

RNA-seq Data Analysis

Leila Taher

WS 2022/2023

1 The molecular basis of flowering

You will be using data from an article by Jiao *et al.*: “Cell-type specific analysis of translating RNAs in developing flowers reveals new levels of control”, *Mol Syst Biol*, 2010 (<https://www.ncbi.nlm.nih.gov/pubmed/20924354>).

Flower development in *Arabidopsis thaliana* has been studied extensively and is one of the best-understood aspects of plant development. However, the genes that are expressed within each domain of the floral meristem or subsequently formed organ primordia at each morphologically defined stage during flower development are still a matter of debate. To describe the expression profiles of cells within spatiotemporal domains of the *Arabidopsis thaliana* flower Jiao et al performed RNA-seq. In addition, Jiao et al compared the translome – the translating transcriptome – and the transcriptome to study the points of regulation during flower development. Thus, RNA samples were separated into:

- the transcriptome (AP3); and
- the translome (TRL).

Create a plain text file describing the FASTQ files:

FASTQFile	Sample	Condition
SRR064154.fastq	AP3_f14a	AP3
SRR064155.fastq	AP3_f14b	AP3
SRR064166.fastq	T1_f14a	TRL
SRR064167.fastq	T1_f14b	TRL

Save it under the name “targets.txt”. This is the only file in this analysis workflow that you will need to generate manually, e.g., using a text editor.

2 Genome annotation

definition 2.1 (Genome Annotation). **Genome annotation** is the process of attaching biological information to sequences.

A common format for storing annotation, and in particular, information about gene structure, is the General Feature Format (GFF). A GFF file is a simple tab-delimited text file for describing genomic *features*.

definition 2.2 (Feature). A **feature** is an interval (i.e., a range of positions) on a chromosome or a union of such intervals.

There are several slightly but significantly different GFF file formats, including GTF (General Transfer Format). The GFF format consists of one line per feature, each containing nine fields of data, separated by tabs, plus optional track definition lines:

- All but the final field in each feature line must contain a value; “empty” fields are denoted with a ‘.’:
 - seqname: the name of the chromosome or scaffold;

- source: the data source (database or project name or name of the program that generated this feature);
- feature: the feature type name, e.g. Gene, Variation, Similarity.
- start: the start position of the feature, with sequence numbering starting at 1.
- end: the end position of the feature, with sequence numbering starting at 1.
- score: a floating point value.
- strand: + (forward) or - (reverse).
- frame: '0', '1' or '2'. '0' indicates that the first base of the feature is the first base of a codon, '1' that the second base is the first base of a codon, and so on.
- attribute: a semicolon-separated list of tag-value pairs, providing additional information about each feature.

```

#!genome-build TAIR10
#!genome-version TAIR10
#!genome-date 2008-04
#!genome-build-accession GCA_000001735.1
#!genebuild-last-updated 2010-09
1      araport11      gene      3631      5899      .      +      .      gene_id "AT1G01010"
; gene_name "NAC001"; gene_source "araport11"; gene_biotype "protein_coding";
1      araport11      transcript      3631      5899      .      +      .      gene_id "AT1G01010"
"; transcript_id "AT1G01010.1"; gene_name "NAC001"; gene_source "araport11";
gene_biotype "protein_coding"; transcript_source "araport11"; transcript_biotype "
protein_coding";

```

This is the annotation of the *A. thaliana* genome (TAIR10, release 43). The file was downloaded from the <https://plants.ensembl.org/info/data/ftp/index.html> Ensembl Plants database.

3 Mapping RNA-seq reads with HISAT2

1. To efficiently map the reads, the reference genome must first be indexed. Indexing takes relatively long, but only needs to be performed once. You will be working with chromosome 1. Enter the following command in the Linux terminal:

```

hisat2-build Arabidopsis_thaliana.TAIR10.dna.chromosome.1.fa
Arabidopsis_thaliana.TAIR10.dna.chromosome.1.fa

```

- hisat2-build is the tool that will index your reference genome (specifically, chromosome 1).
 - The first argument is the name of the FASTA-formatted file containing the reference genome.
 - The second argument provides the prefix that you want to give the index files.
 - Note that you may need to include the complete paths to the tool and input files.
 - What are the names and paths of the output files generated by hisat2-build?
2. Once the genome has been indexed, you can map the reads in each FASTQ file with HISAT2. HISAT2 is a program that aligns RNA-seq reads rapidly to a genome.

Run HISAT2 with the following parameters:

- -q: Defines the input file as fastq files.
- -x: The indexed genome prefix.
- -U: The single-end input read file in fastq format.
- -S: The output file with .sam extension.

Create a folder called “results” in the same folder in which you have stored the data. This is where you should store the mappings.

The following bash script will run HISAT2 on all the .fastq files in the working directory:

```
files='SRR064154.fastq SRR064155.fastq SRR064166.fastq SRR064167.fastq '
for input in $files
do
    ./hisat2 -q -x Arabidopsis_thaliana.TAIR10.dna.chromosome.1.fa
    -U $input -S results/$input.sam"
done
```

- Note that you may need to include the complete paths to the tool and input files and possibly modify the output variable.
- What are the names and paths of the output files generated by HISAT2?

For follow-up steps, output .sam files need to be transformed into .bam files using samtools. You can use the following code for that:

```
files='SRR064154.fastq SRR064155.fastq SRR064166.fastq SRR064167.fastq '
for input in $files
do
    samtools view -bS results/$input.sam" > results/$input.bam"
done
```

4 Counting the reads overlapping with certain genomic features

Given a file with mapped sequencing reads and a list of genomic features, a common task is to count how many reads map to each feature. In the case of RNA-Seq, the features are typically genes or the union of their exons. One may also consider each exon as a feature and quantify, for example, alternative splicing.

4.1 Importing annotation data into R

The `rtracklayer` R package includes several useful functions to import/export `GRanges` objects from and to different data formats commonly used in genomic analyses.

`GRanges` objects define integer ranges (`IRanges`) with two more important pieces of information:

- the chromosome or scaffold *name* (`seqnames`), and
- the DNA strand.

The DNA strand can be specified as plus “+” or minus “-”, or left unspecified with “*”.

Thus `GRanges` are objects representing genomic intervals.

```
library("rtracklayer")
gtf <- import("Arabidopsis_thaliana.TAIR10.48.gtf.gz")
class(gtf)

## [1] "GRanges"
## attr(,"package")
## [1] "GenomicRanges"

head(gtf, 2)

## GRanges object with 2 ranges and 14 metadata columns:
##      seqnames      ranges strand |      source      type      score      phase
##      <Rle> <IRanges> <Rle> | <factor>    <factor> <numeric> <integer>
##   [1]          1 3631-5899      + | araport11     gene      <NA>      <NA>
##   [2]          1 3631-5899      + | araport11 transcript <NA>      <NA>
##      gene_id  gene_name gene_source  gene_biotype transcript_id
##      <character> <character> <character>    <character>    <character>
```

```
## [1] AT1G01010 NAC001 araport11 protein_coding <NA>
## [2] AT1G01010 NAC001 araport11 protein_coding AT1G01010.1
## transcript_source transcript_biotype exon_number exon_id protein_id
## <character> <character> <character> <character> <character>
## [1] <NA> <NA> <NA> <NA> <NA>
## [2] araport11 protein_coding <NA> <NA> <NA>
## -----
## seqinfo: 7 sequences from an unspecified genome; no seqlengths
```

Select only the protein-coding gene features:

```
# Exclude NA values
idx <- !is.na(mcols(gtf)$type) & !is.na(mcols(gtf)$gene_biotype)
genes_gtf <- gtf[idx]

idx <- mcols(genes_gtf)$type == "gene" & mcols(genes_gtf)$gene_biotype == "protein_coding"
genes_gtf <- genes_gtf[idx]
```

Finally, split the GRanges object by gene identifier to produce a GRangesList object. Use this object to compute the number of reads in each sequencing library overlapping each gene in the *Arabidopsis thaliana* genome:

```
genes_gtf <- split(genes_gtf, mcols(genes_gtf)$gene_id)
genes_gtf

## GRangesList object of length 27628:
## $AT1G01010
## GRanges object with 1 range and 14 metadata columns:
## seqnames ranges strand | source type score phase
## <Rle> <IRanges> <Rle> | <factor> <factor> <numeric> <integer>
## [1] 1 3631-5899 + | araport11 gene <NA> <NA>
## gene_id gene_name gene_source gene_biotype transcript_id
## <character> <character> <character> <character> <character>
## [1] AT1G01010 NAC001 araport11 protein_coding <NA>
## transcript_source transcript_biotype exon_number exon_id protein_id
## <character> <character> <character> <character> <character>
## [1] <NA> <NA> <NA> <NA> <NA>
##
## $AT1G01020
## GRanges object with 1 range and 14 metadata columns:
## seqnames ranges strand | source type score phase gene_id
## [1] 1 6788-9130 - | araport11 gene <NA> <NA> AT1G01020
## gene_name gene_source gene_biotype transcript_id transcript_source
## [1] ARV1 araport11 protein_coding <NA> <NA>
## transcript_biotype exon_number exon_id protein_id
## [1] <NA> <NA> <NA> <NA>
##
## $AT1G01030
## GRanges object with 1 range and 14 metadata columns:
## seqnames ranges strand | source type score phase gene_id
## [1] 1 11649-13714 - | araport11 gene <NA> <NA> AT1G01030
## gene_name gene_source gene_biotype transcript_id transcript_source
## [1] NGA3 araport11 protein_coding <NA> <NA>
```

```
##      transcript_biotype exon_number exon_id protein_id
## [1]                <NA>        <NA>    <NA>        <NA>
##
## ...
## <27625 more elements>
## -----
## seqinfo: 7 sequences from an unspecified genome; no seqlengths
```

4.2 Counting reads with summarizeOverlaps

Read the “targets.txt” file into R:

```
targets <- read.delim("targets.txt")
targets

##      FASTQFile      Sample Condition
## 1 SRR064154.fastq AP3_fl4a        AP3
## 2 SRR064155.fastq AP3_fl4b        AP3
## 3 SRR064166.fastq Tl_fl4a         TRL
## 4 SRR064167.fastq Tl_fl4b         TRL
```

To count the reads, you will use the `summarizeoverlaps()` function GenomicAlignments R package. The mapped reads must be passed as arguments of the `summarizeoverlaps()` function. This can be done in several ways, including as a `BamFileList` object.

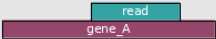
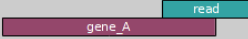


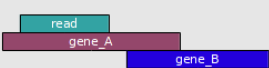
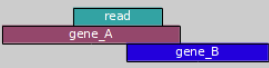
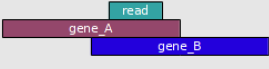

```
library("Rsamtools")
samples <- as.character(targets$FASTQFile)
samplespath <- paste("./results/", samples, ".bam", sep="")
names(samplespath) <- samples
bfl <- BamFileList(samplespath, yieldSize=50000, index=character())
```

The Rsamtools R package is an interface between R and Samtools.

Now, you can count the reads overlapping with genomic features in `genes_gtf`:

```
library("GenomicAlignments")
countDF <- summarizeOverlaps(genes_gtf, bfl, mode="Union", ignore.strand=TRUE)
countDF
```

- "mode" can be one of the pre-defined count methods such as “Union”, “IntersectionStrict”, or “IntersectionNotEmpty” or it a user supplied count function.
 - "Union": (Default) Reads that overlap any portion of exactly one feature are counted. Reads that overlap multiple features are discarded.
 - "IntersectionStrict": A read must fall completely within the feature to be counted. If a read overlaps multiple features but falls within only one, the read is counted for that feature. If the read is within multiple features, the read is discarded.
 - "IntersectionNotEmpty": A read must fall in a unique disjoint region of a feature to be counted. When a read overlaps multiple features, the features are partitioned into disjoint intervals. Regions that are shared between the features are discarded leaving only the unique disjoint regions. If the read overlaps one of these remaining regions, it is assigned to the feature the unique disjoint region came from.

	union	intersection_strict	intersection_nonempty
	gene_A	gene_A	gene_A
	gene_A	no_feature	gene_A
	gene_A	no_feature	gene_A
	gene_A	gene_A	gene_A
	gene_A	gene_A	gene_A
	ambiguous (both genes with --nonunique all)	gene_A	gene_A
	ambiguous (both genes with --nonunique all)		
	alignment_not_unique (both genes with --nonunique all)		

`summarizeOverlaps()` returns a `RangedSummarizedExperiment` object. The assays slot holds the counts:

```
countDF <- assays(countDF)$counts
head(countDF, n=3)
```

- Save the read counts as a table.

5 Testing for differential expression with the DESeq2 R package

Differential expression analysis is used to identify differences in the transcriptome across a cohort of samples. There are many tools available to perform this type of analysis. DESeq2 is a popular differential expression analysis R package. Its differential expression tests are based on a negative binomial generalized linear model.

```
library("DESeq2")
```

Input data for DESeq2 consists of (non-normalized) read counts at either the gene or transcript level.

5.1 Experimental design

First, you need to specify which factor(s) in the metadata are relevant for the experimental design and how they should be used in the analysis. The formula it should take the form of a "~" followed by "+" separating factors. Let us assume that you are interested in evaluating differences between the transcriptome and the transcriptome: ~ Condition

Note that the control or reference sample group for a given factor should appear first when viewing the levels of the corresponding variable. Here, you have only one factor of interest ("Condition") and two sample groups ("AP3" and "TRL"), and you will be using "AP3" as control:

```
levels(targets$Condition)
## [1] "AP3" "TRL"
```

- If you are curious about more complex experimental designs, please refer to the section “Multi-factor designs” in the DESeq2 vignette.

Now you have everything you need to perform a differential expression analysis, namely:

1. a table with the counts for each gene and each RNA-seq library;
2. a table with metadata describing the samples; and
3. an experimental design formula.

The next step is to create an `DESeqDataSet` object, which will store the read counts and intermediate calculations needed for the differential expression analysis. The object will also store the design formula used to estimate the *dispersion* and the fold-changes used to identify differentially expressed genes. For more details refer to PMID:24349066 and PMID:22287627.

```
dds <- DESeqDataSetFromMatrix(countData=countDF, colData=targets, design=~Condition)
# dds
```

Note that DESeq2 assumes that the samples in the count table (columns) are in the same order as in the table with the descriptions (rows).

5.2 Running the DESeq2 pipeline

The next step is to run the `DESeq()` function on our DESeq2 data set object. In this step the algorithm will perform the following:

1. estimate the size factors;
2. estimate the dispersion; and
3. fit a Negative Binomial generalized linear model for each gene and compute the Wald statistic.

```
dds <- DESeq(dds)
```

Indeed, `DESeq()` is a wrapper for three functions, which can also be executed separately:

```
dds <- estimateSizeFactors(dds)
dds <- estimateDispersions(dds)
dds <- nbinomWaldTest(dds)
```

1. `estimateSizeFactors` estimates the scaling factors used as “offsets” by the statistical model to make the different libraries comparable.
2. `estimateDispersions` obtains dispersion estimates for Negative Binomial distributed data.
3. `nbinomWaldTest` computes a P-value for the null hypothesis that the \log_2 fold change relative to the control experiment is zero using previously calculated scaling factors and dispersion estimates.

DESeq2 computes a *P*-value for each gene:

- The null hypothesis H_0 states that “there is no effect of the treatment on the gene and that the observed difference between treatment and control is merely caused by experimental variability”.
- The alternative hypothesis H_1 is that the gene is **differentially expressed** between the treatment and the control.

If the *P*-value is small than a given cut-off α (e.g., 0.05), you should reject H_0 in favor of H_1 and conclude that the gene is differentially expressed.

5.3 Inspecting the results

The results for the **last variable in the design formula** can be extracted using the `results()` function. In our case, there is only one, *Condition*:

```
res <- results(dds)
```

`res` is a `data.frame` with several columns:

```
mcols(res)
```

- `baseMean`, is the average of the normalized count values, taken over all samples. The remaining columns refer to a specific *contrast*, i.e., the comparison of the levels *DPN* versus *Control* of the factor variable *treatment*.
- The column `log2FoldChange` is the fold-change. Given two numbers, *A* and *B*, the fold change from *A* to *B* is

$$\frac{B}{A}.$$

It tells us how much the gene's expression has changed comparison to the control (in this case, AP3):

$$\frac{\text{TRL}}{\text{AP3}}.$$

The value is reported on a logarithmic scale:

$$\log_2 \frac{\text{TRL}}{\text{AP3}}.$$

- `lfcSE` is the standard error estimate for the \log_2 fold change estimate.
- `stat` is the Wald statistic: the `log2FoldChange` divided by `lfcSE`.
- `pvalue` is the *P*-value of the Wald test.
- `padj` is the *P*-value adjusted for multiple testing, to reflect the fact that you perform many statistical tests (one for each gene!) on the same dataset. The adjustment is done using Benjamini and Hochberg's method to control the false discovery rate (FDR). The FDR is the fraction of tests that are expected to result in rejecting the null hypothesis when the null hypothesis is actually true.

```
res
```

- For more information, visit:
 - <http://bioconductor.org/packages/release/bioc/vignettes/DESeq2/inst/doc/DESeq2.html>, and
 - <https://bioconductor.org/packages/devel/bioc/manuals/DESeq2/man/DESeq2.pdf>.