1 RNA-Seq Expression Analysis

1.1 Introduction

RNA sequencing (RNA-Seq) is a high-throughput method used to profile the transcriptome, quantify gene expression and discover novel RNA molecules or splice variants. This tutorial uses RNA sequencing of human cancer samples to walk you through transcriptome alignment, visualisation, simple quality control checks and shows you how to profile transcriptomic differences by identifying differentially expressed genes.

For an introduction to RNA-Seq principles and best practices see:

> **A survey of best practices for RNA-Seq data analysis**
> Ana Conesa, Pedro Madrigal, Sonia Tarazona, David Gomez-Cabrero, Alejandra Cervera, Andrew McPherson, Michał Wojciech Szcześniak, Daniel J. Gaffney, Laura L. Elo, Xuegong Zhang and Ali Mortazavi
> *Genome Biol. 2016 Jan 26;17:13 doi:10.1186/s13059-016-0881-8*

1.2 Learning Outcomes

By the end of this tutorial you can expect to be able to:

- Align RNA-Seq reads to a reference genome and a transcriptome
- Visualise transcription data using standard tools
- Perform QC of NGS transcriptomic data
- Quantify the expression values of your transcripts using standard tools

1.3 Practical Outline

This tutorial comprises the following sections

1. Introducing the tutorial dataset
2. Mapping RNA-Seq reads to the genome with HISAT2
3. Visualising transcriptomes with IGV
4. Transcript quantification with Kallisto
5. Identifying differentially expressed genes with Sleuth
6. Key aspects of differential expression analysis

1.4 Authors

This tutorial was developed by Victoria Offord and Adam Reid and adapted for use here by Nyasha Chambwe.

1.5 Prerequisites

This tutorial assumes that you have the following software or packages and their dependencies installed on your computer. The software or packages used in this tutorial may be updated from time to time, so we have given you the version which was used when writing the tutorial

Package	Link for download/installation instructions	Version tested
HISAT2	https://ccb.jhu.edu/software/hisat2/index.shtml	2.1.0
samtools	https://github.com/samtools/samtools	1.10
IGV	https://software.broadinstitute.org/software/igv/	2.7.2
kallisto	https://pachterlab.github.io/kallisto/download	0.46.2
R	https://www.r-project.org/	4.0.2
sleuth	https://pachterlab.github.io/sleuth/download	0.30.0
bedtools	http://bedtools.readthedocs.io/en/latest/content/installation.html	2.29.2

1.6 Where can I find the tutorial data

You can find the data for this tutorial by typing the following command in a new terminal window:

```
[ ]: cd /home/manager/course_data/rna_seq_human
```

Within the module directory, create a directory to write the outputs you will generate during this practical using the following command:

```
[ ]: mkdir -p outputs
```

1.6.1 Download Suppplemental Datasets

Internet Access Required

We will need to download two additional annotation files for this tutorial. Follow these instructions:

Use `wget` to download supplemental data files

wget https://www.dropbox.com/s/nfuea7ik6wlidum/hsapiens_chr21_transcript_to_gene.csv?dl=1 -O data/hsapiens_chr21_transcript_to_gene.csv

wget https://www.dropbox.com/s/8xt8q1o0aej1ry1/hsapiens_chr21_transcripts.fa?dl=1 -O data/hsapiens_chr21_transcripts.fa

Confirm that you have the following files in the data folder:

- hsapiens_chr21_transcript_to_gene.csv
- hsapiens_chr21_transcripts.fa

```
[ ]: ls -lhr data/hsapiens_chr21_transcript*
```

You are ready to go.

Now, let's head to the first section of this tutorial which will be introducing the tutorial dataset.

2 Introducing the tutorial dataset

Prostate cancer incidence and the risk of dying from the disease is higher in **African American (AA)** men compared to **European American (EA)** men (American Cancer Association for Cancer Research AACR Cancer Disparities Report 2020). These disparities are thought to be caused by a complex interplay between environmental and lifestyle factors as well as biological factors related to ancestry genetics. The molecular basis for these disparities is not well understood.

Working through this tutorial, you will analyse RNA-sequencing data from high grade prostate cancer and matched noncancer adjacent tissue samples from a cohort of ethnically diverse men (AAs and EAs) to investigate the molecular differences between these groups.

The dataset you will be using for this tutorial have been taken from the following publication:

> **Exogenous IL-6 induces mRNA splice variant MBD2_v2 to promote stemness in TP53 wild-type, African American PCa cells.** Teslow, E. A., Bao, B., Dyson, G., Legendre, C., Mitrea, C., Sakr, W., Carpten, J. D., Powell, I., & Bollig-Fischer, A. (2018). Molecular oncology, 12(7), 1138–1152. https://doi.org/10.1002/1878-0261.12316

This published dataset is publicly available from the Gene Expression Omnibus Database (Accession Number: GSE104131) and has been modified and adapted for use in these education materials.

2.1 Research Question

This dataset can be used to address two research questions namely:

- To what extent can we determine race-specific differential expression in prostate cancer?
- What genes are differentially expressed between tumour and normal prostate cancer samples, taking into account any race-specific effects?

The hypothesis underlying this analysis is that ethnicity-based differences in disease outcomes are driven by genetics define ancestry groups and can be observed at the molecular level.

2.2 Excercise 1

To illustrate the principles of RNA-seq analysis, you will analyse 12 RNA-seq samples that represent tumour-normal pairs (**prostate tumor PT**; **normal normal prostate NP**) from six individuals. Three individuals are African American (AA) and three are European American (EA).

patientID	normalID	tumorID	ethnicity
487	NP4	PT4	AA
881	NP6	PT6	AA
2249	NP10	PT10	AA
376	NP2	PT2	EA
647	NP5	PT5	EA
3365	NP13	PT13	EA

[ ]:
```
ls data/*.fastq.gz
```

The FASTQ files contain the raw sequence reads for each sample. There are four lines per read: 1. Header 2. Sequence 3. Separator (usually a '+') 4. Encoded quality value

Take a look at one of the FASTQ files.

[ ]:
```
zless data/PT6_1.fastq.gz | head
```

Find out more about FASTQ formats at https://en.wikipedia.org/wiki/FASTQ_format

2.3 Questions

2.3.1 Q1: Why is there more than one FASTQ file per sample?

Hint: think about why there is a PT6_1.fastq.gz and a PT6_2.fastq.gz

Now let's move on to mapping these RNA-Seq reads to the genome using HISAT2.

3 Mapping RNA-Seq reads to the genome using HISAT2

3.1 Introduction

For this exercise, we have restricted the number of reads in each sample to those mapping largely to a single chromosome to reduce the mapping time. This is sufficient to illustrate the principles of differential expression analysis.

The objectives of this part of the tutorial are to: * use HISAT2 to build an index from the reference genome * use HISAT2 to map RNA-Seq reads to the reference genome

3.1.1 Mapping RNA-Seq reads to a genome

By this stage, you should have already performed a standard NGS quality control check on your reads to see whether there were any issues with the sample preparation or sequencing. In the interest of time, we won't be doing that as part of this tutorial, but feel free to use the tools from earlier modules to give that a go later if you have time.

Next, we map our RNA-Seq reads to a reference genome to get context. This allows you to visually inspect your RNA-Seq data, identify contamination, novel exons and splice sites as well as giving you an overall feel for your transcriptome.

HISAT2 To map the RNA-Seq reads from our five samples to the reference genome, we will be using HISAT2, a fast and sensitive splice-aware aligner. HISAT2 compresses the genome using an indexing scheme based on the Burrows-Wheeler transform (BWT) and Ferragina-Manzini (FM) index to reduce the amount of space needed to store the genome. This also makes the genome quick to search, using a whole-genome FM index to anchor each alignment and then tens of thousands local FM indexes for very rapid extensions of these alignments.

For more information, and to find the original version of *Figure 2*, please see the HISAT paper:

> **HISAT: a fast spliced aligner with low memory requirements** Daehwan Kim, Ben Langmead and Steven L Salzberg Nat Methods. 2015 Apr;12(4):357-60. doi:10.1038/nmeth.3317

HISAT2 is a splice-aware aligner which means it takes into account that when a read is mapped it may be split across multiple exons with (sometimes large) intronic gaps between aligned regions. As you can see in Figure 2, HISAT2 splits read alignments into five classes based on the number of exons the read alignment is split across and the length of the anchor (longest continuously mapped portion of a split read): * Aligns to a single exon (M) * Alignment split across 2 exons with long anchors over 15bp (2M_gt_15) * Alignment split across 2 exons with intermediate anchors between 8bp and 15bp (2M_8_15) * Alignment split across 2 exons with short anchors less than 7bp (2M_1_7) * Alignment split across more than 2 exons (gt_2M)

HISAT2 used the global index to place the longest continuously mapped portion of a read (anchor). This information is then used to identify the relevant local index. In most cases, HISAT2 will only need to use a single local index to place the remaining portion of the read without having to search the rest of the genome.

For the human genome, HISAT2 will build a single global index and 48,000 local FM indexes. Each of the local indexes represents a 64kb genomic region. The majority of human introns are significantly shorter than 64kb, so >90% of human introns fall into a single local index. Moreover, each of the local indexes overlaps its neighbour by ~1kb which means that it also has the ability to detect reads spanning multiple indexes.

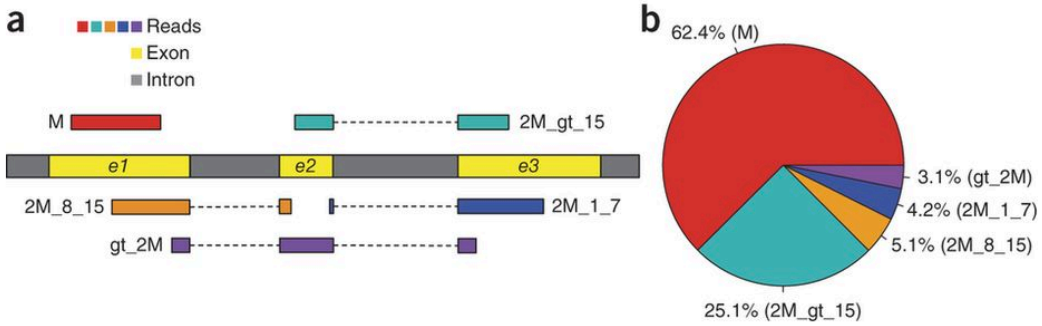


Figure 2. Read types and their relative proportions from 20 million simulated 100-bp reads. Reference: Kim D. et al. Nat Methods. 2015.

There are five HISAT2 RNA-seq read mapping categories: (i) M, exonic read; (ii) 2M_gt_15, junction reads with long, >15-bp anchors in both exons; (iii) 2M_8_15, junction reads with intermediate, 8- to 15-bp anchors; (iv) 2M_1_7, junction reads with short, 1- to 7-bp, anchors; and (v) gt_2M, junction reads spanning more than two exons (Figure 2A). Exoninc reads span only a single exon and represent over 60% of the read mappings in the 20 million 100-bp simulated read dataset.

3.2 Excercise 2

Be patient, each of the following steps will take a couple of minutes!

Look at the usage instructions for hisat2-build.

[ ]:
```
hisat2-build -h
```

This not only tells us the version of HISAT2 we're using (essential for publication methods):

```
HISAT2 version 2.1.0 by Daehwan Kim (infphilo@gmail.com, http://www.ccb.jhu.edu/people/infphil
```

But, that we also need to give `hisat2-build` two pieces of information:

```
Usage: hisat2-build [options]* <reference_in> <ht2_index_base>
```

These are:

- `<reference_in>` location of our reference sequence file (PccAS_v3_genome.fa)
- `<ht2_index_base>` what we want to call our HISAT2 index files (PccAS_v3_hisat2.idx)

Build a HISAT2 index for chromosome 21 of the human reference genome using hisat2-build.

[ ]:
```
hisat2-build data/hsapien_grch38_chr21.fa \
    outputs/hsapien_grch38_chr21_hisat2.idx
```

You can see the generated index files using:

[ ]:
```
ls outputs/hsapien_grch38_chr21_hisat2.idx*
```

Look at the usage for hisat2

[ ]:
```
hisat2 -h
```

Here we can see that HISAT2 needs several parameters so that it can do the mapping:

```
hisat2 [options]* -x <ht2-idx> {-1 <m1> -2 <m2> | -U <r>} [-S <sam>]
```

- `-x <ht2-idx>` the prefix that we chose for our index files with hisat2-build (PccAS_v3_hisat2.idx)
- `{-1 <m1> -2 <m2> | -U <r>}` the left (-1) and right (-2) read files for the sample (MT1_1.fastq and MT1_2.fastq respectively
- `[-S <sam>]` the name of the file we want to write the output alignment to (MT1.sam) as, by default, hisat2 will print the results to the terminal (stdout)

We will also be adding one more piece of information, the maximum intron length (default 500,000 bases). For this analysis, we want to set the maximum intron length to 10,000. We can do this by adding the option `--max-intronlen 10000`.

Map the reads for the PT2 sample using HISAT2

[ ]:
```
hisat2 -x outputs/hsapien_grch38_chr21_hisat2.idx \
    -1 data/PT2_1.fastq.gz \
    -2 data/PT2_2.fastq.gz -S outputs/PT2.sam
```

HISAT2 has written the alignment in SAM format. This is a format which allows humans to look at our alignments. However, we need to convert the SAM file to its binary version, a BAM file. We do this for several reasons. Mainly we do it because most downstream programs require our alignments

to be in BAM format and not SAM format. However, we also do it because the BAM file is smaller and so takes up less (very precious!) storage space. For more information, see the format guide: http://samtools.github.io/hts-specs/SAMv1.pdf.

Convert the SAM file to a BAM file

[ ]:
```
samtools view -S -o outputs/PT2.bam -b outputs/PT2.sam
```

Next we need to sort the BAM file ready for indexing. When we aligned our reads with HISAT2, alignments were produced in the same order as the sequences in our FASTQ files. To index the BAM file, we need the alignments ordered by their respective positions in the reference genome. We can do this using samtools sort to sort the alignments by their co-ordinates for each chromosome

Sort the BAM file

[ ]:
```
samtools sort -o outputs/PT2_sorted.bam outputs/PT2.bam
```

Next, we need to index our BAM file. This makes searching the alignments much more efficient. It allows programs like IGV (which we will be using to visualise the alignment) to quickly get the alignments that overlap the genomic regions you're looking at. We can do this with samtools index which will generate an index file with the extension **.bai**.

Index the BAM file so that it can be read efficiently by IGV.

[ ]:
```
samtools index outputs/PT2_sorted.bam
```

Now, repeat this process of mapping, converting (SAM to BAM), sorting and indexing with the reads from the NP2 sample.

You can run the previous steps as a single command.

[ ]:
```
hisat2 -x outputs/hsapien_grch38_chr21_hisat2.idx \
    -1 data/NP2_1.fastq.gz \
    -2 data/NP2_2.fastq.gz | \
    samtools view -S -b - | \
    samtools sort -o outputs/NP2_sorted.bam - &&
    samtools index outputs/NP2_sorted.bam
```

3.3 Questions

Q1: How many index files were generated when you ran hisat2-build? *Hint: look for the files with the .ht2 extension*

Q2: What was the overall alignment rate for the PT2 sample to the reference genome? *Hint: look at the the output from the HISAT2 commands*

Q3: How does the alignment rate compare with that of the NP2 sample? *Hint: look at the the output from the hisat2 commands*

Q4: How many NP2 reads were not aligned to the reference genome? *Hint: look at the the output from the hisat2 commands, you're looking for reads (not read pairs) which have aligned 0 times (remember that one read from a pair may map even if the other doesn't)*

> The alignments generated here can be used for transcriptome quantification using tools such as feautureCounts or htseq-count and a reference transcriptome to quantify read counts at the gene or transcript level. Here we will use these alignments for visualization and then turn our attention to trancriptome alignments.

4 Visualising transcriptomes with IGV

4.1 Introduction

The Integrative Genome Viewer (IGV) allows us to visualise genomic datasets.

The objectives of this part of the tutorial are: * load an annotation file into IGV and explore gene structure * load read alignments into IGV and inspect read alignments

> Reference: Online IGV User Guide - see more information on all IGV features and functions.

4.1.1 Launch IGV and Load Data

```
[ ]: igv &
```

This will open the IGV main window. Now, we need to tell IGV which genome we want to use. IGV has many pre-loaded genomes available so load `Human (hg38)`.

Load custom gene annotation file We not only want to see where our reads have mapped, but what genes they have mapped to. For this, we have an annotation file in GFF/GTF format. This contains a list of features, their co-ordinates and orientations which correspond to our reference genome.

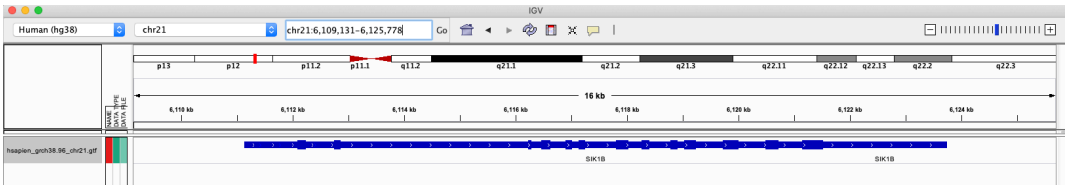
```
#!genome-build GRCh38.p12
#!genome-version GRCh38
#!genome-date 2013-12
#!genome-build-accession NCBI:GCA_000001405.27
#!genebuild-last-updated 2018-11
21      havana  gene    5011799 5017145 .       +       .       gene_id "ENSG00000279493"; gene_version "1"; gene_name "FP565260.4"; ge
21      havana  transcript      5011799 5017145 .       +       .       gene_id "ENSG00000279493"; gene_version "1"; transcript_id "ENS
; gene_biotype "protein_coding"; transcript_name "FP565260.4-201"; transcript_source "havana"; transcript_biotype "protein_coding"; tag
21      havana  exon    5011799 5011874 .       +       .       gene_id "ENSG00000279493"; gene_version "1"; transcript_id "ENST00000624
 "havana"; gene_biotype "protein_coding"; transcript_name "FP565260.4-201"; transcript_source "havana"; transcript_biotype "protein_codi
art_NF"; tag "basic"; transcript_support_level "5";
21      havana  CDS     5011799 5011874 .       +       0       gene_id "ENSG00000279493"; gene_version "1"; transcript_id "ENST00000624
 "havana"; gene_biotype "protein_coding"; transcript_name "FP565260.4-201"; transcript_source "havana"; transcript_biotype "protein_codi
RNA_start_NF"; tag "basic"; transcript_support_level "5";
21      havana  exon    5012548 5012687 .       +       .       gene_id "ENSG00000279493"; gene_version "1"; transcript_id "ENST00000624
 "havana"; gene_biotype "protein_coding"; transcript_name "FP565260.4-201"; transcript_source "havana"; transcript_biotype "protein_codi
art_NF"; tag "basic"; transcript_support_level "5";
```

Example Human Genome Transcript Annotation File

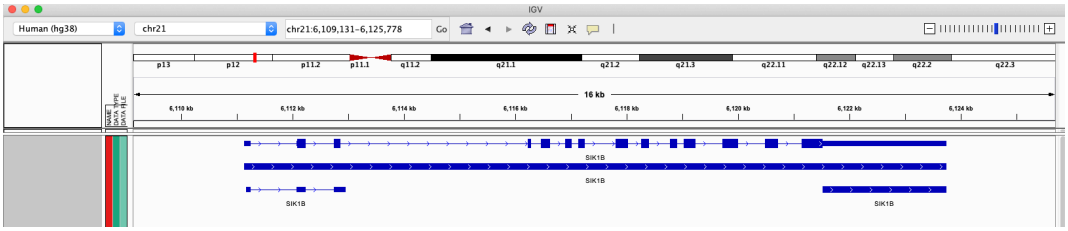
Load your annotation file into IGV. Go to **"File -> Load from File..."**. **Select "hsapien_grch38.96_chr21.gtf"** and click **"Open"**.

This will load a new track called "hsapien_grch38.96_chr21.gtf". The track is currently shown as a density plot. You will need to zoom in to see individual genes.

Search for the gene `SI1K` by typing "SI1K" in the search box to zoom in and centre the view of this gene.



To get a clearer view of the gene structure, right click on the annotation track and click **"Expanded"**.



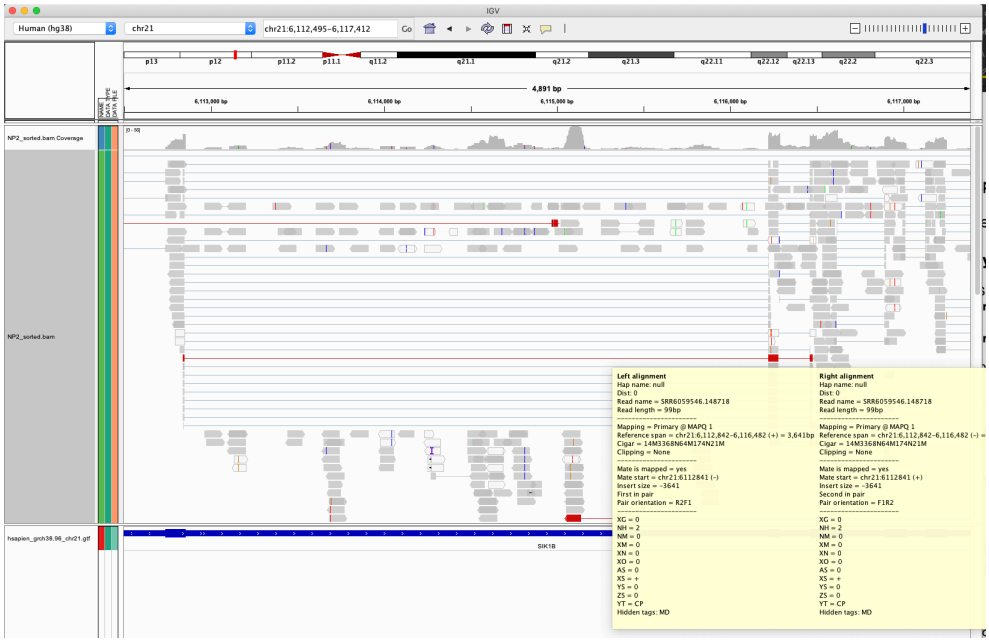
In the annotation track, genes are presented as blue boxes and lines. These boxes represent exons, while the lines represent intronic regions. Arrows indicate the direction (or strand) of transcription for each of the genes. Now we have our genome and its annotated features, we just need to read in alignments from our two samples.

Load alignment files Load your alignment file for the **NP2** sample into IGV. Go to **"File -> Load from File…". Select "NP2_sorted.bam" and click "Open"**.

> **Note:** BAM files and their corresponding index files must be in the same directory for IGV to load them properly.

This will load a new track called “NP2_sorted.bam” which contains the read alignments for the NP2 sample. We can change how we visualise our data by altering the view options. By default, IGV will display reads individually so they are compactly arranged. If you were to hover over a read in the default view, you will only get the details for that read. However, if we change our view so that the reads are visualised as pairs, the read pairs will be joined together by line and when we hover over either of the reads, we will get information about both of the reads in that pair.

To view our reads as pairs, right click on the NP2_sorted.bam alignment track and click “View as pairs”.



To condense the alignment, right click on the NP2_sorted.bam alignment track and click “Squished”.

For more information on sorting, grouping and visualising read alignments, see the IGV User Guide.

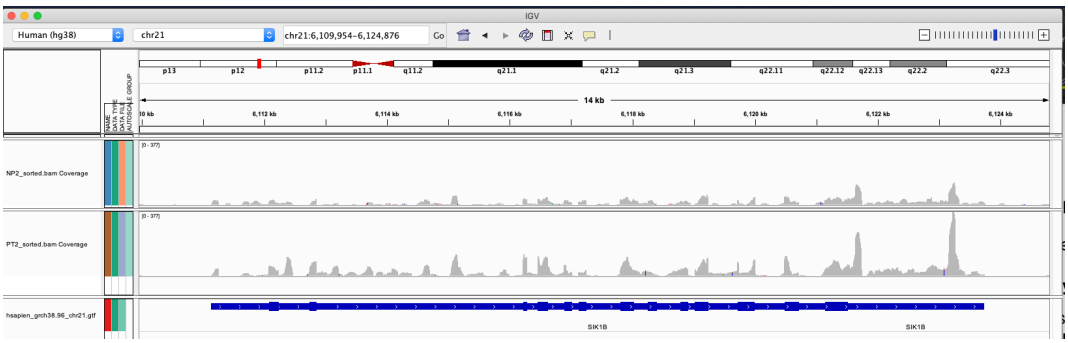
Load the alignment from the matched tumor sample PT2.

Using the search box in the toolbar, go to gene locus: `SI1K`. Look at the coverage range for this viewing window on the left-hand side. While at first glance it may seem like this gene may be differentially expressed between the two conditions, this may not be the case when you consider sequencing depth etc. If a sample has been sequenced to a greater depth we would expect more reads to map there in general.

Here to compare the reads on the same scale, we can use the ’Group Autoscale’ function. First, right click each coverage track and temporarily ’hide’ it. Now by clicking on both coverage tracks, select

'Group Autoscale'.

Now we compare the two samples:



4.2 Questions

4.2.1 Q1: How many CDS features are there in "SIK1B"?

Hint: Jump to this gene or locus or use unix commands to look at the annotation file 'data/h-sapien_grch38.96_chr21.gtf'

4.2.2 Q2: Does the RNA-seq mapping agree with the gene model in blue?

Hint: Look at the coverage track and split read alignments.

4.2.3 Q3: Do you think this gene (SIK1B) is differentially expressed between prostate cancer and normal adjacent tissue? Is looking at the coverage plots alone a reliable way to assess differential expression?

Hint: Look at the coverage similarities/differences between the normal and cancer sample.

5 Transcript quantification with Kallisto

5.1 Introduction

After visually inspecting the genome alignment, the next step in a typical RNA-Seq analysis is to estimate transcript abundance. To do this, reads are assigned to the transcripts they came from. These assignments are then used to quantify gene or transcript abundance (expression level).

For this tutorial, we are using Kallisto to assign reads to a set of transcript sequences and quantify transcript abundance. Kallisto does not assemble transcripts and cannot identify novel isoforms. So, when a reference transcriptome isn't available, the transcripts will need to be assembled *de novo* from the reads. However, for this tutorial, we already have a reference transcriptome available.

The objectives of this part of the tutorial are: * use Kallisto to index a transcriptome * use Kallisto to estimate transcript abundance

5.1.1 Quantify Transcript Expression with Kallisto

Many of the existing methods used for estimating transcript abundance are alignment-based. This means they rely on mapping reads onto the reference genome. The gene expression levels are then calculated by counting the number of reads overlapping the transcripts. However, read alignment is a computationally and time intensive process. So, in this tutorial, we will be running Kallisto which uses a fast, alignment-free method for transcript quantification.

> **Near-optimal probabilistic RNA-seq quantification** Nicolas L Bray, Harold Pimentel, Páll Melsted and Lior Pachter Nat Biotechnol. 2016 May;34(5):525-7. doi: 10.1038/nbt.3519

Kallisto uses a process called pseudoalignment to make it efficient. Rather than looking at where the reads map, Kallisto uses the compatibility between the reads and transcripts to estimate transcript abundance. Thus, most transcript quantification with Kallisto can be done on a simple laptop (**Figure 3**).

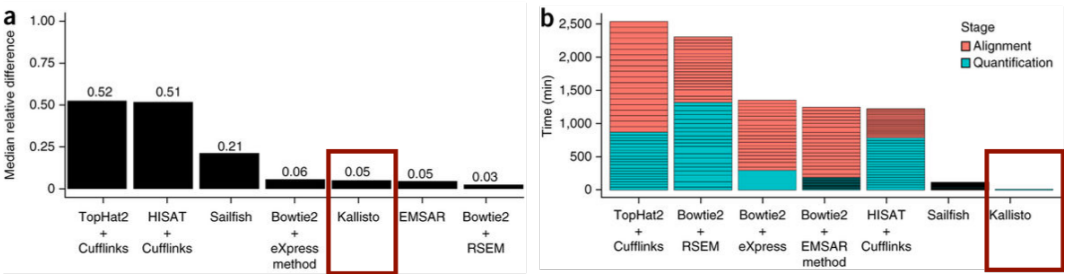


Figure 3. Performance of kallisto and other methods *(a) Accuracy of kallisto, Cufflinks, Sailfish, EMSAR, eXpress and RSEM on 20 RSEM simulations of 30 million 75-bp paired-end reads. (b) Total running time in minutes for processing the 20 simulated data sets of 30 million paired-end reads described in a. Please see the Kallisto publication for original figure and more information.*

Step 1: building a Kallisto index

As with alignment-based methods, Kallisto needs an index. To generate the index, Kallisto first builds a transcriptome de Bruijn Graph (TBDG) from all of the k-mers (short sequences of k nucleotides) that it finds in the transcriptome. Each node in the graph corresponds to a k-mer and

each transcript is represented by its path through the graph. Using these paths, each k-mer is assigned a k-compatibility class. Some k-mers will be redundant i.e. shared by the same transcripts. These are skipped to make the index compact and quicker to search. A great worked example of this process can be found here.

The command kallisto index can be used to build a Kallisto index from transcript sequences.

[ ]:
```
kallisto index
```

Here we can see the version of Kallisto that we're using (useful for publication methods) and the information that we'll need to give kallisto index. The only information we need to give kallisto index is the location of our transcript sequences (data/hsapien_grch38_transcripts.fa). However, it's useful to have a meaningful filename for the resulting index. We can add this by using the option -i which expects a value, our index prefix (GRCh38_kallisto).

Step 2: estimating transcript abundance
With this Kallisto index, you can use kallisto quant to estimate transcript abundances. You will need to run this command separately for each sample.

[ ]:
```
kallisto quant
```

We can see that kallisto quant needs us to tell it where our sample reads are. Although we don't have to, it's usually a good idea to keep the results of each quantification in a different directory. This is because the output filename are always the same (e.g. abundances.tsv). If we ran a second analysis, these could get overwritten. To use a different output directory, we can use the -o option. We will also be using the -b option for bootstrapping.

Bootstrapping Not all reads will be assigned unambiguously to a single transcript. This means there will be "noise" in our abundance estimates where reads can be assigned to multiple transcripts. Kallisto quantifies the uncertainty in its abundance estimates using random resampling and replacement. This process is called bootstrapping and indicates how reliable the expression estimates are from the observed pseudoalignment. Bootstrap values can be used downstream to distinguish the technical variability from the biological variability in your experiment.

5.2 Exercise 4

Build an index called GRCh38_kallisto from transcript sequences

Note Depending on the specifications of your machine, this step may take a long time. While the index is being built, spend time reading through the remaining sections of the tutorial.

[ ]:
```
kallisto index -i outputs/GRCh38_ch21_kallisto \
    data/hsapiens_chr21_transcripts.fa
```

Quantify the transcript expression levels for the PT6 sample with 100 bootstrap samples and store the results in the output directory PT6.

[ ]:
```
kallisto quant -i outputs/GRCh38_ch21_kallisto \
    -o outputs/PT6 -b 100 \
    data/PT6_1.fastq.gz data/PT6_2.fastq.gz
```

You'll find your Kallisto results in a new output directory which we called PT6. Let's take a look.

[ ]:
```
ls outputs/PT6
```

Running kallisto quant generated three output files in our PT6 folder: * **abundance.h5** - HDF5 binary file containing run info, abundance esimates, bootstrap estimates, and transcript length information length. * **abundance.tsv** - Plain text file containing abundance estimates (doesn't contain bootstrap estimates). * **run_info.json** - JSON file containing information about the run.

Note: when the number of bootstrap values (-b) is very high, Kallisto will generate a large amount of data. To help, it outputs bootstrap results in HDF5 format (abundance.h5). This file can be read directly by sleuth.

The file **PT6/abundance.tsv** will have the abundance estimates for each gene for the PT6 sample. Let's take a quick look.

[ ]:
```
head outputs/PT6/abundance.tsv
```

In **PT6/abundance.tsv** there are five columns which give us information about the transcript abundances for our PT6 sample. * **target_id** - Unique transcript identifier. * **length** - Number of bases found in exons. * **eff_length** - Effective length. Uses fragment length distribution to determine the effective number of positions that can be sampled on each transcript. * **est_counts** - Estimated counts. This may not always be an integer as reads which map to multiple transcripts are fractionally assigned to each of the corresponding transcripts. * **tpm** - Transcripts per million. Normalised value accounting for length and sequence depth bias.

In the last column we have our normalised abundance value for each gene. These are our transcripts per million or TPM. If you have time at the end of this tutorial, see our normalisation guide which covers common normalisation methods and has a bonus exercise.

To get the result for a specific transcript, we can use grep.

[ ]:
```
grep ENST00000399975 outputs/PT6/abundance.tsv
```

If we wanted to get the TPM value for a particular transcript, we can use awk.

[ ]:
```
awk -F"\t" '$1=="ENST00000399975.7" {print $5}' outputs/PT6/abundance.tsv
```

Use kallisto to quantify the expression of the remaining samples. You can either use the command shown above for each individual sample. Alternatively, you can create a `bash` script to iterate through all the samples and quantify transcriptome expression using Kallisto (See commands below)

[ ]:
```
for r1 in data/*_1.fastq.gz
do
  echo $r1
  sample=$(basename $r1)
  sample=${sample%_1.fastq.gz}
  echo "Processing sample: "$sample

  kallisto quant -i outputs/GRCh38_ch21_kallisto \
    -o outputs/${sample} -b 100 \
```

```
    data/${sample}_1.fastq.gz data/${sample}_2.fastq.gz

done
```

5.3 Questions

5.3.1 Q1: What k-mer length was used to build the Kallisto index?

Hint: look at the terminal output from kallisto index

5.3.2 Q2: How many transcript sequences are there in hsapien_grch38_transcripts.fa?

Hint: you can use grep or look at the terminal output from kallisto quant or in the run_info.json files

5.3.3 Q3: What is the transcripts per million (TPM) value for ENST00000399975.7 (USP16) in each of the samples?

Hint: use grep to look at the abundance.tsv files

5.3.4 Q4: Do you think ENST00000399975.7 is differentially expressed?

6 Differential Expression Analysis With Sleuth

6.1 Introduction

In the previous section, we quantified transcript abundance. In this section, you will be using sleuth to do some simple quality checks and get a first look at the results.

The objectives of this part of the tutorial are: * use sleuth to perform quality control checks * use sleuth to identify differentially expressed (DE) transcripts * use sleuth to investigate DE transcripts

6.1.1 Differential expression analysis (DEA)

The goal of differential expression analysis is to identify genes whose expression levels differ between experimental conditions. We don't normally have enough replicates to do traditional tests of significance for RNA-Seq data. So, most methods look for outliers in the relationship between average abundance and fold change. The underlying assumption of many approaches is most genes are not differentially expressed.

Rather than just using a fold change threshold to determine which genes are differentially expressed, DEA tools use a variety of statistical tests to assess significant diffferences betweem experimental conditions. These tests give us a **p-value** which is an estimate of how often your observations would occur by chance.

However, we perform these comparisons for each one of the thousands of genes/transcripts in our dataset. A p-value of 0.01 estimates a probability of 1% for seeing our observation just by chance. In an experiment like with 5,000 genes we would expect 5 genes to be significantly differentially expressed by chance (i.e. even if there were no difference between our conditions). Instead of using a p-value we use a **q-value** to account for multiple testing and adjusts the p-value accordingly.

6.1.2 sleuth

sleuth is a companion tool for Kallisto. Unlike most other tools, sleuth can utilize the technical variation information generated by Kallisto so that you can look at both the technical and biological variation in your dataset.

For DEA, sleuth essentially tests two models, one which assumes that the abundances are equal between the two conditions (reduced) and one that does not (full). To identify DE transcripts it identifies those with a significantly better fit to the "full" model. For more information on sleuth and how it works, see Lior Pachter's blog post A sleuth for RNA-Seq.

sleuth is written in the R statistical programming language, as is almost all RNA-Seq analysis software. Helpfully, it produces a web page (Shiny Application) that allows interactive graphical analysis of the data. However, we strongly recommend learning R for anyone doing a significant amount of RNA-seq analysis. It is nowhere near as hard to get started with as full-blown programming languages such as Perl or Python!

6.2 Exercise 5

6.2.1 Running Sleuth

Analysis Objective:

We want to use sleuth to investigate transcript differential expresssion between high grade prostate cancer and matched noncancer adjacent tissue samples in the context of ethnicity differences between African American (AA) and European American (EA) individuals.

Before you begin:

[ ]:
```
# Configure the option for a web browser for R at the command line
export R_BROWSER='firefox'
```

Navigate the the **R programming environment** by typing the following on the command line:

```
R
```

Run all the following commands within the R console - copy and paste the commands as is into the R console.

Step 0: configure differential expression analysis

- load relevant analysis packages
- load pertinent datasets into the R environment

[ ]:
```
# be sure to navigate to the correct working directory within R:
setwd("/home/manager/course_data/rna_seq_human")
```

[ ]:
```
# load sleuth package
library(sleuth)
library(dplyr)
```

[ ]:
```
# load sample metadata
sample_info <- read.table(file="data/sample_info.txt", header = T, sep = "\t")
sample_info
```

How many individual patients have been sequenced in this experiment?

Turn to your neighbor and discuss if you are unsure of the answer.

Hint: Look closely at the `patientID` column.

Next, create a vector of file paths pointing to kallisto quantification results

[ ]:
```
kallisto_result_directory <-  sapply(X = sample_info$sample,
                                     function(id) file.path('outputs', id))
kallisto_result_directory
```

Then configure sample_to_covariates data frame

[ ]:
```
s2c <- dplyr::select(sample_info,
    sample = sample, sample_type, ethnicity, individualID=patientID)

kallisto_result_directory <-  sapply(sample_info$sample, function(id)
    file.path('outputs', id)) # path to kallisto results

s2c <- dplyr::mutate(s2c, path = kallisto_result_directory)
s2c
```

[ ]:
```
# transcript to gene annotations
filename <-"data/hsapiens_chr21_transcript_to_gene.csv"
t2g <- read.table(filename, header = T, sep = ',')
names(t2g) <- c('target_id', 'ensembl_gene_id', 'gene_symbol',
                'gene_biotype', 'gene_descrition')
t2g <- t2g[,1:4]
head(t2g)
dim(t2g)
```

Step 1: create a sleuth object (so) Here it is important to specify the **model design** where you enumerate the covariates you want to model. These can include both technical and biological variables under study. For example, if the processing batch is known, this would be important to add to the model because we need to account for expression differences that are related to technical factors so that we can make sure we are modeling the biological variability that is our true variable of interest.

Here we consider the following covariates: * `individualID`: unique identifier for each individual * `sample_type`: normal vs. cancer (tumor) * `ethnicity`: African American (AA) vs/ European American (EA)

[ ]:
```
# create design matrix specification
design <- ~ individualID + sample_type + ethnicity
```

This statement is specifying how we want to model the variation in gene expression. This statement is loosely saying: we expect that the level of expression of a gene is dependent on: * the individual in whom expression is measured * whether we are looking at a cancer or normal sample * the ethnicity of the individual from whom the sample is collected

With this model specification, we can infer ethnicity-related differential expression while controlling for variability in expression between tumor and normal samples or inherent differences in expression between individual samples.

Note the paired design of this study also suggests that we should consider controlling for inter-individual expression in gene expression between individuals.

For a more extensive treatment of how to setup design matrices for gene expression experiments read through A guide to creating design matrices for gene expression experiments.

[ ]:
```
# Create sleuth object (a group of kallistos) for analysis
so <- sleuth_prep(sample_to_covariates=s2c, # sample_to_covariates data frame
                  full_model=design, # model design matrix
                  target_mapping = t2g,  # transcript to gene annotations
                  aggregation_column = "ensembl_gene_id",
                  extra_bootstrap_summary = TRUE,
                  read_bootstrap_tpm = TRUE,
                  transformation_function = function(x) log2(x + 0.5),
                  num_cores=1)
```

Getting Help In R

If you are unsure about what a function or analysis routine does or what it's inputs or outpus are, you can just type `?[command name]`. The `help` documentation for that function will appear. Scroll through using the [space_bar].

To exit, simply type the letterq

Step 2: Fit the sleuth model Now we fit sleuth's 'measurement error model'

[ ]:
```
# Fit the sleuth model (full - consider all covariates simultaneoulsy)
so <- sleuth_fit(so, formula=design, fit_name="full")
```

[ ]:
```
# fit a reduced model - fit a model without the final factor
so <- sleuth_fit(so, formula=~ individualID + sample_type , fit_name="reduced")
```

Step 3: Statistical testing between conditions

[ ]:
```
# likelihood ratio test between the two models
# tests for ancestry-related differences
# --- difference between the full model with 3 coveriates
# versus the reduced model with 2 covariates
so <- sleuth_lrt(so, 'reduced', 'full')
```

[ ]:
```
sleuth_table_tx <- sleuth_results(obj = so,
                                  test = 'reduced:full',
                                  test_type ='lrt',
                                  show_all = FALSE,
                                  pval_aggregate = FALSE)
head(sleuth_table_tx, 5)
```

What do the results mean?

Here we see a table with the top 5 transcripts in the differential expression analysis results table. The key columns are: * **pval** - p-value of the chosen model * **qval** - false discovery rate adjusted p-value

A rule of thumb is to consider a gene significant if the q-value is less than a prespecified threshold (typically 5% or 0.05).

> How many genes are significantly differentially expressed due to ethnicity? Are the results in line with your expectations?**

Visualize transcript abundance for top hit from differential expression analysis:

[ ]:
```
topDE_hit <- sleuth_table_tx[1,"target_id"]
```

[ ]:
```
# group view
plot_bootstrap(so, topDE_hit, units = "est_counts", color_by = "ethnicity")
```

[ ]:
```
# paired sample view
library(ggplot2)
df <- get_bootstrap_summary(so , topDE_hit)

ggplot(data = df, aes(x = sample_type, ymin = min, lower = lower,
      middle = mid, upper = upper, ymax = max)) +
      geom_boxplot(stat="identity", aes(fill=ethnicity)) +
      facet_wrap(~individualID)
```

Here we see visually that there is a difference in the expression between the cancer and normal sample for European American (EA) samples (3/3) but not for African American (AA) samples. In fact it seems this transcript is not expressed at all in 2/3 AA samples. Overall this observed difference is not statistically significant. There are a few possible explanations some of which could be: * ethnicity-related differences in expression are subtle and are difficult to model * we have not accounted for all the possible technical variation in the data (e.g. we did not model batch variables) * there may be some sample QC issues that need to be double checked

> **Can you think of any other reasons?**

Wald's Test: testing for significant differences between conditions

You can look at your sleuth object to see what models you have fit and their design matrix specification like this:

[ ]:
```
models(so)
```

If you are interested in the effect of a particular covariate, you can use Wald's test to e.g. test for significant expression differences between normal and disease samples (while controlling for *__inter-individual__ variation and ethnicity-specific diffences).

[ ]:
```
# Wald test for individual coefficients (betas) tumor vs. normal
so <- sleuth_wt(obj = so, which_beta = "sample_typetumor", which_model = 'full')
```

[ ]:
```
# summary table
de_sampletype <- sleuth_results(so, test='sample_typetumor',
                     test_type = "wt", which_model = "full", pval_aggregate = F)
head(de_sampletype, 5)
```

[ ]:
```
# visualize results
df <- get_bootstrap_summary(so , de_sampletype[1,'target_id'])

ggplot(data = df, aes(x = sample_type, ymin = min, lower = lower,
      middle = mid, upper = upper, ymax = max)) +
      geom_boxplot(stat="identity", aes(fill=ethnicity)) +
      facet_wrap(~individualID)
```

In this comparison, we see strong statistically significant differences between sample types independent of ethnicity, suggesting that the normal-vs-disease differences in expression are more pronounced than ethnicity-related changes.

[ ]:
```
# save sleuth result object
save(so, file = "outputs/sleuthObj.RData")
```

6.2.2 Using Sleuth for Exploratory Data Analysis

Our model above generated few ancestry-specific differentially expressed genes after false discovery rate correction (**Caveat:** tutorial data is limited to transcripts mapping to chr21). Fortunately, sleuth generates an interactive Shiny application we can use for further exploratory data analysis to better understand the factors driving differential expression in these samples

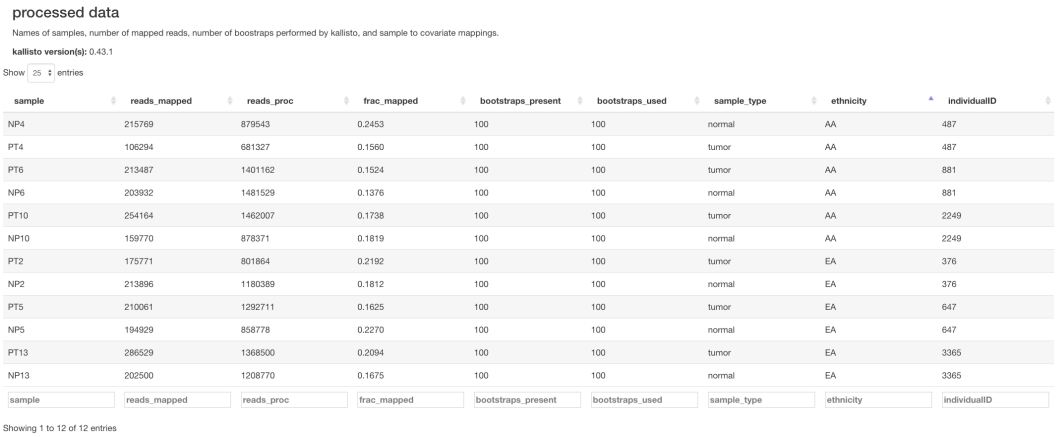
[ ]:
```
# launch interactive exploration of sleuth differential expression object
sleuth_live(so)

# if you have issues connecting to the browswer try this alternative:
# sleuth_live(so, options=list(launch.broswer=FALSE))
```

Using sleuth to quality check transcript quantification Quality control checks are absolutely vital at every step of the experimental process. We can use sleuth to perform simple quality checks (QC) on our dataset.

At the top of the page, sleuth provides several tabs which we can use to determine whether the data is of good quality and whether we should trust the results we get.

First, lets take a look at a summary of our dataset.

In the Shiny App that has been launched, click on *"summaries -> processed data"*.



processed data

Names of samples, number of mapped reads, number of boostraps performed by kallisto, and sample to covariate mappings.

kallisto version(s): 0.43.1

Show 25 entries

sample	reads_mapped	reads_proc	frac_mapped	bootstraps_present	bootstraps_used	sample_type	ethnicity	individualID
NP4	215769	879543	0.2453	100	100	normal	AA	487
PT4	106294	681327	0.1560	100	100	tumor	AA	487
PT6	213487	1401162	0.1524	100	100	tumor	AA	881
NP6	203932	1481529	0.1376	100	100	normal	AA	881
PT10	254164	1462007	0.1738	100	100	tumor	AA	2249
NP10	159770	878371	0.1819	100	100	normal	AA	2249
PT2	175771	801864	0.2192	100	100	tumor	EA	376
NP2	213896	1180389	0.1812	100	100	normal	EA	376
PT5	210061	1292711	0.1625	100	100	tumor	EA	647
NP5	194929	858778	0.2270	100	100	normal	EA	647
PT13	286529	1368500	0.2094	100	100	tumor	EA	3365
NP13	202500	1208770	0.1675	100	100	normal	EA	3365

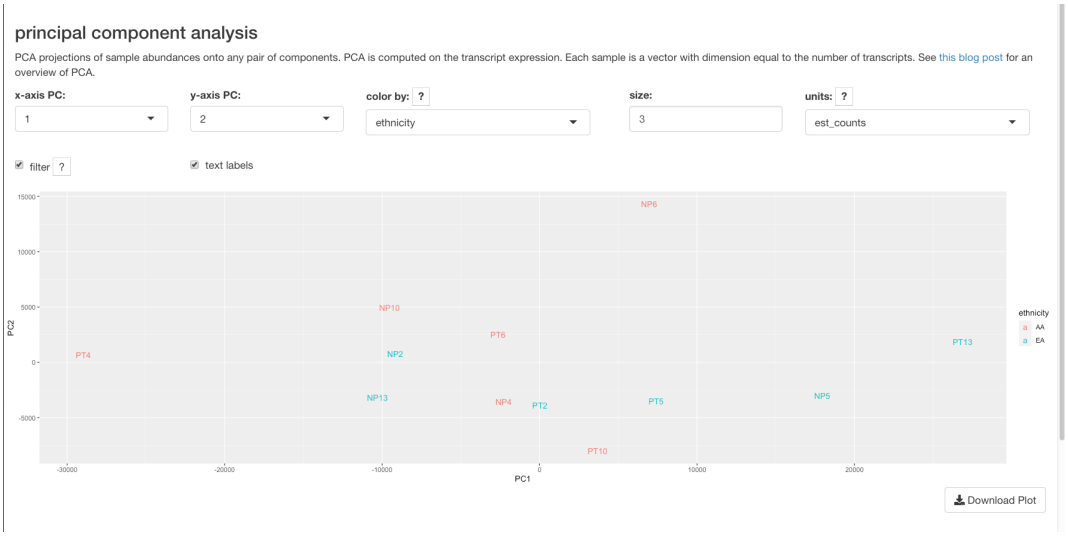
Showing 1 to 12 of 12 entries

Notice that the number of reads mapping differs quite a bit across samples? This is why we QC our data. >90% of the reads mapped to the genome, but only 15-25% are assigned to the transcriptome. This suggests that there may be some residual ribosomal RNA left over from the RNA preparation. It's not a problem if we still have enough reads and replicates for our analysis.

In some cases, we can identify samples which don't agree with other replicates (outliers) and samples which are related by experimental bias (batch effects). If we don't have many replicates, it's hard to detect outliers and batch effects meaning our power to detect DE genes is reduced.

Principal component analysis (PCA) plots can be used to look at variation and strong patterns within the dataset. **Batch effects** and outliers often stand out quite clearly in the PCA plot and mean that you can account for them in any downstream analysis.

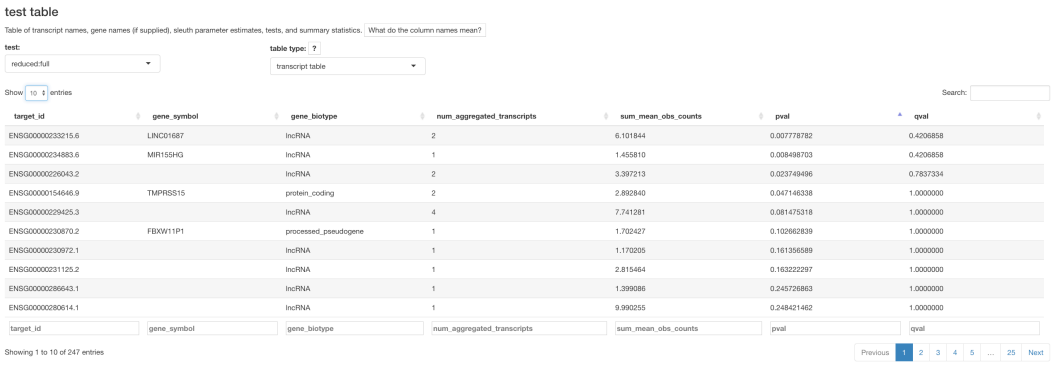
In the Shiny App that has been launched, click on *"maps -> PCA"*.



Here we do not see clear condition-related clusters as we might expect. Indeed samples PT4 and NP6 look like potential outliers. There is perhaps an unaccounted for technical batch effect in these data or those samples represent unmeasured biological signal. Further investigation will be needed included conversations with data generators where appropriate.

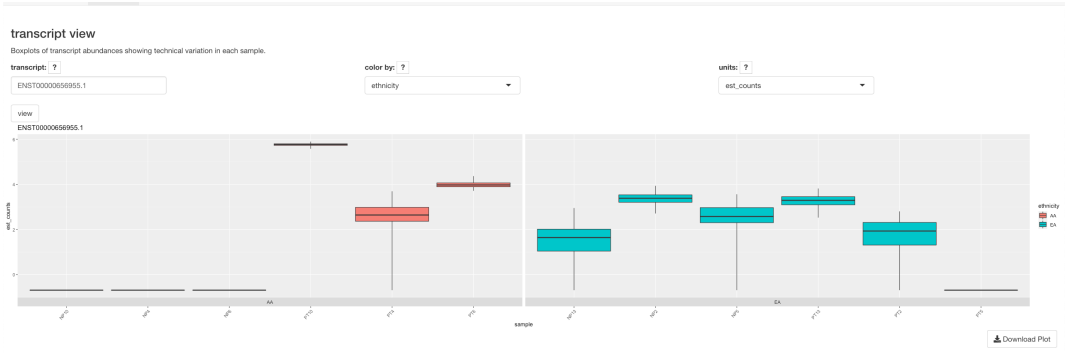
Using sleuth to look at differentially expressed transcripts We used the output from Kallisto to identify DE transcripts using sleuth. Let's take a look and see if we found any.

To see the results of the sleuth DEA, go to *"analyses -> test table".



The important column here is the **q-value**. By default, the table is sorted by the q-value. We can see that the top transcripts do not meet the q-value threshold for statistical significance for this comparison.

To visualize a particular transcript of interest, **Go to *"analyses -> transcript view"*. Enter <TranscriptID> into the "transcript" search box. Click "view".**



On the left you have the abundances for the normal and tumour replicates from AAs and on the right, the EA replicates. We can see that this transcript is differentially expressed between tumour and normal in the AAs but not the EAs. However, while we see a visual difference, the q-value (~0.4) is not statistically significant, suggesting that there may be subtle ancestry-specific effects but additional replicates would be required to test if this is the case

Optional: As this data is publicly available, you can download the full dataset and rerun the analysis to check if some of these ideas hold true or the signal is improved by having additional samples and or more replicates.

6.3 Questions

6.3.1 Q1: What is the most abundantly expressed transcript in the PT6 sample?

Hint: go to *"summaries -> kallisto table"*. Be sure to activate the covariates toggle.

7 Key Aspects of Differential Expression Analysis

7.1 Replicates and power

> 'Biological replicates are parallel measurements of biologically distinct samples that capture random biological variation, which may itself be a subject of study or a noise source.
>
> Technical replicates are repeated measurements of the same sample that represent independent measures of the random noise associated with protocols or equipment'
>
> Blainey, Paul et al. "Points of significance: replication." Nature methods vol. 11,9 (2014): 879-80. doi:10.1038/nmeth.3091'

In order to accurately ascertain which genes are differentially expressed and by how much it is necessary to use replicated data. As with all biological experiments doing it once is simply not enough. There is no simple way to decide how many replicates to do, it is usually a compromise between statistical power and cost. By determining how much variability there is in the sample preparation and sequencing reactions, we can better assess how highly genes are really expressed and more accurately determine any differences. The key to this is performing biological rather than technical replicates. This means, for instance, growing up three batches of parasites, treating them all identically, extracting RNA from each and sequencing the three samples separately. Technical replicates, whereby the same sample is sequenced three times do not account for the variability that really exists in biological systems or the experimental error between batches of parasites and RNA extractions.

Note: more replicates will help improve power for genes that are already detected at high levels, while deeper sequencing will improve power to detect differential expression for genes which are expressed at low levels.

7.2 p-values vs. q-values

When asking whether a gene is differentially expressed we use statistical tests to assign a p-value. If a gene has a p-value of 0.05, we say that there is only a 5% chance that it is not really differentially expressed. However, if we are asking this question for every gene in the genome, then we would expect to see p-values less than 0.05 for many genes even though they are not really differentially expressed. Due to this statistical problem, we must correct the p-values so that we are not tricked into accepting a large number of erroneous results. **Q-values** are p-values which have been corrected for what is known as multiple hypothesis testing. Therefore, it is a qvalue of less than 0.05 that we should be looking for when asking whether a gene is signficantly differentially expressed.

7.3 Altermate software

If you have a good quality genome and genome annotation such as for model organisms e.g. human, mouse, Plasmodium etc. map to the transcriptome to determine transcript abundance. This is even

more relevant if you have variant transcripts per gene as you need a tool which will do its best to determine which transcript is really expressed. Kallisto (Bray et al. 2016; PMID: 27043002 and eXpress (Roberts & Pachter, 2012; PMID: 23160280 are examples of these tools.

7.4 What do I do with a list of differentially expressed genes?

Differential expression analysis results are a list of genes which show differences between conditions. It can be daunting trying to determine what the results mean. On one hand, you may find that there are no real differences in your experiment. Is this due to biological reality or noisy data? On the other hand, you may find several thousands of genes are differentially expressed.

What can you say about that?

Other than looking for genes you expect to be different or unchanged, one of the first things to do is look at common themes in gene functionality across your list. For example, you can carry out Gene Ontology (GO) term enrichment analysis. There are many different algorithms for this, but you could annotate your genes with functional terms from GO using for instance Blast2GO (Conesa et al., 2005; PMID: 16081474) and then use TopGO (Alexa et al., 2005; PMID: 16606683) to determine whether any particular sorts of genes occur more than expected in your differentially expressed genes.

Congratulations, you have reached the end of this tutorial!

8 Normalisation

8.1 Introduction

In the previous section, we looked at estimating transcript abundance with Kallisto. The abundances are reported as **transcripts per million (TPM)**, but what does TPM mean and how is it calculated?

The objectives of this part of the tutorial are:

- *understand why RNA-Seq normalisation metrics are used*
- *understand the difference between RPKM, FPKM and TPM*
- *calculate RPKM and TPM for a gene of interest*

There are many useful websites, publications and blog posts which go into much more detail about RNA-Seq normalisation methods. Here are just a couple (in no particular order):

- What the FPKM? A review of RNA-Seq expression units
- RPKM, FPKM and TPM, clearly explained
- A survey of best practices for RNA-seq data analysis
- The RNA-seq abundance zoo

8.2 Why do we use normalisation units instead of raw counts?

Raw reads counts are the number of reads originating from each transcript which can be affected by several factors:

- **sequencing depth (total number of reads)**
  The more we sequence a sample, the more reads we expect to be assigned.
- **gene/transcript length**
  The longer the gene or transcript, the more reads we expect to be assigned to it.

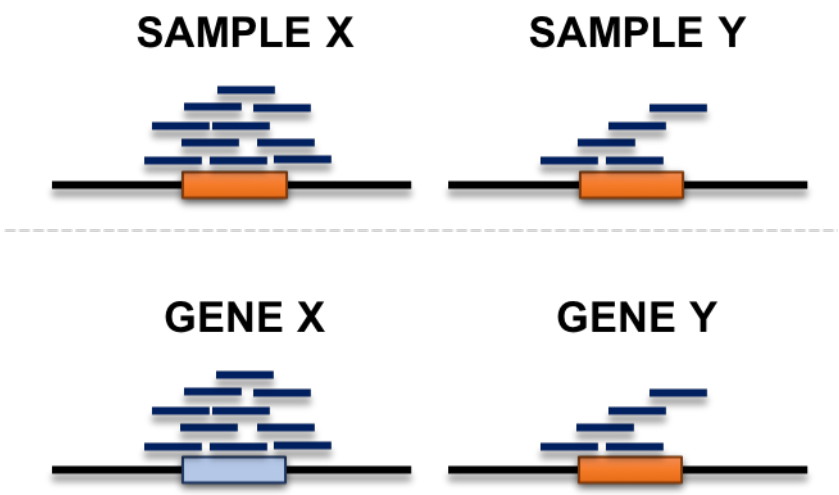


Figure 4. Effect of sequencing depth and gene length on raw read counts

Look at the top part of Figure 4. In which sample, X or Y, is the gene more highly expressed?

Neither, it's the same in both. What we didn't tell you was that the total number of reads generated for sample A was twice the number than for sample B. That meant almost twice the number of reads are assigned to the same gene in sample A than in sample B.

Look at the bottom part of Figure 4. Which gene, X or Y, has the greatest gene level expression?

Neither, they are both expressed at the same level. This time we didn't tell you that gene X is twice the length of gene Y. This meant that almost twice the number reads were assigned to gene X than gene Y.

In the top part of *Figure 4*, the gene in sample X has twice the number of reads assigned to it than the same gene in sample Y. What isn't shown is that sample X had twice the number or total reads than sample Y so we would expect more reads to be assigned in sample X. Thus, the gene is expressed at roughly the same level in both samples. In the bottom part of *Figure 4*, gene X has twice the number of reads assigned to it than gene Y. However, gene X is twice the length of gene Y and so we expect more reads to be assigned to gene X. Again, the expression level is roughly the same.

8.2.1 Reads per kilobase per million (RPKM)

Reads per kilobase (of exon) per million (reads mapped) or **RPKM** is a **within sample** normalisation method which takes into account sequencing depth and length biases.

To calculate RPKM, you first normalise by sequencing depth and then by gene/transcript length.

1. **Get your *per million* scaling factor**
   Count up the total number of reads which have been assigned (mapped) in the sample. Divide

this number by 1,000,000 (1 million) to get your *per million* scaling factor (N).

2. **Normalise for sequencing depth**
   Divide the number of reads which have been assigned to the gene or transcript (C) by the *per million* scaling factor you calculated in step 1. This will give you your reads per million (RPM).
3. **Get your *per kilobase* scaling factor**
   Divide the total length of the exons in your transcript or gene in base pairs by 1,000 (1 thousand) to get your *per kilobase* scaling factor (L).
4. **Normalise for length**
   Divide your RPM value from step 2 by your *per kilobase* scaling factor (length of the gene/-transcript in kilobases) from step 3. This will give you your reads per kilobase per million or RPKM.

This can be simplified into the following equation:

$$RPKM = \frac{C}{LN}$$

Where:

- **C** is number of reads mapped to the transcript or gene
- **L** is the total exon length of the transcript or gene in kilobases
- **N** is the total number of reads mapped in millions

8.2.2 Fragments per kilobase per million (FPKM)

Fragments per kilobase per million or **FPKM** is essentially the same as RPKM except that:

- RPKM is designed for **single-end** RNA-Seq experiments
- FPKM is designed for **paired-end** RNA-Seq experiments

In a paired-end RNA-Seq experiment, two reads may be assigned to a single fragment (in any orientation). Also, in some cases, only one of those reads will be assigned to a fragment (singleton). The only difference between RPKM and FPKM is that FPKM takes into consideration that two reads may be assigned to the same fragment.

8.2.3 Transcripts per million (TPM)

Calculating the **transcripts per million** or **TPM** is a similar process to RPKM and FPKM. The main difference is that you will first normalise for length bias and then for sequencing depth bias. In a nut shell, we are swapping the order of normalisations.

1. **Get your *per kilobase* scaling factor**
   Divide the total length of the exons in your transcript in base pairs by 1,000 (1 thousand) to get your *per kilobase* scaling factor.
2. **Normalise for length**
   Divide the number of reads which have been assigned to the transcript by the per kilobase scaling factor you calculated in step 1. This will give you your reads per kilobase (RPK).

3. **Get the sum of all RPK values in your sample**
   Calculate the RPK value for all of the transcripts in your sample. Add all of these together to get your total RPK value.
4. **Get your *per million* scaling factor**
   Divide your total RPK value from step 3 by 1,000,000 (1 million) to get your *per million* scaling factor.
5. **Normalise for sequencing depth**
   Divide your RPK value calculated in step 2 by the *per million* scaling factor from step 4. You now have your transcripts per millions value or TPM.

8.3 Calculating RPKM and TPM values

To try and answer this, let's look at a worked example. Here, we have three genes (A-C) and three biological replicates (1-3).

Gene	Length	Replicate 1	Replicate 2	Replicate 3
A	2,000 bases	10	12	30
B	4,000 bases	20	25	60
C	1,000 bases	5	8	15

There are two things to notice in our dataset:

- Gene B has twice number reads mapped than gene A, possibly as it's twice the length
- Replicate 3 has more reads mapped than any of the other replicates, regardless of which gene we look at

8.3.1 Calculating RPKM

Step 1: get your per million scaling factor In the table below is the total number of reads which mapped for each of the replicates. To get our *per million* scaling factor, we divide each of these values by 1,000,000 (1 million).

Gene	Replicate 1	Replicate 2	Replicate 3
Total reads mapped	3,500,000	4,500,000	10,600,000
Per million reads	3.5	4.5	10.6

Step 2: normalise for sequencing depth We now divide our read counts by the *per million* scaling factor to get our reads per million (RPM).

Before:

Gene	Replicate 1	Replicate 2	Replicate 3
A	10	12	30
B	20	25	60
C	5	8	15

After:

Gene	Replicate 1 RPM	Replicate 2 RPM	Replicate 3 RPM
A	2.857	2.667	2.830
B	5.714	5.556	5.660
C	1.429	1.778	1.415

Step 3: get your *per kilobase* scaling factor Here we have our gene length in base pairs. For our *per kilobase* scaling factor we need to get our gene length in kilobases by dividing it by 1,000.

Gene	Length (base pairs)	Length (kilobases)
A	2,000	2
B	4,000	4
C	1,000	1

Step 4: normalise for length Finally, we divide our RPM values from step 2 by our *per kilobase* scaling factor from step 3 to get our reads per kilobase per million (RPKM).

Before:

Gene	Replicate 1 RPM	Replicate 2 RPM	Replicate 3 RPM
A	2.857	2.667	2.830
B	5.714	5.556	5.660
C	1.429	1.778	1.415

After:

Gene	Replicate 1 RPKM	Replicate 2 RPKM	Replicate 3 RPKM
A	1.43	1.33	1.42
B	1.43	1.39	1.42
C	1.43	1.78	1.42

Notice that even though replicate 3 had more reads assigned than the other samples and a greater sequencing depth, its RPKM is quite similar. And, that although gene B had twice the number of reads assigned than gene A, its RPKM is the same. This is because we have normalised by both length and sequencing depth.

8.3.2 Calculating TPM

Now we're going to calculate the TPM values for the same example data. As a reminder, here are our three genes (A-C) and three biological replicates (1-3).

Gene	Length	Replicate 1	Replicate 2	Replicate 3
A	2,000 bases	10	12	30
B	4,000 bases	20	25	60
C	1,000 bases	5	8	15

Step 1: get your *per kilobase* scaling factor Again, our gene lengths are in base pairs. For our *per kilobase* scaling factor we need to get our gene length in kilobases by dividing it by 1,000.

Gene	Length (base pairs)	Length (kilobases)
A	2,000	2
B	4,000	4
C	1,000	1

Step 2: normalise for length Now we divide the number of reads which have been assigned to each gene by the per kilobase scaling factor we just calculated. This will give us our reads per kilobase (RPK).

Before:

Gene	Replicate 1	Replicate 2	Replicate 3
A	10	12	30
B	20	25	60

Gene	Replicate 1	Replicate 2	Replicate 3
C	5	8	15

After:

Gene	Replicate 1	Replicate 2	Replicate 3
A	5	6	15
B	5	6.25	15
C	5	8	15

Step 3: get the sum of all RPK values in your sample Next, we sum the RPK values for each of our replices. This will give use our total RPK value for each replicate. To make this example scalable, we assume there are other genes so the total RPK is made up.

Gene	Replicate 1	Replicate 2	Replicate 3
A	5	6	15
B	5	6.25	15
C	5	8	15
...	...	...	...
Total RPK	**150,000**	**202,500**	**450,000**

Step 4: get your *per million* scaling factor Here, instead of dividing our total mapped reads by 1,000,000 (1 million) to get our *per million* scaling factor, we divide our total RPK values by 1,000,000 (1 million).

Gene	Replicate 1	Replicate 2	Replicate 3
Total RPK	150,000	202,500	450,000
Per million RPK	0.1500	0.2025	0.4500

Step 5: normalise for sequencing depth Finally, we divide our individual RPK values from step 2 by the *per million* scaling factor in step 4 to give us our TPM values.

Before:

Gene	Replicate 1	Replicate 2	Replicate 3
A	5	6	15
B	5	6.25	15
C	5	8	15

After:

Gene	Replicate 1	Replicate 2	Replicate 3
A	33.33	29.63	33.33
B	33.33	30.86	33.33
C	33.33	39.51	33.33

8.4 Which normalisation unit should I use?

Well, there's a lot of debate around this, so let's look at our total normalised values for each replicate.

8.4.1 RPKM

Gene	Replicate 1 RPKM	Replicate 2 RPKM	Replicate 3 RPKM
A	1.43	1.33	1.42
B	1.43	1.39	1.42
C	1.43	1.78	1.42
Total RPKM	**4.29**	**4.50**	**4.25**

8.4.2 TPM

Gene	Replicate 1	Replicate 2	Replicate 3
A	33.33	29.63	33.33
B	33.33	30.86	33.33
C	33.33	39.51	33.33
Total TPM	**100**	**100**	**100**

Notice that that total TPM value for each of the replicates is the same. This is not true for RPKM and FPKM where the total values differ. With TPM, having the same total value for each replicate makes it easier to compare the proportion of reads mapping to each gene across replicates (although you shouldn't really compare across experiments). With RPKM and FPKM, the differing total values make it much harder to compare replicates.