home/courses/ph7445/data`

The indexes are in the files with names that start with genome located at

`/export/home/courses/ph7445/data/Drosophila_melanogaster/Ensembl/BDGP5/Sequence/Bowtie2Index`

(there are 6 of these and I got them from the Illumina website).

The file Drosophila_melanogaster.BDGP5.68.gtf was obtained from ensembl.org:

`ftp://ftp.ensembl.org/pub/release-73/gtf/drosophilia_melanogaster.`

RNA-seq using TopHat and Cufflinks

First we must use TopHat to map the reads in the fastq files to the transcriptome.

Note that due to all of the options we are going to use we need to break the command up into multiple lines: we do this by ending each line with a back-slash.

```
[user0001@boris RNAseq]$ tophat2 -p 2 -G \
/export/home/courses/ph7445/data/Drosophila_melanogaster.BDGP5.68.gtf \
-o C1_R1_thout \
/export/home/courses/ph7445/data/Drosophila_melanogaster/Ensembl/BDGP5/Sequence/Bowtie2Index/genome \
/export/home/courses/ph7445/data/GSM794483_C1_R1_1.fq \

/export/home/courses/ph7445/data/GSM794483_C1_R1_2.fq
```

then you should see something like this

RNA-seq using TopHat and Cufflinks

```
[2015-11-11 14:53:01] Beginning TopHat run (v2.1.0)
-----------------------------------------------
[2015-11-11 14:53:01] Checking for Bowtie
                  Bowtie version:        2.2.6
[2015-11-11 14:53:01] Checking for Samtools
                Samtools version:        1.1
[2015-11-11 14:53:01] Checking for Bowtie index files (genome)..
[2015-11-11 14:53:01] Checking for reference FASTA file
[2015-11-11 14:53:01] Generating SAM header for
/export/home/courses/ph7445/data/Drosophila_melanogaster/Ensembl/BDGP5/Sequence/Bowtie2Index/genome
...
```

RNA-seq using TopHat and Cufflinks

Here is an explanation of the options that have been specified

- `-p` specifies the number of processors to use
- `-G` is used to give the location of the GTF file
- `-o` is used to give the desired location for the output
- last one gives the locations of the fastq files.

One then needs to issue similar commands for all of the other samples: for these you just change the name of the output directory and the name of the fastq files.

RNA-seq using TopHat and Cufflinks

For example, to process replicate 2 of condition 1 you would issue the command

```
[user0001@boris RNAseq]$ tophat2 -p 2 -G \
/export/home/courses/ph7445/data/Drosophila_melanogaster.BDGP5.68.gtf \
-o C1_R2_thout \
/export/home/courses/ph7445/data/Drosophila_melanogaster/Ensembl/BDGP5/Sequence/Bowtie2Index/genome \
/export/home/courses/ph7445/data/GSM794484_C1_R2_1.fq \

/export/home/courses/ph7445/data/GSM794484_C1_R2_2.fq
```

Then you would do this for all 3 replicates in both conditions.

RNA-seq using TopHat and Cufflinks

You will notice that the program first looks for a FASTA file holding the genome of D. melanogaster and if it can't find it reconstitutes the file from the Bowtie index (here we have obtained this along with the Bowtie indices).

As it can do this very quickly this isn't much of a concern but we will need it latter on and it is easy to generate: just issue this command.

```
[user0001@boris RNAseq]$ bowtie2-inspect
/export/home/courses/ph7445/data/Drosophila_melanogaster/Ensembl/BDGP5/Sequence/Bowtie2Index/genome
> d_melanogaster.fa
```

You should then reference this directory when you specify where your index files are located.

RNA-seq using TopHat and Cufflinks

Then we process all of the resulting BAM files using cufflinks as follows.

```
[user0001@boris RNAseq]$ cufflinks -p 2 -o C1_R1_clout
C1_R1_thout/accepted_hits.bam
```

Again `-p` controls the number of processors and `-o` gives the output directory.

Rather than typing all of these commands at the command line, it is generally better to create a *shell script* that has these instructions and is then run as a single job.

A shell script is a plain text file with linux commands, one per line.

RNA-seq using TopHat and Cufflinks

One of these is already set up-it is called `rnaseqDrosophilia.sh`-so let's take a look.

These files customarily end with the extension `.sh` and one executes the commands in such a script with the following syntax:

```
[user0001@boris RNAseq]$ sh rnaseqDrosophilia.sh
```

When processing many files one can use R to create these files (using for loops, paste and write)-this is much less error prone, you have a record of what you did and it is more time efficient for you.

A complicated script

Here is an example of code that generates a script for an RNA-seq analysis.

```
fls=list.files()
fls=fls[nchar(fls)>50]
chstr1=matrix(NA,3,11)
i1=1
dataDir="/run/media/cavan/MyBook/debesData/"
gtfDir="/home/cavan/Documents/NGS/"
for(i in seq(1,22,by=2)){
  bsnm=substr(fls[i],1,nchar(fls[i])-21)
  chstr1[1,i1]=paste("java -jar trimmomatic-0.33.jar PE -phred33 ",
    dataDir,fls[i]," ",dataDir,fls[i+1]," ",dataDir,bsnm,"_R1_trim_paired.fastq ",
    dataDir,bsnm,"_R1_trim_unpaired.fastq ",dataDir,bsnm,"_R2_trim_paired.fastq ",
    dataDir,bsnm,"_R2_trim_unpaired.fastq ILLUMINACLIP:adapters/TruSeq3-PE.fa:
    2:30:10 LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36",sep="")
  chstr1[2,i1]=paste("tophat2 -p 8 -G ",gtfDir,"Homo_sapiens.GRCh38.82.gtf -o ",
    dataDir,bsnm,"_th ",gtfDir,"Homo_sapiens.GRCh38.dna ",dataDir,bsnm,
    "_R1_trim_paired.fastq ",dataDir,bsnm,"_R2_trim_paired.fastq",sep="")
  chstr1[3,i1]=paste("cufflinks -p 8 -g ",gtfDir,"Homo_sapiens.GRCh38.82.gtf -o ",
    dataDir,bsnm,"_cl ",dataDir,bsnm,"_th/accepted_hits.bam",sep="")
  i1=i1+1
}

write.table(chstr1,"newDataProcessing.sh",quote=F,row.name=F,col.name=F)
```

RNA-seq using TopHat and Cufflinks

The next step is to run cuffmerge to merge all of the transcriptome data into a common set of transcripts.

To do this you need to create a text file that has the locations of all of the GTF files that were created running cufflinks.

RNA-seq using TopHat and Cufflinks

Given our conventions for naming the output files from the cufflinks runs this file just has the contents

```
./C1_R1_clout/transcripts.gtf
./C1_R2_clout/transcripts.gtf
./C1_R3_clout/transcripts.gtf
./C2_R1_clout/transcripts.gtf
./C2_R2_clout/transcripts.gtf
./C2_R3_clout/transcripts.gtf
```

let's call that file assemblies.txt, then one runs cuffmerge as follows

RNA-seq using TopHat and Cufflinks

```
[user0001@boris RNAseq]$ cuffmerge -p 2 \
-g /export/home/courses/ph7445/data/Drosophila_melanogaster.BDGP5.68.gtf \
-s d_melanogaster.fa \

assemblies.txt
```

This command generates a directory called `merged_asm` that contains several GTF files and files that hold the FPKM estimates.

RNA-seq using TopHat and Cufflinks

Finally you issue the cuffdiff command to test for differential expression

```
[user0001@boris RNAseq]$ cuffdiff -o diff_out \
-b d_melanogaster.fa -p 2 -L \
C1,C2 merged_asm/merged.gtf C1_R1_thout/accepted_hits.bam,\
C1_R2_thout/accepted_hits.bam,C1_R3_thout/accepted_hits.bam \
C2_R1_thout/accepted_hits.bam,C2_R2_thout/accepted_hits.bam,\

C2_R3_thout/accepted_hits.bam
```

RNA-seq using TopHat and Cufflinks

As this demonstrates, we need to instruct the program where the FASTA file is, the output directory and the number of processors as before, but now we must also include information on

- the labels to use `-L C1,C2` to denote the conditions
- the location of the merged GTF file
- the locations of all of the BAM files generated by TopHat

RNA-seq using TopHat and Cufflinks

Then this will generate the following output

```
You are using Cufflinks v2.1.1, which is the most recent release.
[17:17:55] Loading reference annotation and sequence.
Warning:  couldn't find fasta record for '2LHet'!
This contig will not be bias corrected.
Warning:  couldn't find fasta record for '2RHet'!
This contig will not be bias corrected.
...
[17:17:31] Testing for differential expression and regulation in locus.
> Processed 11618 loci.
[************************] 100%
Performed 14929 isoform-level transcription difference tests
Performed 10380 tss-level transcription difference tests
Performed 8078 gene-level transcription difference tests
Performed 11234 CDS-level transcription difference tests
Performed 2605 splicing tests
Performed 1595 promoter preference tests
Performing 1893 relative CDS output tests
...
Writing CDS-level read group tracking
Writing read group info
Writing run info
```

New protocols for using Cufflinks

While the approach just describes works fine, the program has recently changed to allow for direct output of feature summaries and to facilitate the analysis of many samples.

This is accomplished via 2 new programs: cuffquant and cuffnorm.

The program cuffquant estimates gene expression from a single bam file and produces a CXB file.

These CXB files are then submitted to cuffdiff, which makes cuffdiff run faster.

New protocols for using Cufflinks

The CXB files can then be submitted to cuffnorm.

The program cuffnorm just produces a table of gene expression values: it does no testing for differential expression.

This is useful if you just want to make some plots (e.g. heatmaps) or use some other method for testing for differential expression.

This is useful when your experiment entails more than just a 2 sample comparison.

New protocols for using Cufflinks

The output from cuffnorm can either be a gene tracking table that can be read by cummeRbund or just a tab delimited file.

To run cuffquant one gives an ouput dirrectory name and supplies a gtf file and a bam file

```
[user0001@boris RNAseq]$ cuffquant -o C1_R1_quant_out -b genome.fa -p 2
merged_asm/merged.gtf C1_R1_thout/accepted_hits.bam
```

which gives the following output

```
You are using Cufflinks v2.2.1, which is the most
recent release.
[13:16:41] Loading reference annotation and sequence.
Warning:  couldn't find fasta record for '2LHet'!
This contig will not be bias corrected.
...
```

New protocols for using Cufflinks

then run cuffdiff on this

```
[user0001@boris RNAseq]$ cuffdiff -o diff_out \
-b genome.fa -p 2 -L \
C1,C2 merged_asm/merged.gtf C1_R1_quant_out/abundances.cxb,\
C1_R2_quant_out/abundances.cxb,C1_R3_quant_out/abundances.cxb \
C2_R1_quant_out/abundances.cxb,C2_R2_quant_out/abundances.cxb,\
C2_R3_quant_out/abundances.cxb
```

which gives output like

New protocols for using Cufflinks

```
You are using Cufflinks v2.2.1, which is the most recent release.
[15:53:31] Loading reference annotation and sequence.
Warning:  couldn't find fasta record for '2LHet'!
This contig will not be bias corrected.
Warning:  couldn't find fasta record for '2RHet'!
...
> Processed 11619 loci.  [*************************] 100%
Performed 14900 isoform-level transcription difference tests
Performed 10382 tss-level transcription difference tests
Performed 8078 gene-level transcription difference tests
Performed 11232 CDS-level transcription difference tests
Performed 2606 splicing tests
Performed 1595 promoter preference tests
Performing 1895 relative CDS output tests
Writing isoform-level FPKM tracking
Writing TSS group-level FPKM tracking
...
Writing run info
```

RNA-seq using cummeRbund

We can then examine the output using the R package cummeRbund, so start up R by typing R at the linux command prompt.

Then install the package from bioconductor as usual, and when asked about a personal library just type a y and hit return.

```
> source("http://www.bioconductor.org/biocLite.R")
Bioconductor version 2.14 (BiocInstaller 1.14.2),
?biocLite for help
> biocLite("cummeRbind")
Would you like to use a personal library instead?
(y/n) y
Would you like to create a personal library
 /R/x86_64-unknown-linux-gnu-library/3.1
to install packages into?  (y/n) y
```

RNA-seq using cummeRbund

then as usual use biocLite

```
> biocLite("cummeRbund")
```

this has failed for me previously (this year too), in that event issue this command

```
> biocLite("BiocUpgrade")
```

and then the installation was fine, so now load the library and read in the output from cufflinks

```
> library(cummeRbund)
> cuff_data <- readCufflinks('diff_out')
Creating database diff_out/cuffData.db
Reading Run Info File diff_out/run.info
Writing runInfo Table
...
```

RNA-seq using cummeRbund

Then we can examine the object to see what sorts of tests we can examine.

```
> cuff_data
CuffData instance with:
        2 samples
        14700 genes
        27813 isoforms
        18216 TSS
        19291 CDS
        14700 promoters
        18216 splicing
        13364 relCDS
```

RNA-seq using cummeRbund

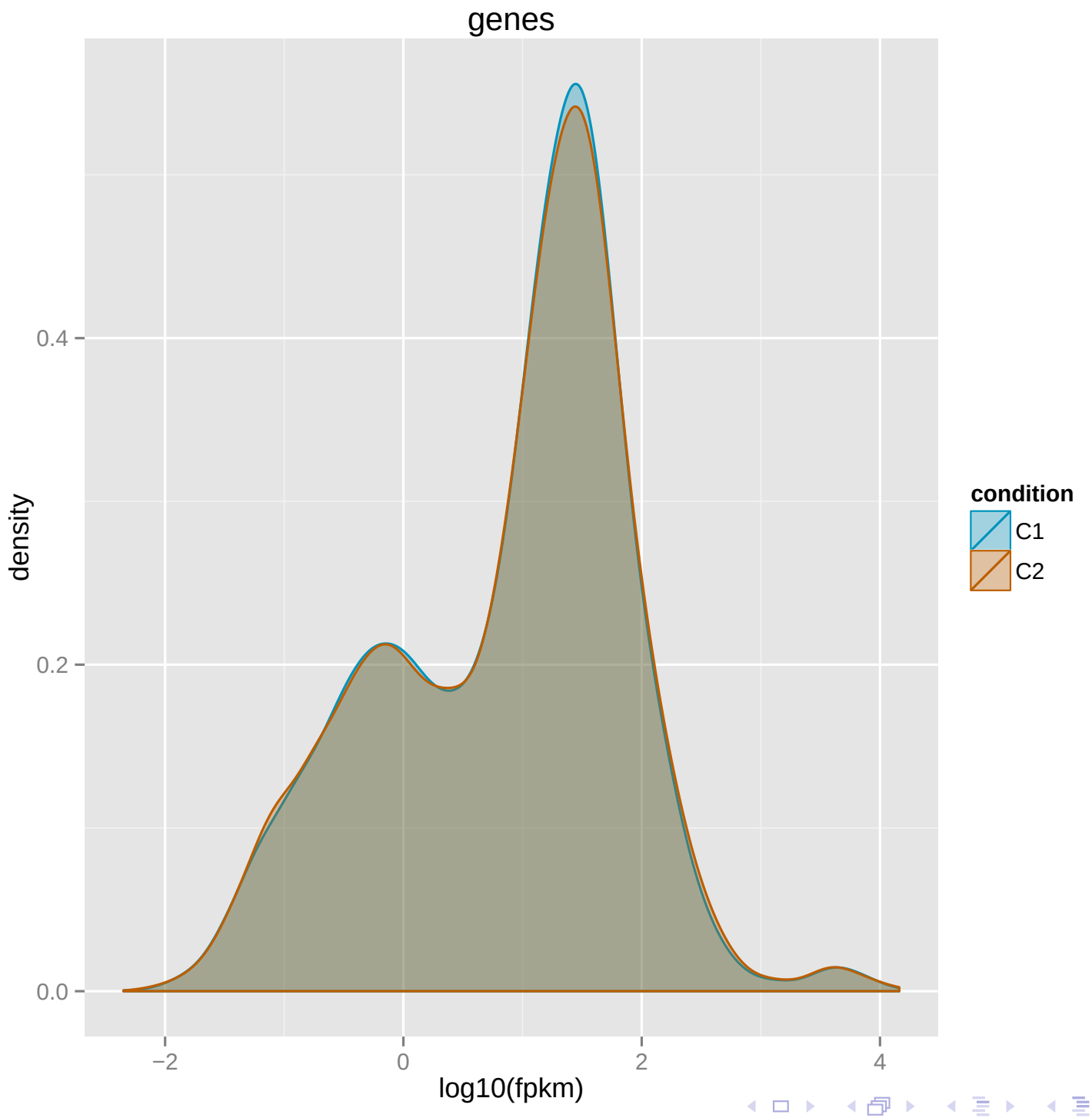
And we can test for differences in gene expression using straightforward commands as follows.

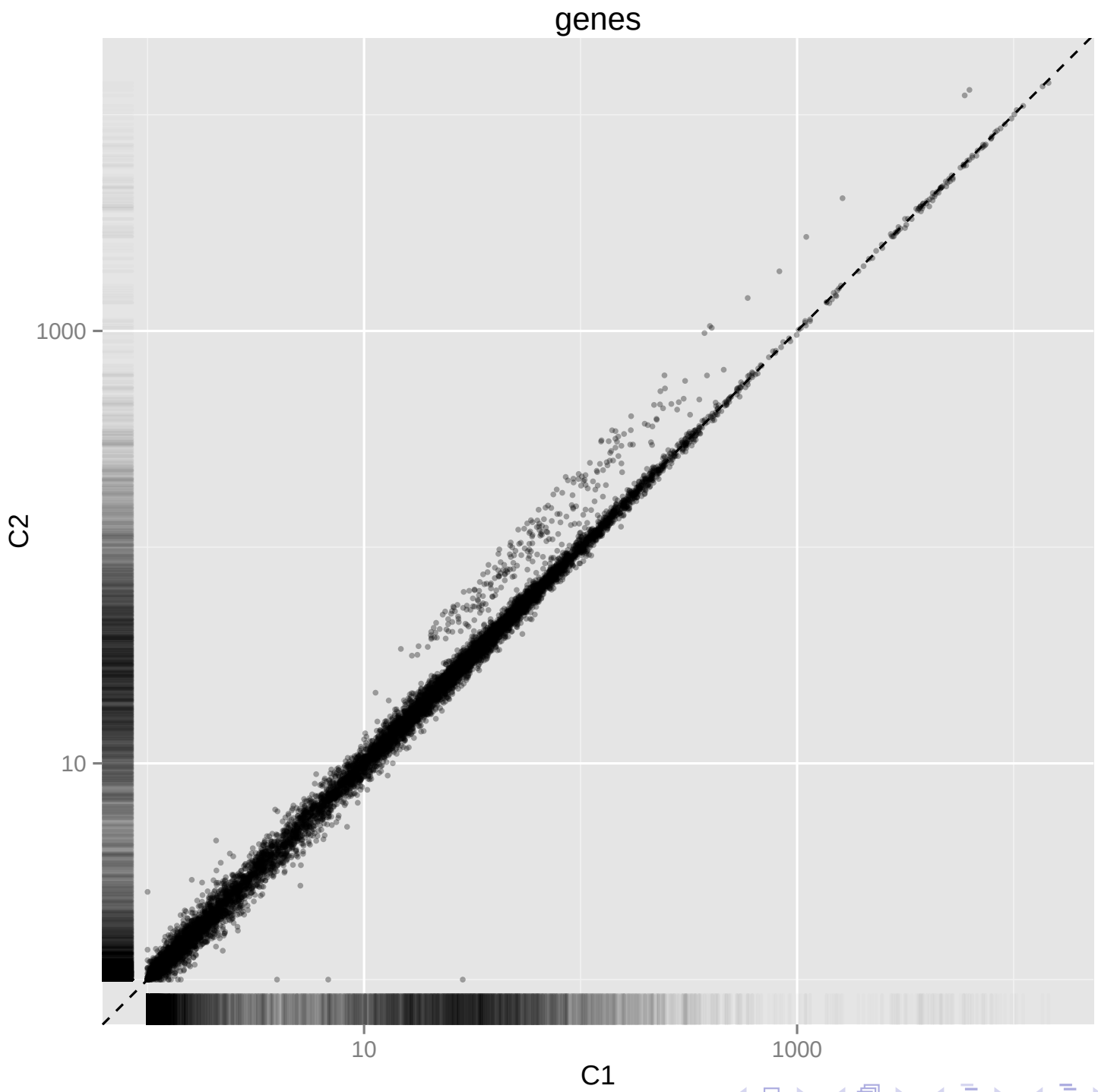
```
> gene_diff <- diffData(genes(cuff_data))
> sig_gene_diff <- subset(gene_diff,
+ significant=='yes')
> nrow(sig_gene_diff)
[1] 271
```

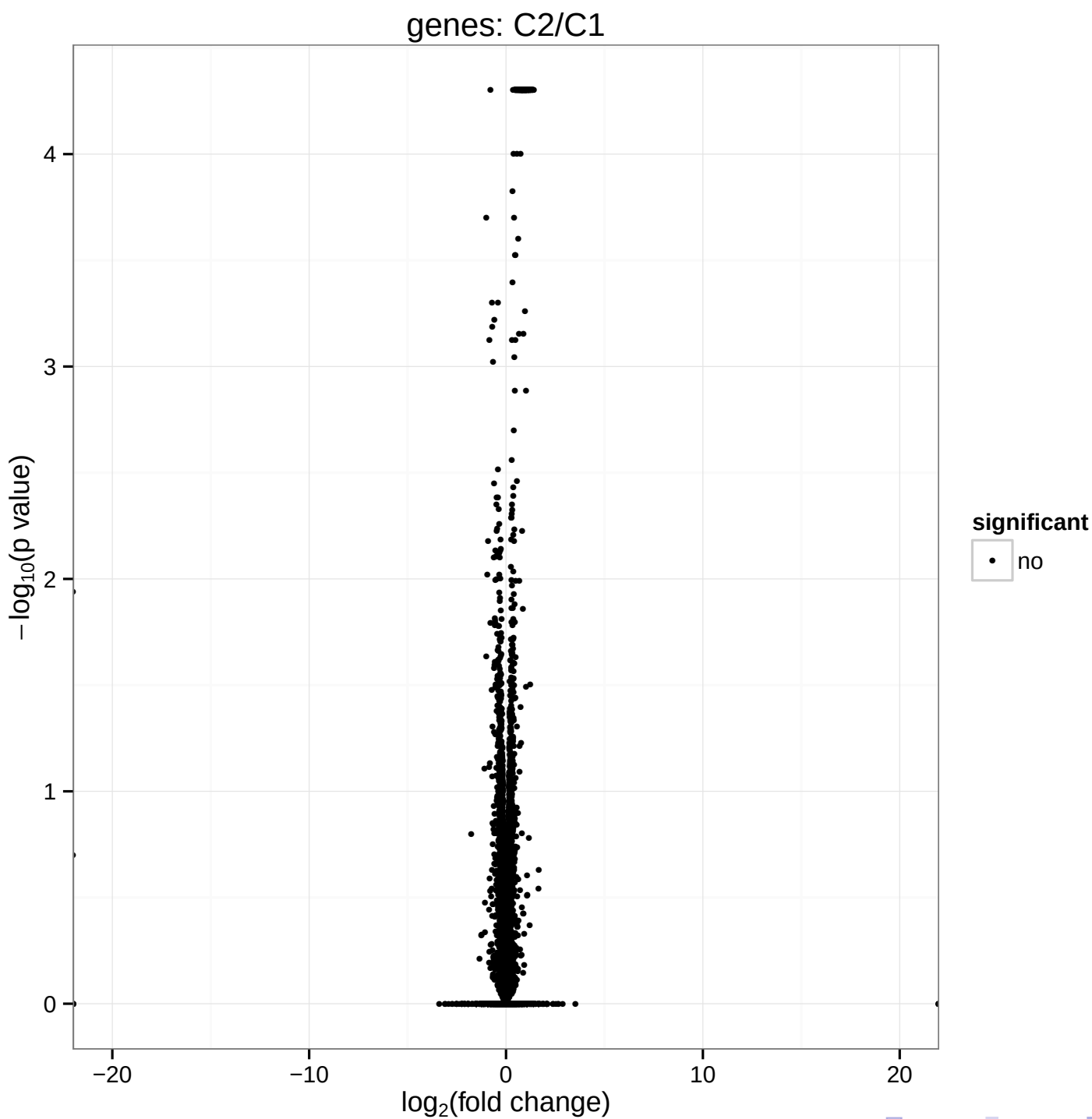
RNA-seq using cummeRbund

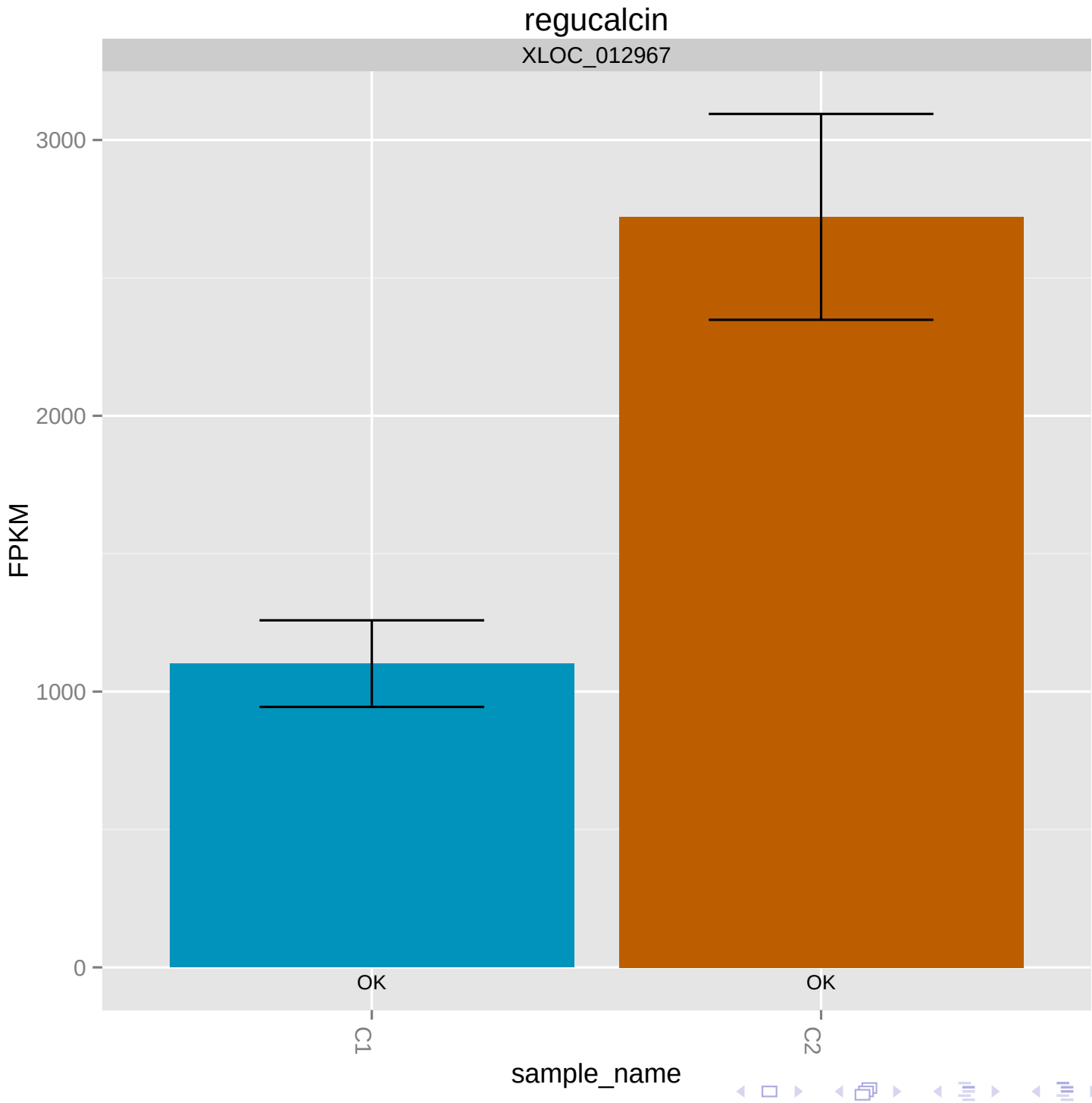
We can make some exploratory plots as follows

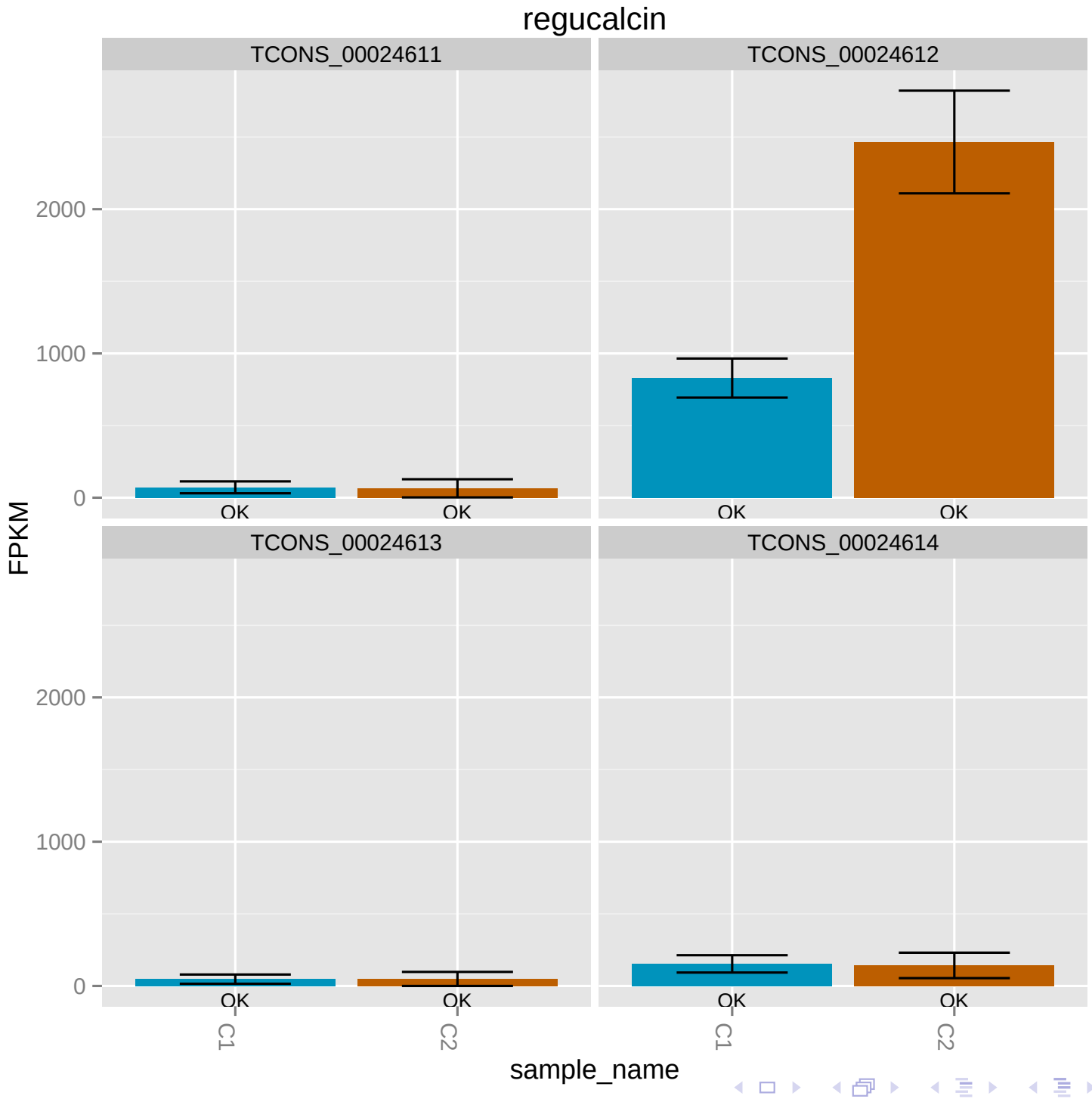
```
> pdf("explorPlt1.pdf")
> csDensity(genes(cuff_data))
> dev.off()
> pdf("explorPlt2.pdf")
> csScatter(genes(cuff_data), 'C1', 'C2')
> dev.off()
> pdf("explorPlt3.pdf")
> csVolcano(genes(cuff_data), 'C1', 'C2')
> dev.off()
> mygene <- getGene(cuff_data, 'regucalcin')
> pdf("explorPlt4.pdf")
> expressionBarplot(mygene)
> dev.off()
> pdf("explorPlt5.pdf")
> expressionBarplot(isoforms(mygene))
> dev.off()
```


genes
density
0.4
0.2
0.0
−2
0
2
4
log10(fpkm)
condition
C1
C2


genes
1000
10
C2
10
1000
C1


genes: C2/C1
4
3
2
1
0
−log10(p value)
−20
−10
0
10
20
log2(fold change)
significant
no


regucalcin
XLOC_012967
3000
2000
1000
0
FPKM
OK
OK
C1
C2
sample_name


regucalcin
TCONS_00024611
TCONS_00024612
TCONS_00024613
TCONS_00024614
FPKM
2000
1000
0
OK
C1
C2
sample_name

RNA-seq using cummeRbund

But one can go beyond just testing for differences in gene expression. We can also do tests for differences in isoform, TSS, and CDS usage as follows.

```
> isoform_diff <- diffData(isoforms(cuff_data), 'C1', 'C2')
> sig_isoform_diff <- subset(isoform_diff,
+ significant=='yes')
> nrow(sig_isoform_diff)
[1] 295
> tss_diff <- diffData(TSS(cuff_data), 'C1', 'C2')
> sig_tss_diff <- subset(tss_diff, significant=='yes')
> nrow(sig_tss_diff)
[1] 281
> cds_diff <- diffData(CDS(cuff_data), 'C1', 'C2')
> sig_cds_diff <- subset(cds_diff, significant=='yes')
> nrow(sig_cds_diff)
[1] 276
```

RNA-seq using cummeRbund

Note that the values returned by diffData are data.frames so we can examine and manipulate individual elements for further investigation.

```
> colnames(tss_diff)
 [1] "TSS_group_id"     "TSS_group_id"     "sample_1"         "sample_2"
 [5] "status"           "value_1"          "value_2"          "log2_fold_change"
 [9] "test_stat"        "p_value"          "q_value"          "significant"
```

so we can just do

```
> table(tss_diff[,12])
   no    yes
17935    281
```

RNA-seq using cummeRbund

Promoters, splicing and relative CDS are treated a little differently: one uses the distValues function as follows (this also returns a data frame so we can manipulate the results).

```
> promoter_diff <- distValues(promoters(cuff_data))
> sig_promoter_diff <- subset(promoter_diff,
+ significant=='yes')
> nrow(sig_promoter_diff)
[1] 137
> splicing_diff <- distValues(splicing(cuff_data))
> sig_splicing_diff <- subset(splicing_diff,
+ significant=='yes')
> nrow(sig_splicing_diff)
[1] 163
> relCDS_diff <- distValues(relCDS(cuff_data))
> sig_relCDS_diff <- subset(relCDS_diff,
+ significant=='yes')
> nrow(sig_relCDS_diff)
[1] 150
```

Tuxedo Suite 2.0

There is ongoing development of all of these tools: instead of Bowtie, TopHat, Cufflinks and CummeRbund...

there is a more current set of tools called HISAT2, StringTie and ballgown.

HISAT2 works almost exactly like Bowtie: there are 3 wrapper functions that allow

1. index construction
2. deduction of the sequence from a set of indices
3. alignment

HISAT2

```
$ hisat2 -p 8 --dta -x chrX_data/indexes/chrX_tran -1 chrX_data/samples/ERR188104_chrX_1.fastq.gz -2 \\
chrX_data/samples/ERR188104_chrX_2.fastq.gz -S ERR188104_chrX.sam
1292343 reads; of these:
  1292343 (100.00%) were paired; of these:
    105837 (8.19%) aligned concordantly 0 times
    1171022 (90.61%) aligned concordantly exactly 1 time
    15484 (1.20%) aligned concordantly >1 times
    ----
    105837 pairs aligned concordantly 0 times; of these:
      3907 (3.69%) aligned discordantly 1 time
    ----
    101930 pairs aligned 0 times concordantly or discordantly; of these:
      203860 mates make up the pairs; of these:
        103193 (50.62%) aligned 0 times
        98300 (48.22%) aligned exactly 1 time
        2367 (1.16%) aligned >1 times
96.01% overall alignment rate
```

RNA-seq with Tuxedo 2.0

To use the new algorithm, you first align reads to an index using hisat2 as on the last slide.

Then you use samtools to sort by position and convert to bam files.

```
$ samtools sort -@ 8 -o ERR188044_chrX.bam ERR188044_chrX.sam
```

Then you use StringTie to assemble the transcripts

```
$ stringtie -p 8 -G chrX_data/genes/chrX.gtf -o ERR188044_chrX.gtf l ERR188044_chrX.bam -l ERR188044
```

RNA-seq with Tuxedo 2.0

Then you create a text file that has the names of all of your gtf files. Here we call this mergelist.txt and this file has the following text:

```
ERR188044_chrX.gtf
ERR188104_chrX.gtf
ERR188234_chrX.gtf
ERR188245_chrX.gtf
ERR188257_chrX.gtf
ERR188273_chrX.gtf
ERR188337_chrX.gtf
ERR188383_chrX.gtf
ERR188401_chrX.gtf
ERR188428_chrX.gtf
ERR188454_chrX.gtf
ERR204916_chrX.gtf
```

RNA-seq with Tuxedo 2.0

Then you merge the results from all of the assemblies

```
$ stringtie --merge -p 8 -G chrX_data/genes/chrX.gtf \\
-o stringtie_merged.gtf chrX_data/mergelist.txt
```

RNA-seq with Tuxedo 2.0

Then you can compare to reference (this is optional and requires the program gffcompare, which is basically cuffcompare from the first Tuxedo protocol)

```
$ gffcompare -r chrX_data/genes/chrX.gtf -G -o merged \\
stringtie_merged.gtf
```

then estimate expression levels for each sample

```
$ stringtie -p 8 -G stringtie_merged.gtf -o \\
ballgown/ERR188044/ERR188044_chrX.gtf \\
ERR188044_chrX.bam -e -B
```

Then one can use ballgown or just read into R oneself.

RNA-seq with Tuxedo 2.0

The results are found in the gtf file that is produced after using StringTie with the merged.gtf file

Here is a function that can be used to read the results into R:

```
> getTx <- function(smpNm){
+   flnm=paste0("ballgown/",smpNm,"/",smpNm,"_chrX.gtf")
+   fl=read.delim(flnm,skip=2,header=F)
+   tx=fl[fl[,3]=="transcript",]
+   txid=substr(tx[,9],regexpr("transcript_id ",tx[,9]),regexpr("; cov",tx[,9])-1)
+   fpkm=as.numeric(substr(tx[,9],regexpr("FPKM ",tx[,9])+5,regexpr("; TPM",tx[,9])-1))
+   gprst=regexpr("gene_name",txid)>0
+   gname=ifelse(gprst,substr(txid,regexpr("gene_name",txid)+10,10000), substr(txid,15,10000))
+   data.frame(gname=gname,fpkm=fpkm,txid=txid)
+}
```

RNA-seq with Tuxedo 2.0

One could use that then as follows:

```
> fls=list.files()
> smpNms=substr(fls[grep("bam",fls)],1,9)
> txData=vector("list",length(smpNms))
> for(i in 1:length(smpNms)) txData[[i]]=getTx(smpNms[i])
```

RNA-seq with Tuxedo 2.0

Then you would check that all samples have same set of transcripts and then merge as follows

```
> datmat=matrix(NA,dim(txData[[1]])[1],length(smpNms))
> txid=sort(as.character(txData[[1]][,3]))
> for(i in 1:length(smpNms)){
+   datmat[,i]=txData[[i]][order(as.character(txData[[i]][,3])),2]
+ }
```

Then one has a data matrix of gene expression values and one can proceed as one wishes.

Negative binomial distribution

The Poisson distribution is commonly used as a probability model for data that arises as counts (i.e. the number of occurrences of some event).

For this reason, early RNA-Seq papers used this as a model for the number of reads mapping to some genomic feature (e.g. an exon).

However one feature of the Poisson distribution is that the mean is equal to the variance and frequently the observed variance exceeds the mean: we call this *overdispersion*.

If one assumes that the means of the count data follow a gamma distribution (which is a particular probability distribution) then if you average over the gamma distribution the distribution of the counts follows a negative binomial distribution.

Negative binomial distribution

We can do this in R with simulation:

```
> c1 <- rpois(100, 2)
> table(c1)
c1
 0  1  2  3  4  5  9
12 26 22 29  9  1  1
```

and if we look at the mean and variance:

```
> mean(c1)
[1] 2.07
> var(c1)
[1] 1.984949
```

Negative binomial distribution

Now simulate data for 100 exons so that first 50 have mean 2 and the last 50 have mean 6 and examine

```
> for(i in 1:50) exCnt[i]=rpois(1, 2)
> for(i in 51:100) exCnt[i]=rpois(1, 6)
```

then the mean and variance are still about equal if you look within exon groups with the same parameters

```
> mean(exCnt[1:50])
[1] 2.3
> var(exCnt[1:50])
[1] 1.846939
> mean(exCnt[51:100])
[1] 6.14
> var(exCnt[51:100])
[1] 7.265714
```

Count based methods

but

```
> mean(exCnt)
[1] 4.22
> var(exCnt)
[1] 8.23394
```

the variance is larger than the mean.

RNA-seq data analysis

Now, rather than just using 2 distinct values for the means suppose that they are generated from a gamma distribution, then use these simulated means to simulate data using the Poisson distribution

```
> mSim <- rgamma(100,5,3)
> exCnt <- rpois(100, mSim)
> mean(exCnt)
[1] 1.87
> var(exCnt)
[1] 2.599091
```

So here, the distribution of exCnt is negative binomial.

RNA-seq data analysis

There are 2 popular packages (called edgeR and DESeq) that assume that one has data in the form of counts of the number of times a read mapped to a genomic feature (e.g. a gene or a transcript).

The counts are then modeled as being independently distributed according to the negative binomial distribution.

The negative binomial distribution has 2 parameters: one determines the mean and the other, called a dispersion, determines how much bigger the variance is than the mean.

Both packages first estimate the size of the library (in slightly different ways) then use a regularized estimate of the dispersion (like variance estimation in microarray analysis).

RNA-seq data analysis

The differences are:

1. the packages use different methods for estimating the library sizes (i.e. the sequencing depth in the different samples)
2. the DESeq package assumes that the dispersions are related to the means and uses this relationship to estimate the feature level dispersions.
3. the DESeq package uses a computationally faster and simpler method for estimating the dispersion parameters (but relies on an approximation that the sum of 2 negative binomial random variables is itself distributed according to the negative binomial distribution)
4. the DESeq package provides a set of graphics functions for model diagnostics

RNA-seq data analysis

The authors of the DESeq package claim that the second property is what makes their method superior because it allows their algorithm to have equal power over the entire range of dispersions.

This makes edgeR conservative for highly expressed genes and anticonservative for genes expressed at low levels.

A mathematical proof of this claim isn't presented, but this is seen when analyzing data and in simulations.

RNA-seq data analysis

We can also get data that has been aligned and for which feature counts are available from GEO.

As an example, we will use data from the experiment in which 2 groups of cows were under different levels of stress and liver biopsies were obtained near the time of calving (we saw this when discussing quality assessment).

The table of counts is on the server, so we can just read in the data.

```
> bovCnts <- read.table("/export/home/courses/ph7445/data/bovineCounts.txt")
> dim(bovCnts)
[1] 21996 11
```

Now set up an indicator for group membership.

```
grp <- factor(c(rep(1,5),rep(2,6)))
```

RNA-seq data analysis

We will next apply a filter to the data and exclude transcripts where the minimum count across all subjects is less than 5

```
> f1 <- apply(bovCnts,1,min)
> sum(f1>4)
[1] 10990
> bovCntsF <- bovCnts[f1 > 4 ,]
> dim(bovCntsF)
[1] 10990 11
```

Now we will use the edgeR package to test for differential expression.

RNA-seq with edgeR

After installing and loading the edgeR package from Bioconductor we first initialize our DGE list, then estimate the mean dispersion parameter and then use this to get the individual dispersions.

```
> delist <- DGEList(counts=bovCntsF, group=grp)
Calculating library sizes from column totals.
> delist <- estimateCommonDisp(delist)
> delist <- estimateTagwiseDisp(delist)
```

Then we conduct the exact test that is similar to Fisher's exact test.

```
> et <- exactTest(delist)
```

RNA-seq data analysis

Then we examine the result

```
> head(et$table)
        logFC   logCPM      PValue
2  0.06505177 5.056512 0.779727454
4  0.70017935 2.995358 0.037290862
5  0.32148049 6.605787 0.049108650
6  0.43768318 8.275121 0.002175286
7 -0.30086851 3.758206 0.345883682
8 -0.08476951 3.576230 0.813614805
```

RNA-seq data analysis

Then we use the Benjamini Hochberg method to correct for multiple hypothesis testing.

```
> padj <- p.adjust(et$table[,3], method="BH")
> sum(padj<0.05)
[1] 130
```

So we are detecting 130 differences in transcript levels while controlling the FDR at 5%.

For this example there doesn't appear to be any connection between the estimated level of gene expression (in terms of the average $\log_2$ of the counts per million) and the p-values.

```
> cor(et$table[,2], et$table[,3])
[1] -0.02783438
```

RNA-seq data analysis

We can also use the DEseq package in a similar fashion: initialize the count data set, estimate library sizes, estimate dispersions and then conduct a test.

There is a newer version (called DESeq2) that differs in some internal details, uses different statistical tests and does some outlier remediation.

```
> cds <- newCountDataSet(bovCntsF, grp)
> cds <- estimateSizeFactors(cds)
> cds <- estimateDispersions(cds)
> res <- nbinomTest(cds, "1", "2")
```

RNA-seq data analysis

Next we examine res

```
> head(res)
  id  baseMean baseMeanA  baseMeanB foldChange log2FoldChange       pval
1  2 104.14912 102.96024  105.13985  1.0211695     0.03022234 0.92734232
2  4  22.14239  16.58914   26.77009  1.6137116     0.69038273 0.08921768
3  5 305.48155 271.18398  334.06285  1.2318679     0.30084759 0.19480965
4  6 976.65013 828.93664 1099.74470  1.3266933     0.40783491 0.02163098
5  7  41.14755  46.32049   36.83677  0.7952586    -0.33050401 0.35883953
6  8  36.31335  37.45815   35.35935  0.9439694    -0.08318802 0.69298641
       padj
1 1.0000000
2 1.0000000
3 1.0000000
4 0.8406186
5 1.0000000
6 1.0000000
```

RNA-seq data analysis

Then we can examine the adjusted p-values (these use the Benjamini Hochberg adjustment) and see that there are fewer differences when we use this method compared to edgeR.

```
> sum(res[,8]<.05)
[1] 42
```

In fact, almost every transcript identified by DESeq was also detected to differ using edgeR.

```
> table(res[,8]<.05, padj<0.05)
       FALSE  TRUE
  FALSE 10859    89
  TRUE      1    41
```

We can also make a plot to investigate how the dispersions depend on the mean (which is the crucial difference between DESeq and edgeR).

RNA-seq data analysis

To this end create the following function,

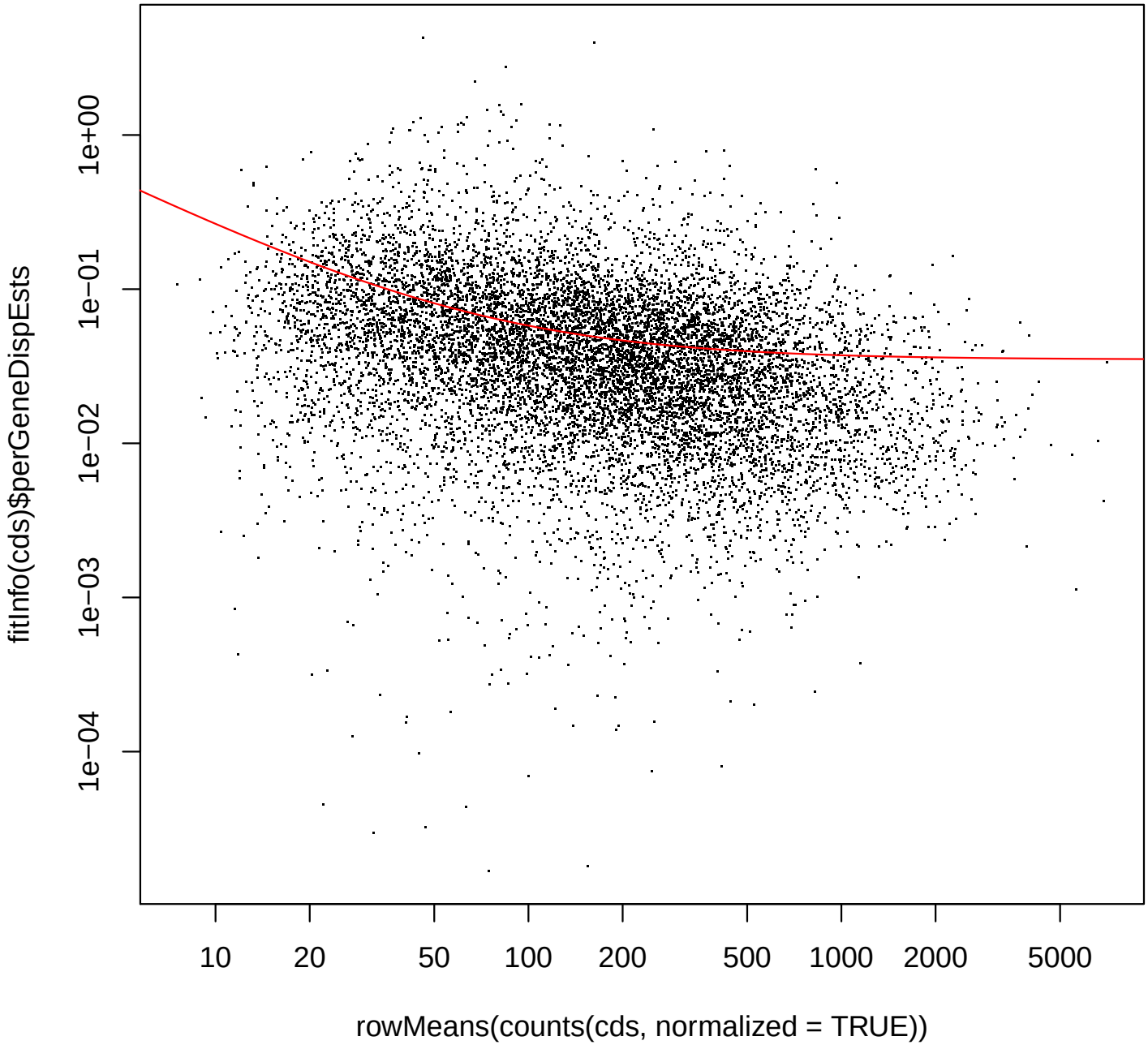
```
plotDispEst <- function(cds){
  plot(rowMeans(counts(cds, normalized=TRUE)),
    fitInfo(cds)$perGeneDispEsts, pch='.', log="xy")
  xg <- 10^seq(-.5, 5, length.out=300)
  lines(xg, fitInfo(cds)$dispFun(xg), col="red")
}
```

RNA-seq data analysis

and use it as follows

```
> pdf("DESeqPlot.pdf")
> plotDispEst(cds)
Warning message:
In xy.coords(x, y, xlabel, ylabel, log) :
  474 y values <= 0 omitted from logarithmic plot
> dev.off()
```

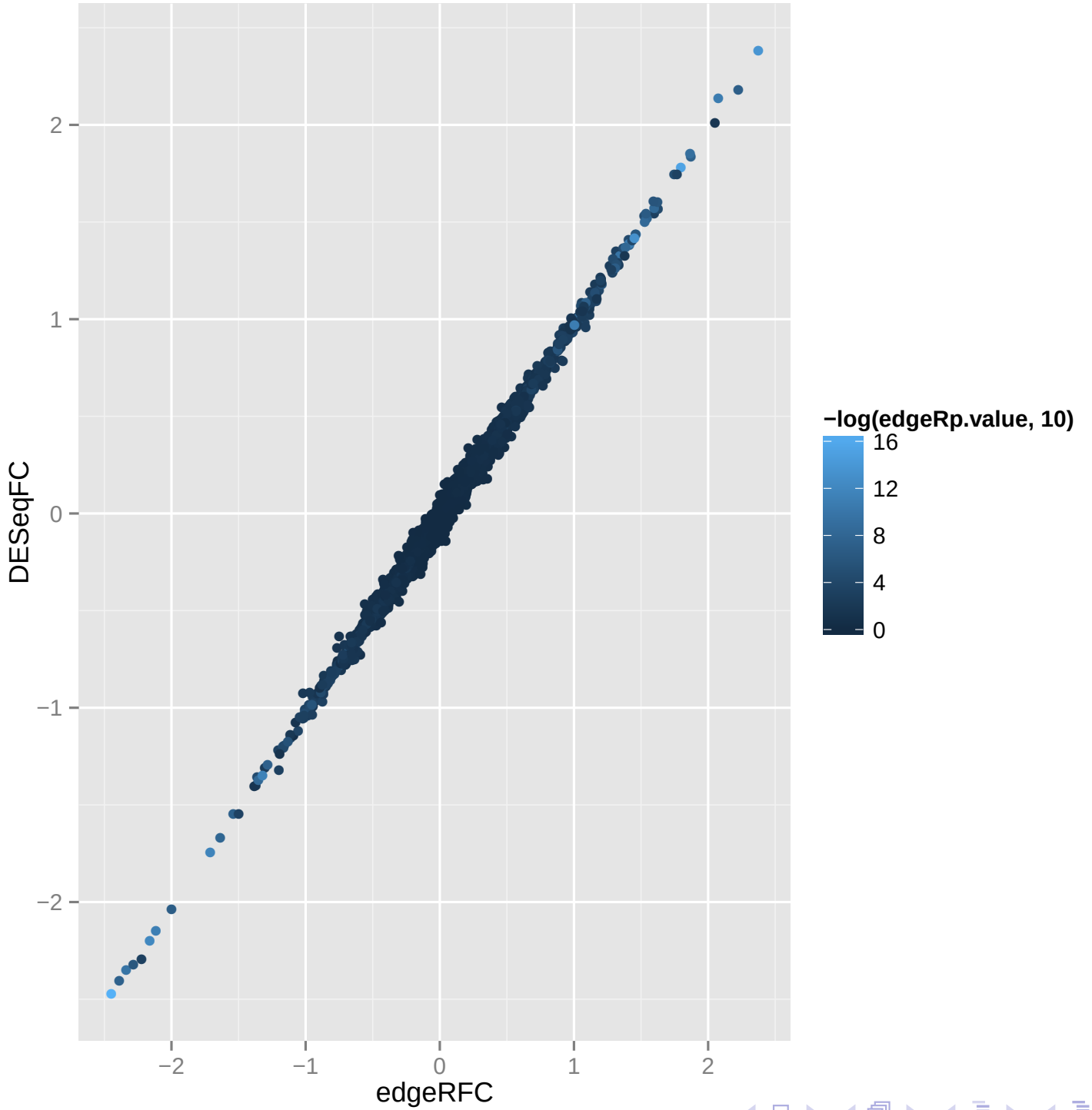
Then we see for this data set there is not a very strong relationship between the means and the estimated dispersions.


fitInfo(cds)$perGeneDispEsts
1e+00
1e−01
1e−02
1e−03
1e−04
10
20
50
100
200
500
1000
2000
5000
rowMeans(counts(cds, normalized = TRUE))

RNA-seq data analysis

We can also examine the association between the estimated fold changes produced by the 2 methods (they are highly correlated).

```
> cor(2^et$table[,1], res[,5])
[1] 0.9978404
> edgeRFC <- et$table[,1]
> edgeRp.value <- et$table[,3]
> DESeqFC <- res[,6]
> pdf("edgeRvsDESeq.pdf")
> qplot(edgeRFC, DESeqFC, colour= -log(edgeRp.value,10))
> dev.off()
```


DESeqFC
edgeRFC
2
1
0
−1
−2
−2
−1
0
1
2
−log(edgeRp.value, 10)
16
12
8
4
0