Stat 115 Lab 5

RNA-seq

Matt Ploenzke

February 20/22, 2018

Roadmap

- RNA-seq analysis workflow
  - Read alignment
  - Differential expression
- python practice

Anouncements

- Do not share your code with fellow students via github, please make sure your repo is private.
- Double check you have access to Odyssey and /n/stat115/. You will need it for homework 3 (due 3/4).
- Start early on homework 3!
- Email me if you did not yet receive a grade for homework 1.

RNA-seq analysis workflow

Fragment alignment

- https://academic.oup.com/bioinformatics/article/29/1/15/272537
  - Suffix array (SA) approach to leverage speed of binary searching
  - Requires an index (the SA) to be created prior to aligning
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5600148/
  - Pseudo-alignment concept to rapidly align reads
  - Reads are classified according to their *fragment equivalence class*
  - (Optionally) removes sequencing biases and computes coverage via likelihood maximization
  - Similar in concept and result to https://www.nature.com/articles/nbt.3519

Star alignment

- Needs to be run on Odyssey HPC
- Takes several hours to complete (given an index)
- How do we do that again?

```
#!/bin/bash
#SBATCH -n 10           #Number of cores
#SBATCH -t 1440         #Runtime in minutes
#SBATCH -p general      #Partition
#SBATCH --contiguous    #Cores on same infiniband switch
#SBATCH --mem=100000    #Total memory
#SBATCH --mail-type=END # Email
#SBATCH --mail-user=YOUR_EMAIL

module load gcc/4.8.2-fasrc01 STAR/2.5.0c-fasrc02
```

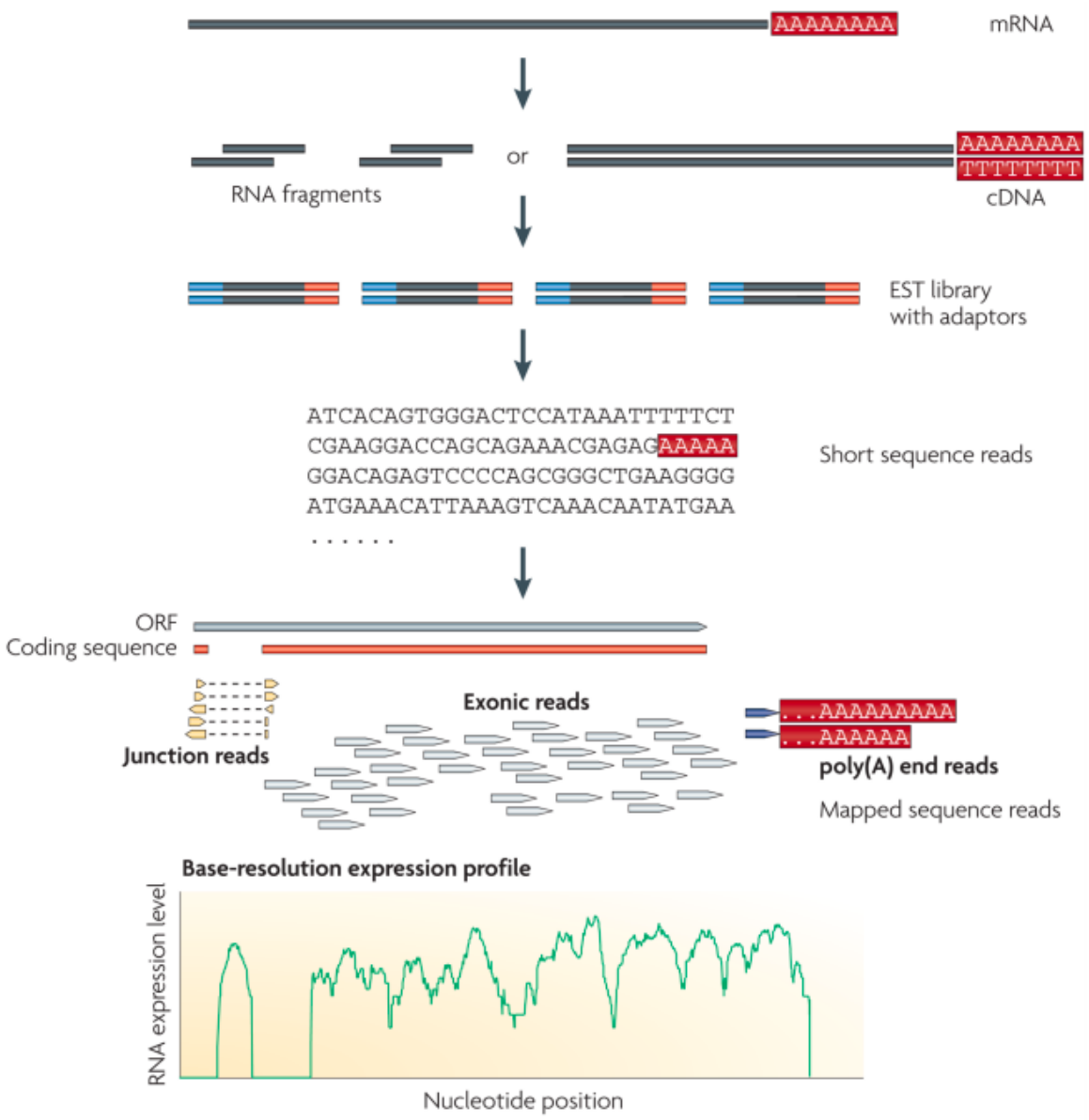


Figure 1: Wang, Z., Gerstein, M., & Snyder, M. (2009). RNA-Seq: a revolutionary tool for transcriptomics. Nature reviews genetics, 10(1), 57.

```
STAR --genomeDir $GENOME \
     --readFilesIn $FASTQ1 $FASTQ2 \
     --outFileNamePrefix $OUTDIR/ \
     --outSAMprimaryFlag AllBestScore \
     --outSAMtype BAM SortedByCoordinate \
     --runThreadN 10 \
     --alignEndsType EndToEnd

module purge
```

```
sbatch STARalignment.sh
```

Salmon alignment

- Much quicker but will still be run on Odyssey
- First step is to create an index, then align

```
# shebang and cluster specs here
module load salmon

salmon index -t $TRANSCRIPTOME -i $INDEX

module purge
```

```
sbatch createSalmonIndex.sh
```

```
# shebang and cluster specs here
module load salmon

salmon quant -i $INDEX \
             -l A \
             -1 $FASTQ/ENCFF500PDO_sub.fastq\
             -2 $FASTQ/ENCFF708KQE_sub.fastq \
             -o $OUT/ \
             --numBootstraps 100 \
             -p 8 \
             --gcBias

module purge
```

```
sbatch Salmonalignment.sh
```

Running your own alignment

- Use the commands given earlier to run your own alignment in Homework 3
- Recall the idea of variables in bash to simplify your code
  - `export VARIABLE=file/path`
- Batch submit your job to the cluster, outputting files to your local folder
- Upon completion, you may receive an email
- View the slurm- file and the log file (located in the output directory)
- Abundance estimates are given in the `.sf` file. Copy these to your local directory (e.g. `scp` or fileZilla) for downstream analysis

Differential expression

- Given the abundance estimates, we now wish to determine which genes/transcripts are differentially expressed between condition.
- Just like in microarrays, we need to create a design matrix specifying what we wish to estimate
- We will be making use of the `DESeq2` packages. Install this via bioconductor.

```
source("https://bioconductor.org/biocLite.R")
biocLite("DESeq2")
```

Differential expression

- Load the package.
- Create a data frame with columns as specified below containing the necessary information for the design matrix (sampleName, fileName, condition).

```
library(DESeq2)
files <- grep("sf",list.files("data"),value=TRUE)
condition <- c("4oh", "4oh", "4oh", "ctrl", "ctrl", "ctrl")
names <- c("4oh1", "4oh2", "4oh3", "ctrl1", "ctrl2", "ctrl3")
sampleTable <- data.frame(sampleName = files, fileName = files, condition = condition)
```

Differential expression

- Utilizing the mouse transcriptome as below, extract the transcript id columns and the gene id columns.

```
library(EnsDb.Mmusculus.v79)
txdf <- transcripts(EnsDb.Mmusculus.v79, return.type="DataFrame")
tx2gene <- as.data.frame(txdf[,c("tx_id", "gene_id")])
```

Differential expression

- Now we're going to import the data and format the data for analysis.
- Remove rows with a total row count less than or equal to 1.
- Run the analysis!

```
library(tximport)
txi <- tximport(file.path("data",files), type="salmon", ignoreTxVersion = TRUE, tx2gene = tx2gene)
dds <- DESeqDataSetFromTximport(txi,colData=sampleTable,design=~condition)
dds <- dds[rowSums(counts(dds)) > 1, ]
dds <- DESeq(dds)
```

Differential expression

- Extract the significantly DE genes at the .05 significance level.
- Remove incomplete observations (those with missing column data).
- Find those DE genes significantly up/down regulated with an adjusted p-value below .05. How many are there?

```
res <- results(dds, alpha = 0.05)
res <- res[complete.cases(res),]
res <- res[order(res$padj),]
```

```
upR <- res[(res$padj < 0.05) & (res$log2FoldChange > 0),]
downR <- res[(res$padj < 0.05) & (res$log2FoldChange < 0),]
nrow(upR)
```

```
## [1] 382
```

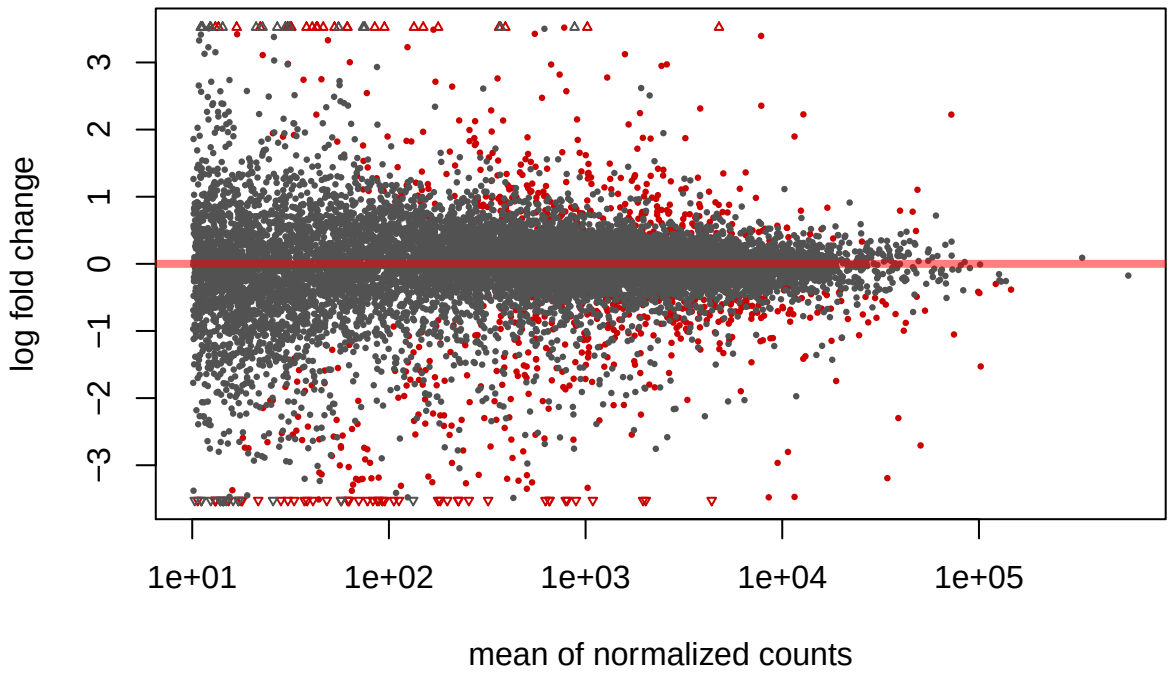
```
nrow(downR)
```

```
## [1] 517
```

Visualizing results

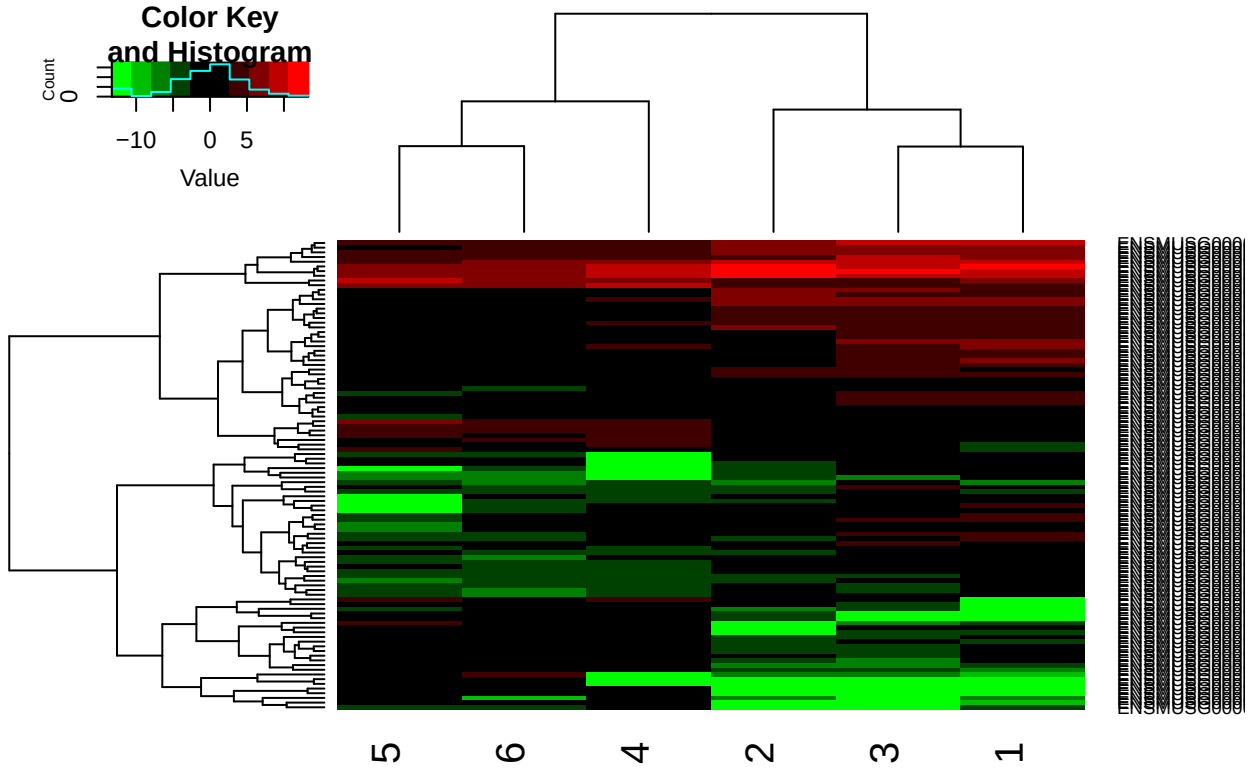
- Which plot should we use to visualize the differential expression? Use it to visualize the results.
- Any other thoughts on how to further visualize the results?

```
plotMA(res)
```




```
absOrdered <- rbind(upR,downR)
absOrdered <- absOrdered[order(abs(absOrdered$log2FoldChange),decreasing = TRUE),]
mostvariable <- log2(txi$abundance[row.names(absOrdered),]+.0001)

library(gplots)
heatmap.2(mostvariable[1:100,],trace="none",col=greenred(10))
```



python practice

Write a python program to merge two paired-end Fastq files (4OH1_5M_a_sub.fastq;4OH1_5M_b_sub.fastq in /n/stat115/2017/HW2/Homework2/FastqData) in to one.

The results should look like below: Sequace ID,sequence a,sequence b,sequence a quality,sequence b
@SRR2994643.15099964 15099964 length=50, CTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA, CCCCCGGGGGGGGGGGGGGGGGC
BBB@BGGGGGGGGGGGFGGGGGFGGGGGD;?/:9??9/999:9::>9?C?

Objectives

1) Read in the two files into python
2) Distinguish quality ID and quality score, and sequence Id and sequence information
3) Match the sequence according to their sequence ID
4) Write out the results in csv files (remember to name your file with .csv suffix so that it can be opened directly by excel)

Solution steps

Possible solutions steps: 1) Define object class to hold read 2) Define tailored `read` function 3) Iterate through file 1 matching in file 2 4) Write results

Class definitions

- Classes are user-defined objects allowing instances of that object to be created

- We may code up specific methods for the class depending on the particular use case
- Once intialized (default behavior), we may utilize these methods
- Here we want to define a class to store the name, sequence, and quality score, along with a method to format the result

```
class myClass(object):

    def __init__(self, option):
        self.option = option

    def myFunction(self, option2="."):
        self.option = self.option.split(option2)
        return self
```

Class definitions

- Define a class called Fastq which is initialized with a name, sequence, and quality score slot.
- Add a trimDetail method which removes the last column of the input row.

```
class Fastq(object):

def __init__(self, name, seq, qual):
    self.name = name
    self.seq = seq
    self.qual = qual

def trimDetail(self, separator=" "):
    self.temp = self.name.split(separator)
    del(self.temp[-1])
    return separator.join(self.temp)
```

Function definition

- We saw function definitions in Lab 2
- Write a function called readFastq to open a file in reading mode, strip off each column in the line, and return a Fastq object

```
def readFastq(infile):
    with open(infile,mode="r") as f:
        while True:
            name = f.readline().strip()
            if not name:
                break
            seq = f.readline().strip()
            name2 = f.readline().strip()
            qual = f.readline().strip()
            yield Fastq(name, seq, qual)
```

Main code evaulation

- Now write code to read in the two files in the data folder. To do so, create variables to store the file names, create dictionaries for each file, read the files in with your readFastq function. Lastly, define

two booleans: `s1_finished`, `s2_finished` and assign them as false.

```
if __name__ == "__main__":
import csv
in1 = '4OH1_5M_a_sub.fastq'
in2 = '4OH1_5M_b_sub.fastq'
seq1_dict = {}
seq2_dict = {}
seq1 = readFastq('data/' + in1)
seq2 = readFastq('data/' + in2)
s1_finished = False
s2_finished = False
```

Iterate through the reads

- We now need to iterate the files and collect all the information we need from each. These will be output to csv.
- Write code to check the value of the boolean variables and evaluate while both are still false. At each step, check if a subsequent element in `seq1` exists, and if not, assign a true value to s1_finished. Do the same for `seq2`. If a subsequent element exists, assign the value to `s1`, `s2` respectively.
- If a value was assigned to `s1`, assign it to the `seq1_dict` and name the element `s1.trimDetail()` making use of your defined method from earlier. Do the same for `s2`.

```
with open(in1 + '.csv', "w") as reads1:
  fieldnames = ['SequenceID', 'SequenceA', 'SequenceB','QualityA','QualityB']
  writer = csv.DictWriter(reads1, fieldnames=fieldnames)
  writer.writeheader()
  while not (s1_finished and s2_finished):
      # iterate through reads (rows)
      try:
          s1 = seq1.next()
      except:
          s1_finished = True
      try:
          s2 = seq2.next()
      except:
          s2_finished = True

      # add next read to fastq1 dictionary and fastq2 dictionary
      if not s1_finished:
          seq1_dict[s1.trimDetail()] = s1

      if not s2_finished:
          seq2_dict[s2.trimDetail()] = s2
```

Write results

- The last step is to write the results. Code provided below.
- Write code to evaluate if `s1_finished=False` and `s1.trimDetail() in seq2_dict`, and if evaluated writes a row to our output file filling in values for each of the columns defined earlier. `pop` off this added row from each of the dictionaries.

- Do the same (but flip!) for s2.

```
# compare and write out reads in both dictionaries
        if not s1_finished and s1.trimDetail() in seq2_dict:
            writer.writerow({'SequenceID':seq1_dict[s1.trimDetail()].name,
                'SequenceA':seq1_dict[s1.trimDetail()].seq,
                'SequenceB':seq2_dict[s1.trimDetail()].seq,
                'QualityA':seq1_dict[s1.trimDetail()].qual,
                'QualityB':seq2_dict[s1.trimDetail()].qual})
            seq1_dict.pop(s1.trimDetail())
            seq2_dict.pop(s1.trimDetail())

        if not s2_finished and s2.trimDetail() in seq1_dict:
            writer.writerow({'SequenceID':seq1_dict[s2.trimDetail()].name,
                'SequenceA':seq1_dict[s2.trimDetail()].seq,
                'SequenceB':seq2_dict[s2.trimDetail()].seq,
                'QualityA':seq1_dict[s2.trimDetail()].qual,
                'QualityB':seq2_dict[s2.trimDetail()].qual})
            seq1_dict.pop(s2.trimDetail())
            seq2_dict.pop(s2.trimDetail())
```

Full solution

```
# define a class to hold each read
class Fastq(object):
    def __init__(self, name, seq, qual):
        self.name = name
        self.seq = seq
        self.qual = qual
    def trimDetail(self, separator=" "):
        self.temp = self.name.split(separator)
        del(self.temp[-1])
        return separator.join(self.temp)
def readFastq(infile):
    with open(infile,mode="r") as f:
        while True:
            name = f.readline().strip()
            if not name:
                break
            seq = f.readline().strip()
            name2 = f.readline().strip()
            qual = f.readline().strip()
            yield Fastq(name, seq, qual)
if __name__ == "__main__":
    import csv
    in1 = '4OH1_5M_a_sub.fastq'
    in2 = '4OH1_5M_b_sub.fastq'
    seq1_dict = {}
    seq2_dict = {}
    seq1 = readFastq('data/' + in1)
    seq2 = readFastq('data/' + in2)
    s1_finished = False
    s2_finished = False
```

```
    with open(in1 + '.csv', "w") as reads1:
        fieldnames = ['SequenceID', 'SequenceA', 'SequenceB','QualityA','QualityB']
        writer = csv.DictWriter(reads1, fieldnames=fieldnames)
        writer.writeheader()
        while not (s1_finished and s2_finished):
            # iterate through reads (rows)
            try:
                s1 = seq1.next()
            except:
                s1_finished = True
            try:
                s2 = seq2.next()
            except:
                s2_finished = True
            # add next read to fastq1 dictionary and fastq2 dictionary
            if not s1_finished:
                seq1_dict[s1.trimDetail()] = s1
            if not s2_finished:
                seq2_dict[s2.trimDetail()] = s2
            # compare and write out reads in both dictionaries
            if not s1_finished and s1.trimDetail() in seq2_dict:
                writer.writerow({'SequenceID':seq1_dict[s1.trimDetail()].name,
                    'SequenceA':seq1_dict[s1.trimDetail()].seq,
                    'SequenceB':seq2_dict[s1.trimDetail()].seq,
                    'QualityA':seq1_dict[s1.trimDetail()].qual,
                    'QualityB':seq2_dict[s1.trimDetail()].qual})
                seq1_dict.pop(s1.trimDetail())
                seq2_dict.pop(s1.trimDetail())
            if not s2_finished and s2.trimDetail() in seq1_dict:
                writer.writerow({'SequenceID':seq1_dict[s2.trimDetail()].name,
                    'SequenceA':seq1_dict[s2.trimDetail()].seq,
                    'SequenceB':seq2_dict[s2.trimDetail()].seq,
                    'QualityA':seq1_dict[s2.trimDetail()].qual,
                    'QualityB':seq2_dict[s2.trimDetail()].qual})
                seq1_dict.pop(s2.trimDetail())
                seq2_dict.pop(s2.trimDetail())
```