Subread/Rsubread Users Guide

Rsubread v1.12.0/Subread v1.4.0

14 October 2013

Wei Shi and Yang Liao

Bioinformatics Division
The Walter and Eliza Hall Institute of Medical Research
Melbourne, Australia

Contents

Chapter 1

Introduction

This document describes in details the programs included in the `Subread` software suite, including read alignment, junction detection, read summarization and SNP detection.

The `Subread` aligner is a superfast, sensitive and accurate read aligner[1]. It employs a mapping paradigm called "seed-and-vote" [1], which is fundamentally different from the "seed-and-extend" paradigm used by many read aligners. The "seed-and-vote" paradigm extracts a number of subreads (16 mers) from the read and then uses these subreads to vote for the mapping location of the read, rather than performing an computational expensive extension operation to determine the location of the read like "seed-and-extend" does. The power and flexibility of the new mapping paradigm enables `Subread` to achieve a superior efficiency without losing accuracy and sensitivity. This paradigm is especially powerful for the alignment of RNA-seq data because it can automatically determines if the reads should be globally aligned or locally aligned. This paradigm also enables a highly accurate detection of insertions and deletions, via using perfectly matched subreads flanking the indels to call them.

`Subread` supports a variety of sequencing platforms including Illumina GA/HiSeq, ABI SOLiD, Life Science 454, Helicos Heliscope and Ion Torrent. It can align short reads, long reads and reads of variable lengths. It has been found to be useful in a number of high-profile studies [2, 3, 4, 5, 6]

The `subjunc` program included in this package is designed to detect exon-exon junctions and to perform full alignments for RNA-seq reads. It also take advantage of the powerful "seed-and-vote" paradigm to achieve a highly accurate junction detection and read mapping [1]. It outputs chromosomal locations of discovered exon-exon junctions, number of reads supporting these junctions and results of full alignments performed for reads.

The featureCounts program is designed to assign mapped reads or fragments (for paired-end data) to genomic features such as genes, exons and promoters. It is a light-weight read counting program suitable for count both gDNA-seq and RNA-seq reads for genomic features[7].

Also included in this software suite is a very efficient and accurate SNP caller – `exactSNP`. `exactSNP` measures local background noise for each candidate SNP and performs Fisher's Exact test to confidently call SNPs. This approach effectively removes false positives arising from sequencing errors and mapping errors.

Programs included in the `Subread` software suite have also been implemented in the Bioconductor package `Rsubread`, providing R users easy access to these programs via their fimiliar programming environment.

Chapter 2

Preliminaries

2.1 Citation

If you use Subread or Subjunc aligners, please cite:

> Liao Y, Smyth GK and Shi W. The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote. Nucleic Acids Research, 41(10):e108, 2013
> http://nar.oxfordjournals.org/content/early/2013/04/03/nar.gkt214.abstract

If you use featureCounts, please cite:

> Liao Y, Smyth GK and Shi W. featureCounts: an efficient general-purpose read summarization program. arXiv:1305.3347, 2013
> http://arxiv.org/abs/1305.3347

2.2 Download and installation

2.2.1 Subread package

Download the Subread package from http://subread.sourceforge.net and type the following command to uncompress it:

```
tar zxvf subread-1.x.x.tar.gz
```

Enter the `src` subdirectory under the home directory of this package and then issue the following command to install it on a Linux operating system:

```
make -f Makefile.Linux
```

To install it on a Mac OS X operating system, issue the following command:

```
make -f Makefile.MacOS
```

To install it on a FreeBSD operating system, issue the following command:

```
make -f Makefile.FreeBSD
```

To install it on Oracle Solaris or OpenSolaris computer operating systems, issue the following command:

```
make -f Makefile.SunOS
```

A new subdirectory called `bin` will be created under the home directory of the software package, and the executables generated from the compilation will be saved to that subdirectory. To enable easy access to these executables, you may copy them to a system directory such as `/usr/bin` or add the path to them to your search path (your search path is usually specified in the environment variable `'PATH'`).

2.2.2 Bioconductor Rsubread package

You have to get R installed on my computer to install this package. Lauch an R session and issue the following command to install it:

```
source("http://bioconductor.org/biocLite.R")
biocLite("Rsubread")
```

Alternatively, you may download the Rsubread source package directly from http://bioconductor.org/packages/release/bioc/html/Rsubread.html and install it to your R from the source.

2.3 How to get help

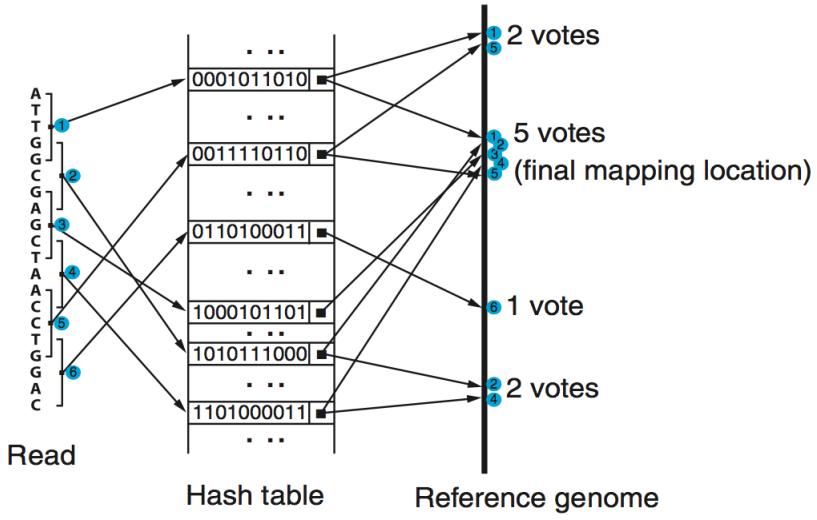
Bioconductor mailing list (`http://bioconductor.org/`) and SeqAnswer forum (`http://www.seqanswers.com`) are the best places to get help and report bugs. Alternatively, you may contact Wei Shi (shi at wehi dot edu dot au) directly.

Chapter 3

The seed-and-vote mapping paradigm

3.1 Seed-and-vote

We have developed a new read mapping paradigm called "seed-and-vote" for efficient, accurate and scalable read mapping [1]. The seed-and-vote strategy uses a number of overlapping seeds from each read, called *subreads*. Instead of trying to pick the best seed, the strategy allows all the seeds to vote on the optimal location for the read. The algorithm then uses more conventional alignment algorithms to fill in detailed mismatch and indel information between the subreads that make up the winning voting block. The following figure illustrates the proposed seed-and-vote mapping approach with an toy example.

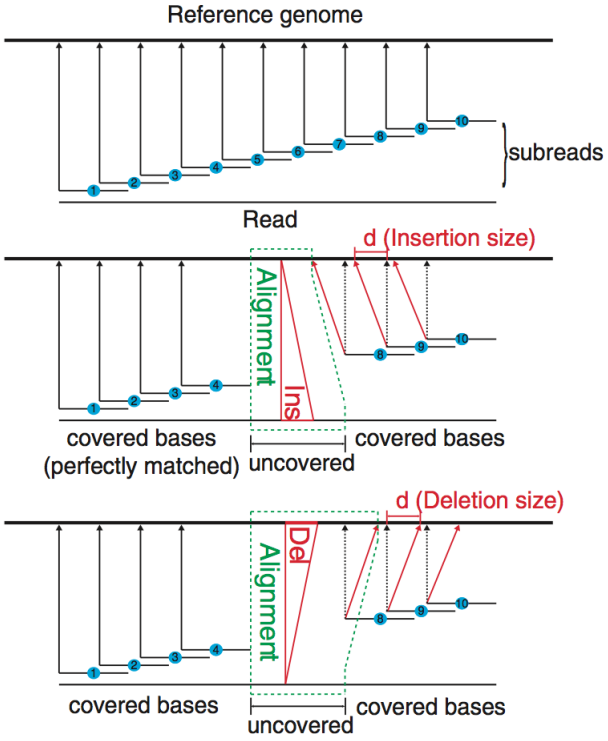


Two aligners have been developed under the seed-and-vote paradigm, including `Subread` and `Subjunc`. `Subread` is a general-purpose read aligner, which can be used to map both genomic DNA-seq and RNA-seq read data. Its running time is determined by the number of *subreads* extracted from each read, not by the read length. Thus it has an excellent maping scalability, ie its running time has only very modest increase with the increase of read length.

`Subread` uses the largest mappable region in the read to determine its mapping location, therefore it automatically determines whether a global alignment or a local alignment should be found for the read. For the exon-spanning reads in a RNA-seq dataset, `Subread` performs local alignments for them to find the target regions in the reference genome that have the largest overlap with them. Note that `Subread` does not perform global alignments for the exon-spanning reads and it soft clips those read bases which could not be mapped. However, the `Subread` mapping result is sufficient for carrying out the gene-level expression analysis using RNA-seq data, because the mapped read bases can be reliably used to assign reads, including both exonic reads and exon-spanning reads, to genes.

To get the full alignments for exon-spanning RNA-seq reads, the `Subjunc` aligner can be used. `Subjunc` is designd to discover exon-exon junctions from using RNA-seq data, but it performs full alignments for all the reads at the same time. The `Subjunc` mapping results should be used for detecting genomic variations in RNA-seq data, allele-specific expression analysis and exon-level gene expression analysis. The Section 3.3 describes how exon-exon junctions are discovered and how exon-spanning reads are aligned using the seed-and-vote paradigm.

3.2 Indel detection



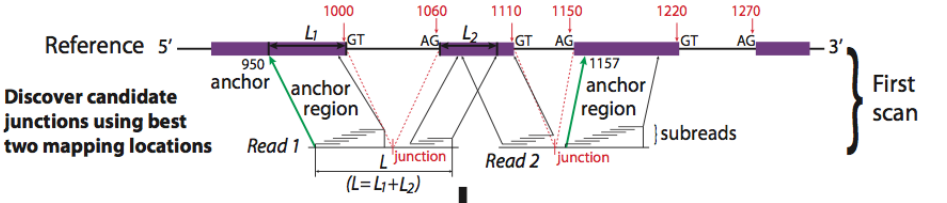
The seed-and-vote paradigm is very powerful in detecting indels (insertions and deletions). The figure below shows how we use the *subreads* to confidently detect indels. When there is an indel existing in a read, mapping locations of subreads extracted after the indel will be shifted to the left (insertion) or to the right (deletion), relative to the mapping locations of

subreads at the left side of the indel. Therefore, indels in the reads can be readily detected by examining the difference in mapping locations of the extracted subreads. Moreover, the number of bases by which the mapping location of subreads are shifted gives the precise length of the indel. Since no mismatches are allowed in the mapping of the subreads, the indels can be detected with a very high accuracy.

3.3 Junction detection

The seed-and-vote paradigm is also very useful in detecting exon-exon junctions, because the short subreads extracted across the entire read can be used to detect short exons in a sensitive and accurate way. The figure below shows the schematic of detecting exon-exon junctions and mapping RNA-seq reads by `Subjunc`, which uses this paradigm.

The first scan detects all possible exon-exon junctions using the mapping locations of the subreads extracted from each read. Matched donor and receptor sites are required for calling junctions. Exons as short as 16bp can be detected in this step. The second scan verifies the putative exon-exon junctions discovered from the first scan by performing re-alignments for the junction reads. The output from `Subjunc` includes the list of verified junctions and also the mapping results for all the reads.

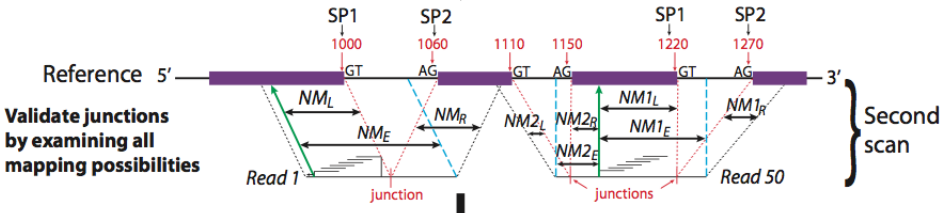


Putative junctions

Chr	Splicing Point 1 (SP1)	Splicing Point 2 (SP2)
1	1000	1060
1	1100	1150
1	1220	1270
•••	•••	•••
10	500	10,000
•••	•••	•••

Anchor information

Reads	Chr	Location in genome	Location in read	Region length
Read 1	1	950	5	46
Read 2	1	1157	44	52
•••	•••	•••	•••	•••
Read 50	1	1168	26	46
•••	•••	•••	•••	•••



Validated junctions

Chr	SP1	SP2	# supporting reads	
•••	•••	•••	•••	
1	1000	1060	20	✓
1	1110	1150	22	✓
1	1220	1270	20	✓
•••	•••	•••	•••	
10	500	10,000	0	X
•••	•••	•••	•••	

Read mapping results

Reads	Chr	Location	CIGAR
Read 1	1	945	55M50N45M
Read 2	1	1073	38M30N62M
•••	•••	•••	•••
Read 50	1	1102	9M40N70M50N5M1D16M
•••	•••	•••	•••

Chapter 4

Mapping reads generated by genomic DNA sequencing technologies

4.1 A quick start for SourceForge Subread

An index must be built for the reference first and then the read mapping can be performed.

Step 1: Building an index

Build a base-space index (default). You can provide a list of FASTA files or a single FASTA file including all the reference sequences.

```
subread-buildindex -o my_index chr1.fa chr2.fa ...
```

Step 2: Aligning the reads

Map single-end reads using 5 threads:
`subread-align -T 5 -i my_index -r reads.txt -o subread_results.sam`

Detect indels of up to 16bp:
`subread-align -I 16 -i my_index -r reads.txt -o subread_results.sam`

Report up to three best mapping locations:
`subread-align -B 3 -i my_index -r reads.txt -o subread_results.sam`

Report uniquely mapped reads only:
`subread-align -u -i my_index -r reads.txt -o subread_results.sam`

Map paired-end reads:
`subread-align -d 50 -D 600 -i my_index -r reads1.txt -R reads2.txt -o subread_results.sam`

4.2 A quick start for Bioconductor Rsubread

An index must be built for the reference first and then the read mapping can be performed.

Step 1: Building an index

To build the index, you must provide a single FASTA file (eg. "genome.fa") which includes all the reference sequences.

```
library(Rsubread)
buildindex(basename="my_index",reference="genome.fa")
```

Step 2: Aligning the reads

Map single-end reads using 5 threads:

```
align(index="my_index",readfile1="reads.txt",output_file="rsubread.sam",nthreads=5)
```

Detect indels of up to 16bp:

```
align(index="my_index",readfile1="reads.txt",output_file="rsubread.sam",indels=16)
```

Report up to three best mapping locations:

```
align(index="my_index",readfile1="reads.txt",output_file="rsubread.sam",nBestLocations=3)
```

Report uniquely mapped reads only:

```
align(index="my_index",readfile1="reads.txt",output_file="rsubread.sam",unique=TRUE)
```

Map paired-end reads:

```
align(index="my_index",readfile1="reads1.txt",readfile2="reads2.txt",output_file="rsubread.sam",
minFragLength=50,maxFragLength=600)
```

4.3 Index building

The `subread-buildindex` (`buildindex` function in Rsubread) program builds an base-space or color-space index using the reference sequences. The reference sequences should be in FASTA format (the header line for each chromosomal sequence starts with ">").

This program extracts all the 16 mer sequences from the reference genome at a 2bp interval and then uses them to build a hash table. Keys in the hash table are unique 16 mers and values are their chromosomal locations. Table 1 describes the arguments used by the `subread-buildindex` program.

Table 1: Arguments used by the `subread-buildindex` program (`buildindex` function in Rsubread). Arguments in parenthesis in the first column are used by `buildindex`.

Arguments	Description
-o < *basename* > (`basename`)	Specify the base name of the index to be created.
-f < *int* > (`TH_subread`)	Specify the threshold for removing uninformative subreads (highly repetitive 16mers). Subreads will be excluded from the index if they occur more than threshold number of times in the reference genome. Default value is 24.
-M < *int* > (`memory`)	Specify the Size of requested memory(RAM) in megabytes, 8000MB by default. With the default value, the index built for a mammalian genome (eg. human or mouse genome) will be saved into one block, enabling the fastest mapping speed to be achieved. The amount of memory used is $\sim$ 7600MB for mouse or human genome (other species have a much smaller memory footprint), when performing read mapping. Using less memory will increase read mapping time.
-c (`colorspace`)	Build a color-space index.
chr1.fa, chr2.fa, ... (`reference`)	Give names of chromosome files. Note that in Rsubread, only a single FASTA file including all reference sequences should be provided.

4.4 Read mapping

The `subread-align` program (`align` in Rsubread) extracts a number of subreads from each read and then uses these subreads to vote for the mapping location of the read. It uses the the "seed-and-vote" paradigm for read mapping. `subread-align` program automatically determines if a read should be globally aligned or locally aligned, making it particularly poweful for mapping RNA-seq reads. Table 2 describes the arguments used by the `subread-align` program (and also the `subjunc` program). These arguments are used by the read mapping programs included in both SourceForge `Subread` package and Bioconductor `Rsubread` package, although argument names are different in these two packages (arguments names used by Bioconductor `Rsubread` are included in parenthesis).

Table 2: arguments used by the `subread-align`/`subjunc` programs included in the SourceForge Subread package. Arguments in parenthesis in the first column are the equivalent arguments used in Bioconductor Rsubread package.

Arguments	Description
-i < *index* > (`index`)	Specify the base name of the index.
-r < *input* > (`readfile1`)	Give the name of an input file(FASTQ/FASTA format). For paired-end read data, this gives the first read file and the other read file should be provided via the -R option.
-R < *input* > (`readfile2`)	Provide the name of the second reads file from paired-end data. The program will then be switched to paired-end read mapping mode.
-o < *output* > (`output_file`)	Give the name of the output file (SAM format).
–SAMinput (`input_format`)	specify that the input read data are in SAM format.
–BAMinput (`input_format`)	specify that the input read data are in BAM format.
–BAMoutput (`output_format`)	specify that mapping results are saved into a BAM format file.
-n < *int* > (`nsubreads`)	Specify the number of subreads extracted from each read, 10 by default.
-m < *int* > (`TH1`)	Specify the consensus threshold, which is the minimal number of consensus subreads required for reporting a hit. The consensus subreads are those subreads which vote for the same location in the reference genome for the read. If pair-end read data are provided, at least one of the two reads from the same pair must satisfy this criteria. 3 by default.
-p < *int* > (`TH2`)	Specify the minimum number of consensus subreads both reads from the same pair must have. This argument is only applicable for paired-end read data. The value of this argument should not be greater than that of '-m' option, so as to rescue those read pairs in which one read has a high mapping quality but the other does not. 1 by default.
-d < *int* > (`minFragLength`)	Specify the minimum fragment/template length, 50 by default. Note that if the two reads from the same pair do not satisfy the fragment length criteria, they will be mapped individually as if they were single-end reads.
-D < *int* > (`maxFragLength`)	Specify the maximum fragment/template length, 600 by default.

-S $< ff : fr : rf >$ (PE_orientation)	Specify the orientation of the two reads from the same pair. It has three possible values including 'fr', 'ff' and ''rf. Letter 'f' denotes the forward strand and letter 'r' the reverse strand. 'fr' by default (ie. the first read in the pair is on the forward strand and the second read on the reverse strand).
-I $< int >$ (indels)	Specify the number of INDEL bases allowed in the mapping. 5 by default.
-u (unique)	Output the uniquely mapped reads only.
-Q (codetieBreakQS)	Use mapping quality scores to break ties when more than one best mapping location is found.
-H (tieBreakHamming)	Use Hamming distance to break ties when more than one best mapping location is found.
* -B $< int >$ (nBestLocations)	Specify the maximal number of equally-best mapping locations allowed to be reported for a read. Its value has to be within the range of 1 to 16. The default value is 1. The number of equally-best locations reported for a read will be less than or equal to the specified value. For example, if a read has two equally-best mapping locations, but the 'B' was set to 5, then only two locations will be reported for this read.
-P $< 3 : 6 >$ (phredOffset)	Specify the format of Phred scores used in the input data, '3' for phred+33 and '6' for phred+64. '3' by default. For `align` function in Rsubread, the possible values are '33' (for phred+33) and '64' (for phred+64). '33' by default.
-T $< int >$ (nthreads)	Specify the number of threads/CPUs used for mapping. 1 by default.
-b (color2base)	Output base-space reads instead of color-space reads in the mapping output. Note that the mapping itself will still be performed at color-space. This option is only applicable for color-space read mapping.
* -G $< int >$ (DP_GapOpenPenalty)	Specify the penalty for opening a gap when applying the Smith-Waterman dynamic programming to detecting indels. -2 by defaut.
* -E $< int >$ (DP_GapExtPenalty)	Specify the penalty for extending the gap when performing the Smith-Waterman dynamic programming. 0 by defaut.
* -X $< int >$ (DP_MismatchPenalty)	Specify the penalty for mismatches when performing the Smith-Waterman dynamic programming. 0 by defaut.
* -Y $< int >$ (DP_MatchScore)	Specify the score for the matched base when performing the Smith-Waterman dynamic programming. 2 by defaut.
–trim5 $< int >$ (nTrim5)	trim off $< int >$ number of bases from 5' end of each read. 0 by default.
–trim3 $< int >$ (nTrim3)	trim off $< int >$ number of bases from 3' end of each read. 0 by default.

–rg-id < *string* > (`readGroupID`)	specify the read group ID. If specified, the read group ID will be added to the read group header field and also to each read in the mapping output.
–rg < *string* > (`readGroup`)	add a < *tag* : *value* > to the read group (RG) header in the mapping output.
-v	Output version of the program.

* *arguments that are not supported by* ***subjunc****.*

4.5 Mapping quality scores

Both Subread and Subjunc aligners output a mapping quality score for each mapped read, defined by

$$\mathrm{MQS} = 100 + \frac{100}{l}\left\{\sum_{i \in b_m}(1-p_i) \; - \sum_{i \in b_{mm}}(1-p_i)\right\}$$

where l is the read length, p_i is the base-calling p-value for the ith base in the read, b_m is the set of locations of matched bases, and b_{mm} is the set of locations of mismatched bases.

Base-calling p values can be readily computed from the base quality scores. High quality bases have low base-calling p values. Read bases which were found to be insertions are treated as matched bases in the MQS calculation. The MQS is a read-length normalized value, which is in the range of 0 to 200. If a read can be best mapped to more than one location, its MQS will be divided by the number of such locations.

Chapter 5

Mapping reads generated by RNA sequencing technologies

5.1 A quick start for SourceForge Subread

An index must be built for the reference first and then the read mapping and/or junction detection can be carried out.

Step 1: Building an index

The following command can be used to build a base-space index. You can provide a list of FASTA files or a single FASTA file including all the reference sequences.

```
subread-buildindex -o my_index chr1.fa chr2.fa ...
```

For more details about index building, see Section 4.3.

Step 2: Aligning the reads

Subread

For the purpose of differential expression analysis (ie. discovering differentially expressed genes), we recommend you to use the Subread aligner. Subread carries out local alignments for RNA-seq reads. The commands used by Subread to align RNA-seq reads are the same as those used to align gDNA-seq reads. Below is an example of using Subread to map single-end RNA-seq reads.

```
subread-align -i my_index -r rnaseq-reads.txt -o subread_results.sam
```

Another RNA-seq aligner included in this package is the Subjunc aligner. Subjunc not only performs read alignments but also detects exon-exon junctions. The main difference between

Subread and Subjunc is that Subread does not attempt to detect exon-exon junctions in the RNA-seq reads. For the alignments of the exon-spanning reads, Subread just uses the largest mappable regions in the reads to find their mapping locations. This makes Subread more computationally efficient. The largest mappable regions can then be used to reliably assign the reads to their target genes by using a read summarization program (eg. featureCounts, see Section 6.2), and differential expression analysis can be readily performed based on the read counts yielded from read summarization. Therefore, Subread is sufficient for read mapping if the purpose of the RNA-seq analysis is to perform a differential expression analysis. Also, Subread could report more mapped reads than Subjunc. For example, the exon-spanning reads that are not aligned by Subjunc due to the lack of GT/AG splicing signals (this is the only donor/receptor site accepted by Subjunc) could be aligned by Subread, as long as they have a good match with the target region.

Subjunc

For other purposes of the RNA-seq data anlayses such as exon-exon junction detection and genomic mutation detection, in which reads need to be fully aligned (especially the exon-spanning reads), Subjunc aligner should be used. Below is an example command of using Subjunc to perform global alignments for paired-end RNA-seq reads. Note that there are two files included in the output: one containing the discovered exon-exon junctions (BED format) and the other containing the mapping results for reads (SAM or BAM format).

```
subjunc -i my_index -r rnaseq-reads1.txt -R rnaseq-reads2.txt -o subjunc_result
```

5.2 A quick start for Bioconductor Rsubread

An index must be built for the reference first and then the read mapping can be performed.

Step 1: Building an index

To build the index, you must provide a single FASTA file (eg. "genome.fa") which includes all the reference sequences.

```
library(Rsubread)
buildindex(basename="my_index",reference="genome.fa")
```

Step 2: Aligning the reads

Please refer to Section 5.1 for difference between Subread and Subjunc in mapping RNA-seq data. Below is an example for mapping a single-end RNA-seq dataset using Subread. Useful information about `align` function can be found in its help page (type `?align` in your R prompt).

```
align(index="my_index",readfile1="rnaseq-reads.txt",output_file="subread_results.sam")
```

Below is an example for mapping a single-end RNA-seq dataset using Subjunc. Useful information about `subjunc` function can be found in its help page (type `?subjunc` in your R prompt).

```
subjunc(index="my_index",readfile1="rnaseq-reads.txt",output_file="subjunc_results.sam")
```

5.3 Local read alignment

The `Subread` and `Subjunc` can both be used to map RNA-seq reads to the reference genome. If the goal of the RNA-seq data is to perform expression analysis, eg. finding genes expressing differentially between different conditions, then `Subread` is recommended. `Subread` performs fast local alignments for reads and reports the mapping locations that have the largest overlap with the reads. These reads can then be assigned to genes for expression analysis. For this type of analysis, global alignments for the exon-spanning reads are not required because local aligments are sufficient to get reads to be accurately assigned to genes.

However, for other types of RNA-seq data analyses such as exon-exon junction discovery, genomic mutation detection and allele-specific gene expression analysis, global alignments are required. The next section describes the Subjunc aligner, which performs global aligments for RNA-seq reads.

5.4 Global read alignment

Subjunc aligns each exon-spanning read by firstly using a large number of subreads extracted from the read to identify multiple target regions matching the selected subreads, and then using the splicing signals (donor and receptor sites) to precisely determine the mapping locations of the read bases. It also includes a verification step to compare the quality of mapping reads as exon-spanning reads with the quality of mapping reads as exonic reads to finally decide how to best map the reads. Reads may be re-aligned if required.

Output of Subjunc aligner includes a list of discovered exon-exon junction locations and also the complete alignment results for the reads. Table 2 describes the arguments used by the Subjunc program.

Chapter 6

Read summarization

6.1 Introduction

Sequencing reads often need to be assigned to genomic features of interest after they are mapped to the reference genome. This process is often called *read summarization* or *read quantification*. Read summarization is required by a number of downstream analyses such as gene expression analysis and histone modification analysis. The output of read summarization is a count table, in which the number of reads assigned to each feature in each library is recorded.

A particular challenge to the read summarization is how to deal with those reads that overlap more than one feature (eg. an exon) or meta-feature (eg. a gene). Care must be taken to ensure that such reads are not over-counted or under-counted. Here we describe the featureCounts program, an efficient and accurate read quantifier. featureCounts has the following features:

- It carries out precise and accurate read assignments by taking care of indels, junctions and fusions in the reads.
- It takes less than 4 minutes to summarize 20 million pairs of reads to 26k RefSeq genes using one thread, and uses <20MB of memory (you can run it on a Mac laptop).
- It supports multi-threaded running, making it extremely fast for summarizing large datasets.
- It supports GTF/SAF format annotation and SAM/BAM read data.
- It supports strand-specific read summarization.
- It can perform read summarization at both feature level (eg. exon level) and meta-feature level (eg. gene level).
- It allows users to specify whether reads overlapping with more than one feature should be counted or not.

- It gives users full control on the summarization of paired-end reads, including allowing them to check if both ends are mapped and/or if the fragment length falls within the specified range.
- It can discriminate the features that were overlapped by both ends of the fragment from the features that were overlapped by only one end of the same fragment to get more accurate read assignments.
- It allows users to specify whether chimeric fragments should be counted.

6.2 featureCounts

6.2.1 Input data

The data input to featureCounts consists of (i) one or more files of aligned reads in either SAM or BAM format and (ii) a list of genomic features in either Gene Transfer Format (GTF) or General Feature Format (GFF) or Simplified Annotation Format (SAF). Both the read alignment and the feature annotation should correspond to the same reference genome, which is a set of reference sequences representing chromosomes or contigs. For each read, the SAM file gives the name of the reference chromosome or contig to which the read mapped, the start position of the read on the chromosome or contig/scaffold, and the so-called CIGAR string giving the detailed alignment information including insertions and deletions and so on relative to the start position.

The genomic features can be specified in either GTF/GFF or SAF format. The SAF format is the simpler and includes only five required columns for each feature (see next section). In either format, the feature identifiers are assumed to be unique, in accordance with commonly used Gene Transfer Format (GTF) refinement of GFF.

featureCounts supports strand-specific read counting if strand-specific information is provided. Read mapping results usually include mapping quality scores for mapped reads. Users can optionally specify a minimum mapping quality score that the assigned reads must satisfy.

6.2.2 Annotation format

The genomic features can be specified in either GTF/GFF or SAF format. A definition of the GTF format can be found at UCSC website (`http://genome.ucsc.edu/FAQ/FAQformat.html#format4`). The SAF format includes five required columns for each feature: feature identifier, chromosome name, start position, end position and strand. These five columns provide the minimal sufficient information for read quantification purposes. Extra annotation data are allowed to be added from the sixth column.

A SAF-format annotation file should be a tab-delimited text file. It should also include a header line. An example of a SAF annotation is shown as below:

```
GeneID Chr Start End Strand
497097 chr1 3204563 3207049 -
```

```
497097 chr1 3411783 3411982 -
497097 chr1 3660633 3661579 -
100503874 chr1 3637390 3640590 -
100503874 chr1 3648928 3648985 -
100038431 chr1 3670236 3671869 -
...
```

`GeneID` column includes gene identifiers that can be numbers or character strings. Chromosomal names included in the `Chr` column must match the chromosomal names of reference sequences to which the reads were aligned.

6.2.3 Single and paired-end reads

Reads may be paired or unpaired. If paired reads are used, then each pair of reads defines a DNA or RNA fragment bookended by the two reads. In this case, featureCounts will count fragments rather than reads.

For paired reads, they are sometimes not adjacent to each other in the SAM/BAM file because for example they were sorted by chromosomal locations. However, featureCounts requires reads from the same pair to be adjacent to each other in the input. The featureCounts program provides an option, "-S" for featureCounts in SourceForge Subread package or "PEReadsReordering=TRUE" for featureCounts in Bioconductor Rsubread package, to sort reads by their names so that reads from the same pair can be placed to next to each other.

6.2.4 Features and meta-features

featureCounts is a general-purpose read summarization function, which assigns mapped reads (RNA-seq reads or genomic DNA-seq reads) to genomic features or meta-features. Each feature is an interval (range of positions) on one of the reference sequences. We define a meta-feature to be a set of features representing a biological construct of interest. For example, features often correspond to exons and meta-features to genes. Features sharing the same feature identifier in the GTF or SAF annotation are taken to belong to the same meta-feature. featureCounts can summarize reads at either the feature or meta-feature levels.

We recommend to use unique gene identifiers, such as NCBI Entrez gene identifiers, to cluster features into meta-features. Gene names are not recommended to use for this purpose because different genes may have the same names. Unique gene identifiers were often included in many publicly available GTF annotations which can be readily used for summarization. The Bioconductor Rsubread package also includes NCBI RefSeq annotations for human and mice. Entrez gene identifiers are used in these annotations.

6.2.5 Overlap of reads with features

featureCounts preforms precise read assignment by comparing mapping location of every base in the read or fragment with the genomic region spanned by each feature. It takes account of any gaps (insertions, deletions, exon-exon junctions or fusions) that are found in the read. It

calls a hit if any overlap (1bp or more) is found between the read or fragment and a feature. A hit is called for a meta-feature if the read or fragment overlaps any component feature of the meta-feature.

6.2.6 Multiple overlaps

A multi-overlap read or fragment is one that overlaps more than one feature, or more than one meta-feature when summarizing at the meta-feature level. featureCounts provides users with the option to either exclude multi-overlap reads or to count them for each feature that is overlapped. The decision whether or not to counting these reads is often determined by the experiment type. We recommend that reads or fragments overlapping more than one gene are not counted for RNA-seq experiments, because any single fragment must originate from only one of the target genes but the identity of the true target gene cannot be confidently determined. On the other hand, we recommend that multi-overlap reads or fragments are counted for most ChIP-seq experiments because epigenetic modifications inferred from these reads may regulate the biological functions of all their overlapping genes.

Note that, when counting at the meta-feature level, reads that overlap multiple features of the same meta-feature are always counted exactly once for that meta-feature, provided there is no overlap with any other meta-feature. For example, an exon-spanning read will be counted only once for the corresponding gene even if it overlaps with more than one exon.

6.2.7 Program usage

Table 3 describes the parameters used by the featureCounts program.

Table 3: arguments used by the featureCounts program included in the SourceForge Subread package. Arguments included in parenthesis are the equivalent parameters used by featureCounts function in Bioconductor Rsubread package.

Arguments	Description
input_files [-b if BAM] (`files, file.type="BAM" if BAM`)	Give the names of input read files that include the read mapping results. By default, input files should be in SAM format. `-b` option should be specified if BAM files are provided. Multiple files can be provided at the same time.
-a < *input* > (`annot.ext, annot.inbuilt`)	Give the name of an annotation file.
-o < *input* >	Give the name of the output file. The output file contains the number of reads assigned to each meta-feature (or each feature if `-f` is specified). Note that the featureCounts function in Rsubread does not use this parameter. It returns a `list` object including read summarization results and other data.
-A (`chrAliases`)	Give the name of a file that contains aliases of chromosome names. The file should be a comma delimited text file that includes two columns. The first column gives the chromosome names used in the annotation and the second column gives the chromosome names used by reads. This file should not contain header lines. Names included in this file are case sensitive.
-F (`isGTFAnnotationFile`)	Specify the format of the annotation file. Acceptable formats include 'GTF' and 'SAF' (see Section 6.2.2 for details). The C version of featureCounts program uses a GTF format annotation by default, but the R version uses a SAF format annotation by default. The R version also includes in-built annotations.
-t < *input* > (`GTF.featureType`)	Specify the feature type. Only rows which have the matched feature type in the provided GTF annotation file will be included for read counting. 'exon' by default.
-g < *input* > (`GTF.attrType`)	Specify the attribute type used to group features (eg. exons) into meta-features (eg. genes), when GTF annotation is provided. 'gene_id' by default. This attribute type is usually the gene identifier. This argument is useful for the meta-feature level summarization.
-b (`file.type`)	Indicate that the input read files are in BAM format.
-f (`useMetaFeatures`)	If specified, read summarization will be performed at feature level (eg. exon level). Otherwise, it is performed at meta-feature level (eg. gene level).

-O (`allowMultiOverlap`)	If specified, reads (or fragments if `-p` is specified) will be allowed to be assigned to more than one matched meta-feature (or feature if `-f` is specified). Reads/fragments overlapping with more than one meta-feature/feature will be counted more than once. Note that when performing meta-feature level summarization, a read (or fragment) will still be counted once if it overlaps with multiple features belonging to the same meta-feature but does not overlap with other meta-features.
-s $< int >$ (`isStrandSpecific`)	Indicate if strand-specific read counting should be performed. It has three possible values: 0 (unstranded), 1 (stranded) and 2 (reversely stranded). 0 by default. For paired-end reads, strand of the first read is taken as the strand of the whole fragment and FLAG field of the current read is used to tell if it is the first read in the fragment.
-M	If specified, multi-mapping reads/fragments will be counted (ie. a multi-mapping read will be counted up to N times if it has N reported mapping locations). The program uses the 'NH' tag to find multi-mapping reads.
-Q $< int >$ (`minMQS`)	The minimum mapping quality score a read must satisfy in order to be counted. For paired-end reads, at least one end should satisfy this criteria. 0 by default.
-T $< int >$ (`nthreads`)	Number of the threads. 1 by default.
-R	Output read counting result for each read/fragment. For each input read file, read counting results for reads/fragments will be saved to a tab-delimited file that contains four columns including read name, status(assigned or the reason if not assigned), name of target feature/meta-feature and number of hits if the read/fragment is counted multiple times. Name of the file is the same as name of the input read file except a suffix '.featureCounts' is added.
-p (`isPairedEnd`)	If specified, fragments (or templates) will be counted instead of reads. This option is only applicable for paired-end reads. The two reads from the same fragment must be adjacent to each other in the provided SAM/BAM file. If SAM/BAM input does not meet this requirement, the `-S` option should be provided as well.
-P (`checkFragLength`)	If specified, the fragment length will be checked when assigning fragments to meta-features or features. This option should be used together with `-p` (or `isPairedEnd` in Rsubread featureCounts). The fragment length thresholds should be specified using `-d` and `-D` options.

-d < *int* > (`minFragLength`)	Minimum fragment/template length, 50 by default.
-D < *int* > (`maxFragLength`)	Maximum fragment/template length, 600 by default.
-B (`requireBothEndsMapped`)	If specified, only fragments that have both ends successfully aligned will be considered for summarization. This option should be used together with `-p` (or `isPairedEnd` in Rsubread featureCounts).
-C (`countChimericFragments`)	If specified, the chimeric fragments (those fragments that have their two ends aligned to different chromosomes) will NOT be counted. This option should be used together with `-p` (or `isPairedEnd` in Rsubread featureCounts).
-S (`PEReadsReordering`)	If specified, the program will reorder input reads according to their names and make reads from the same pair be adjacent to each other. This option should be provided when reads from the same pair are not adjacent to each other in input SAM/BAM files (for instance sorting reads by chromosomal locations could decouple reads from the same pair).

6.3 A quick start for **featureCounts** in SourceForge **Subread**

You need to provide read mapping results (in either SAM or BAM format) and an annotation file for the read summarization. The example commands below assume your annotation file is in GTF format.

Summarizing single-end reads using 5 threads:

```
featureCounts -T 5 -a annotation.gtf -t exon -g gene_id
-o counts.txt mapping_results_SE.sam
```

Summarizing BAM format single-end read data:

```
featureCounts -b -a annotation.gtf -t exon -g gene_id
-o counts.txt mapping_results_SE.bam
```

Summarizing paired-end reads and counting fragments (instead of reads):

```
featureCounts -p -a annotation.gtf -t exon -g gene_id
-o counts.txt mapping_results_PE.sam
```

Summarizing a BAM input that included paired-end reads and the reads were sorted by their mapping locations (ie. reads from the same pair may not be adjacent to each other):

```
featureCounts -b -p -S -a annotation.gtf -t exon -g gene_id
-o counts.txt mapping_results_PE.bam
```

Counting fragments satisfying the fragment length criteria, eg. [50bp, 600bp]:

```
featureCounts -p -P -d 50 -D 600 -a annotation.gtf -t exon -g gene_id
-o counts.txt mapping_results_PE.sam
```

Counting fragments which have both ends successfully aligned without considering the fragment length constraint:

```
featureCounts -p -B -a annotation.gtf -t exon -g gene_id
-o counts.txt mapping_results_PE.sam
```

Excluding chimeric fragments from the fragment counting:

```
featureCounts -p -C -a annotation.gtf -t exon -g gene_id
-o counts.txt mapping_results_PE.sam
```

6.4 A quick start for featureCounts in Bioconductor Rsubread

You need to provide read mapping results (in either SAM or BAM format) and an annotation file for the read summarization. The example commands below assume your annotation file is in GTF format.

Load Rsubread library from you R session:

```
library(Rsubread)
```

Summarizing single-end reads using built-in RefSeq annotation for mouse genome mm9:

```
featureCounts(files="mapping_results_SE.sam",annot.inbuilt="mm9")
```

Summarizing single-end reads using a user-provided GTF annotation file:

```
featureCounts(files="mapping_results_SE.sam",annot.ext="annotation.gtf",
isGTFAnnotationFile=TRUE,GTF.featureType="exon",GTF.attrType="gene_id")
```

Summarizing single-end reads using 5 threads:

```
featureCounts(files="mapping_results_SE.sam",nthreads=5)
```

Summarizing BAM format single-end read data:

```
featureCounts(files="mapping_results_SE.bam",file.type="BAM")
```

Summarizing paired-end reads and counting fragments (instead of reads):

```
featureCounts(files="mapping_results_PE.sam",isPairedEnd=TRUE)
```

Summarizing a BAM input that included paired-end reads and the reads were sorted by their mapping locations (ie. reads from the same pair may not be adjacent to each other):

```
featureCounts(files="mapping_results_PE.bam",file.type="BAM",isPairedEnd=TRUE,PEReadsReordering=TRUE)
```

Counting fragments satisfying the fragment length criteria, eg. [50bp, 600bp]:

```
featureCounts(files="mapping_results_PE.sam",isPairedEnd=TRUE,checkFragLength=TRUE,
minFragLength=50,maxFragLength=600)
```

Counting fragments which have both ends successfully aligned without considering the fragment length constraint:

```
featureCounts(files="mapping_results_PE.sam",isPairedEnd=TRUE,requireBothEndsMapped=TRUE)
```

Excluding chimeric fragments from the fragment counting:

```
featureCounts(files="mapping_results_PE.sam",isPairedEnd=TRUE,countChimericFragments=FALSE)
```

Chapter 7

SNP calling

7.1 Algorithm

SNPs(Single Nucleotide Polymorphisms) are the mutations of single nucleotides in the genome. It has been reported that many diseases were initiated and/or driven by such mutations. Therefore, successful detection of SNPs is very useful in designing better diagnosis and treatments for a variety of diseases such as cancer. SNP detection also is an important subject of many population studies.

Next-gen sequencing technologies provide an unprecedented opportunity to identify SNPs at the highest resolution. However, it is extremely computing-intensive to analyze the data generated from these technologies for the purpose of SNP discovery because of the sheer volume of the data and the large number of chromosomal locations to be considered. To discover SNPs, reads need to be mapped to the reference genome first and then all the read data mapped to a particular site will be used for SNP calling for that site. Discovery of SNPs is often confounded by many sources of errors. Mapping errors and sequencing errors are often the major sources of errors causing incorrect SNP calling. Incorrect alignments of indels, exon-exon junctions and fusions in the reads can also result in wrong placement of blocks of continuous read bases, likely giving rise to consecutive incorrectly reported SNPs.

We have developed a highly accurate and efficient SNP caller, called *exactSNP* [8]. *exactSNP* calls SNPs for individual samples, without requiring control samples to be provided. It tests the statistical significance of SNPs by comparing SNP signals to their background noises. It has been found to be an order of magnitude faster than existing SNP callers.

7.2 exactSNP

Below is the command for running `exactSNP` program. The complete list of parameters used by `exactSNP` can be found in Table 4.

```
exactSNP [options] -i input -g reference_genome -o output
```

Table 4: arguments used by the `exactSNP` program included in the SourceForge Subread package. Arguments included in parenthesis are the equivalent parameters used by `exactSNP` function in Bioconductor Rsubread package.

Arguments	Description
-i $< file >$ $[-b\ if\ BAM]$ ($readFile$)	Specify name of an input file including read mapping results. The format of input file can be SAM or BAM (`-b` needs to be specified if a BAM file is provided).
-b (isBAM)	Indicate the input file provided via $-i$ is in BAM format.
-g $< file >$ (refGenomeFile)	Specify name of the file including all reference sequences. Only one single FASTA format file should be provided.
-o $< file >$ (outputFile)	Specify name of the output file. This program outputs a VCF format file that includes discovered SNPs.
-Q $< int >$ (qvalueCutoff)	Specify the q-value cutoff for SNP calling at sequencing depth of 50X. 12 by default. The corresponding p-value cutoff is 10^{-Q}. Note that this program automatically adjusts the q-value cutoff according to the sequencing depth at each chromosomal location.
-f $< float >$ (minAllelicFraction)	Specify the minimum fraction of mis-matched bases a SNP-containing location must have. Its value must between 0 and 1. 0 by default.
-n $< int >$ (minAllelicBases)	Specify the minimum number of mis-matched bases a SNP-containing location must have. 1 by default.
-r $< int >$ (minReads)	Specify the minimum number of mapped reads a SNP-containing location must have (ie. the minimum coverage). 1 by default.
-x $< int >$ (maxReads)	Specify the maximum number of mapped reads a SNP-containing location could have. 3000 by default. Any location having more than the threshold number of reads will not be considered for SNP calling. This option is useful for removing PCR artefacts.
-s $< int >$ (minBaseQuality)	Specify the cutoff for base calling quality scores (Phred scores) read bases must satisfy to be used for SNP calling. 13 by default. Read bases that have Phred scores lower than the cutoff value will be excluded from the analysis.
-t $< int >$ (nTrimmedBases)	Specify the number of bases trimmed off from each end of the read. 3 by default.
-T $< int >$ (nthreads)	Specify the number of threads. 1 by default.

Chapter 8

Case studies

8.1 A Bioconductor R pipeline for analyzing RNA-seq data

Here we illustrate how to use two Bioconductor packages - Rsubread and limma - to perform a complete RNA-seq analysis, including Subread read mapping, featureCounts read summarization, voom normalization and limma differential expresssion analysis.

Data and software. The RNA-seq data used in this case study include four libraries: A_1, A_2, B_1 and B_2. A_1 and A_2 are both Universal Human Reference RNA (UHRR) samples but they underwent separate sample preparation. B_1 and B_2 are both Human Brain Reference RNA (HBRR) samples and they also underwent separate sample preparation. Note that these libraries only included reads originating from human chromosome 1 (according to Subread aligner). These read data were generated by the SEQC Consortium. We have put into a tar ball these read data and the reference sequence data of chromosome 1 from human genome build GRCh37/hg19, and it can be downloaded from
`http://bioinf.wehi.edu.au/RNAseqCaseStudy/data.tar.gz` (283MB).

After downloading the dataset, uncompress it and save it to your current working directory. Launch R and load Rsubread and limma libraries by issuing the following commands at your R prompt. Version of your R should be 3.0.2 or later. Rsubread version should be 1.12.0 or later and limma version should be 3.18.0 or later. Note that Rsubread only supports Linux/Unix and Mac OS X.

```
library(Rsubread)
library(limma)
```

To install/update Rsubread and limma packages, issue the following commands at your R prompt:

```
source("http://bioconductor.org/biocLite.R")
biocLite(pkgs=c("Rsubread","limma"))
```

Index building. Build an index for human chromosome 1. This will take ∼3 minutes. Index files with basename 'chr1' will be generated in your current working directory.

```
buildindex(basename="chr1",reference="hg19_chr1.fa")
```

Alignment. Perform read alignment for all four libraries and report uniquely mapped reads only. This will take ∼4 minutes. SAM files which include the mapping results will be generated in your current working directory.

```
for(i in c("A_1","A_2","B_1","B_2"))
  align(index="chr1",readfile1=paste(i,"txt",sep="."),output_file=paste(i,"sam",sep="."),
  unique=TRUE,indels=5)
```

Read summarization. Summarize mapped reads to RefSeq genes. This will take less than half a minute. Note that the featureCounts function contains built-in RefSeq annotations. featureCounts returns an R 'List' object that includes a read count table and annotation data. The read count table can be directly fed into limma for normalization and differential expresssion analysis.

```
counts <- featureCounts(files=c("A_1.sam","A_2.sam","B_1.sam","B_2.sam"),annot.inbuilt="hg19")
```

Filtering. Calculate RPKM (reads per kilobases of exon per million reads mapped) values for genes and use these values to filter out those genes which failed to achieve a 0.5 RPKM in at least two libraries.

```
counts_rpkm <- apply(counts$counts,2,function(x) x*(1000/counts$annotation$Length)*(1e6/sum(x)))
isexpr <- rowSums(counts_rpkm >= 0.5) >= 2
x <- counts$counts[isexpr,]
```

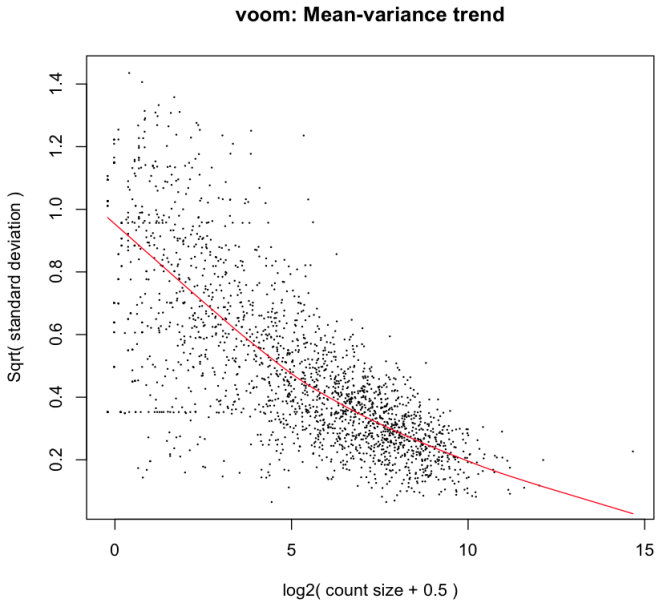
Design matrix. The following analyses are very similar to the analyses performed for microarray expression data. Firstly, we create a design matrix:

```
celltype <- factor(c("A","A","B","B"))
design <- model.matrix(~0+celltype)
colnames(design) <- levels(celltype)
```

Normalization. Then we perform voom normalization:

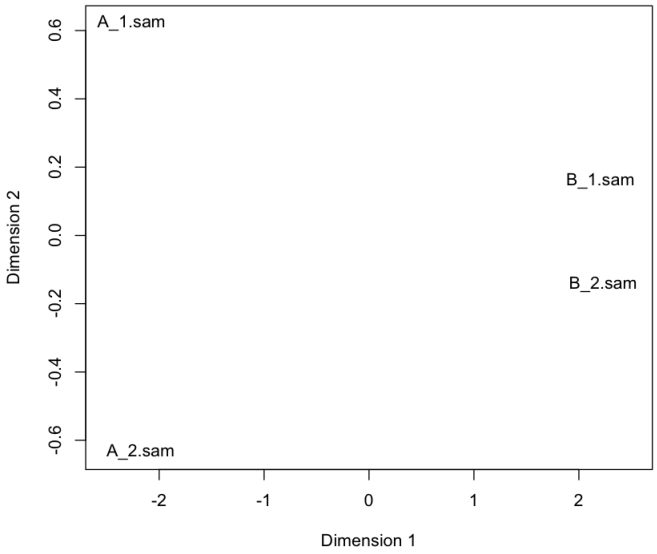
```
y <- voom(x,design,plot=TRUE)
```

The figure below shows the mean-variance relationship estimated by voom for the data.



Sample clustering. The following multi-dimensional scaling plot shows that sample A libraries are clearly separated from sample B libraries.

```
plotMDS(y,xlim=c(-2.5,2.5))
```



Linear model fitting and differential expression analysis. Fit linear models to genes and assess differential expression using the eBayes moderated t statistic. Here we compare A vs B. 556 and 983 genes were found down- and up-regulated in sample A compared to sample B, respectively.

```
fit <- lmFit(y,design)
contr <- makeContrasts(AvsB=A-B,levels=design)
```

```
fit.contr <- eBayes(contrasts.fit(fit,contr))
dt <- decideTests(fit.contr)
summary(dt)
   AvsB
-1  556
0   514
1   983
```

List top 10 differentially expressed genes:

```
options(digits=3)
topTable(fit.contr)
            ID logFC AveExpr     t  P.Value adj.P.Val    B
202       2752 -2.39    12.9 -89.7 1.99e-19  2.54e-16 35.1
1907 100131754 -1.63    16.0 -88.2 2.47e-19  2.54e-16 33.3
772      22883 -2.24    12.5 -69.6 5.17e-18  3.13e-15 31.9
417       6135  2.24    12.1  68.2 6.73e-18  3.13e-15 31.6
321       4904  3.00    11.5  67.6 7.62e-18  3.13e-15 31.4
420       6202  2.40    12.1  65.2 1.20e-17  4.11e-15 31.0
146       2023  2.72    13.4  63.8 1.61e-17  4.72e-15 30.5
794      23154 -3.73    11.4 -58.3 5.10e-17  1.31e-14 29.3
416       6125  2.01    11.8  50.2 3.47e-16  7.83e-14 27.7
565       8682 -2.59    11.7 -49.8 3.82e-16  7.83e-14 27.6
```

Bibliography

[1] Y. Liao, G. K. Smyth, and W. Shi. The subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic Acids Research*, 41:e108, 2013.

[2] K. W. Tang, B. Alaei-Mahabadi, T. Samuelsson, M. Lindh, and E. Larsson. The landscape of viral expression and host gene fusion and adaptation in human cancer. *Nature Communications.*, 2013 Oct 1;4:2513. doi: 10.1038/ncomms3513, 2013.

[3] K. Man, M. Miasari, W. Shi, A. Xin, D. C. Henstridge, S. Preston, M. Pellegrini, G. T. Belz, G. K. Smyth, M. A. Febbraio, S. L. Nutt, and A. Kallies. The transcription factor IRF4 is essential for TCR affinity-mediated metabolic programming and clonal expansion of T cells. *Nature Immunology*, 2013 Sep 22. doi: 10.1038/ni.2710, 2013.

[4] L. Spangenberg, P. Shigunov, A. P. Abud, A. R. Cofr, M. A. Stimamiglio, C. Kuligovski, J. Zych, A. V. Schittini, A. D. Costa, C. K. Rebelatto, P. R. Brofman, S. Goldenberg, A. Correa, H. Naya, and B. Dallagiovanna. Polysome profiling shows extensive posttranscriptional regulation during human adipocyte stem cell differentiation into adipocytes. *Stem Cell Research*, 11:902–12, 2013.

[5] J. Z. Tang, C. L. Carmichael, W. Shi, D. Metcalf, A. P. Ng, C. D. Hyland, N. A. Jenkins, N. G. Copeland, V. M. Howell, Z. J. Zhao, G. K. Smyth, B. T. Kile, and W. S. Alexander. Transposon mutagenesis reveals cooperation of ETS family transcription factors with signaling pathways in erythro-megakaryocytic leukemia. *Proc Natl Acad Sci U S A*, 110:6091–6, 2013.

[6] B. Pal, T. Bouras, W Shi, F. Vaillant, J. M. Sheridan, N. Fu, K. Breslin, K. Jiang, M. E. Ritchie, M. Young, G. J. Lindeman, G. K. Smyth, and J. E. Visvader. Global changes in the mammary epigenome are induced by hormonal cues and coordinated by Ezh2. *Cell Reports*, 3:411–26, 2013.

[7] Y. Liao, G. K. Smyth, and W. Shi. featureCounts: an efficient general-purpose read summarization program. *arXiv:1305.3347*, 2013.

[8] Y. Liao, G. K. Smyth, and W. Shi. ExactSNP: an efficient and accurate SNP calling algorithm. *In preparation.*