Exercise Sheet 2: Genome Sequencing

Introduction to Bioinformatics (MVE510)

Autumn, 2017

Introduction

In this computer exercise, we take a closer look at genome sequencing. We will analyze three datasets, each corresponding to a sequencing experiment of a single bacterial genome. The aim of the exercise is to get experience of all the major step in the analysis. We will start with **quality assessment and preprocessing of the reads**. After that, we will **map the reads to a reference genome** and visualiz them in a software called Integrative Genome Viewer (IGV). At last, the data will be loaded into R, where we will analyz it and **identify single nucleotide polymorphisms (SNPs)**.

An important aim of this exercise is to provide a first **introduction to Linux**, which is the typical computer environment used to perform bioinformatics analysis. Since you have not previously been working extensively in the Linux environment, the first part of the computer exercise comes with detailed direction on how to run commands in Linux. If you have issues or questions or issues regarding Linux, don't hesitate to ask the assistants!

The computer exercise should be performed in groups of two students but working alone is also acceptable. The exercise will be examined through a written report that describes the different steps you took, the generated figures and your answers to the questions posed throughout the exercise. All written code should also be added as an appendix. The reports should be handed in via the course homepage in PingPong latest **November 24th**. To submit your report, you must first have created a project group on the PingPong course page, even if you are working alone. Only project groups can submit reports through PingPong.

Data description

In this computer exercise you take the role of an employee at the Swedish Center for Disease Control (Folkhälsomyndigheten). In a recent investigation of disease-causing bacteria, you have encountered three suspicious isolates collected from vegetables in a supermarket. The isolates are identified to be strains of *Escherichia coli* and you suspect that this may be the start of an outbreak. You therefore want to make sure that this particular form of *Escherichia coli* is treatable with antibiotics. Resistance to antibiotics is caused by mutations in the bacterial genome and you therefore order a whole-genome sequencing of all three isolates at the local sequencing center. The sequencing was done using the Illumina sequencing platform at a read length of 100 nucleotides. The sequencing resulted in approximately 1 million reads for each isolate genome.

Your ultimate aim of the exercise ahead is to **decide whether any of the three isolates contains an *Escherichia coli* that is resistant to antibiotics**. This will be done by processing and analyzing the whole-genome sequencing data and interpret the result. We will start from the very first step, with pre-processing the raw data and end with identifying potential mutations and examine their biological impact.

Step 1 – Preprocessing and sequence alignment

The first part of the computer exercise will done in Linux. This is the environment for which the vast majority of all methods and software used to handle and processes sequencing are developed for. We will run these methods through the Linux shell, which is similar to the command line that is available in Windows.

You can connect to a remote Linux system at Chalmers using your account with the Windows software Bitwise (alternatives are for instance Putty). To login, write `remote11.chalmers.se` in the 'host' field and your CID in the 'username' field. Once you have logged in, a shell is opened which we will use this shell to start various bioinformatical software. Another window is also opened which you can use to transfer files between the Windows and Linux systems using the SFTP protocol (you have a file storage area on both systems). A list of basic commands that can be used in the shell can be found in Table 1 below.

Note that in this part of the exercise we will only work with a small subset of the sequencing data (approximately 10% of the total number of sequence reads). The reason for this is to minimize the waiting time and to be able to perform the analysis within the disk quota of your student accounts. In step 2 of the exercise, we will however use the full dataset.

Table 1: A list of important Linux commands

Linux command	Description
ls	Lists all files in the current directory
cd directory	Changes directory to 'directory'
cd	Changes directory to the home directory
pwd	Prints the full path of the current directory
rm file	Removes 'file'
rm –R directory	Removes directory and all its content (use with caution!)
less file	Shows 'file' on the screen (use 'q' to quit less)
nano file	Opens 'file' in the basic editor nano

Exercise 1: Downloading the sequence files to your Linux account

Log in to your Linux account. After successfully logging in into the server, you will be placed in your home directory. You can, at any point, move back to this directory by simply writing

```
cd
```

`cd` is short for 'change directory' and when no specific directory is given, it takes you to your home directory.

Create a directory `MVE510E2` to store the data files used in the computer exercise. This is done using

```
mkdir MVE510E2
```

where `mkdir` is short for 'make directory'. You can, if you want, use an alternative name for the directory or place it at any other location. Note, however, that the names of directories and files in the Linux environment is case sensitive, which means that 'mve510E2' and 'MVE510E2' are not identical.

Enter the new directory by typing

```
cd MVE510E2
```

Next, the sequence data for the computer exercise needs to be downloaded. We will do that using the command 'wget' which retrieves files available at the web. The files for this exercise are located at a local server here at Chalmers and to **download the files**, type

```
wget http://bioinformatics.math.chalmers.se/courses/MVE510/genome1.fq.gz
```

After the download is complete, the file will be placed in your current directory. The downloaded file is compressed with `gzip` to save space on the hard drive (note: gzip-compression is very common and many programs can read them even without decompression). To **decompress the file**, type

```
gunzip genome1.fq.gz
```

This will generate a new file called `genome1.fq`. To **view the content** of this file, type

```
less genome1.fq
```

Is the file a proper FASTQ-file? Can you identify the different parts?

Repeat the procedure for the two other genomes `genome2.fq.gz` and `genome3.fq.gz`.

Exercise 2: Preprocessing of the data

Our next aim is to **assess the sequence quality** of the downloaded data. This will be done using a software called `fastqc`, which calculates statistics about reads from next generations sequencing, including curves of the quality score. This software is run directly on a fastq file and summarizes the quality of the data. To run `fastqc`, type

```
fastqc genome1.fq
```

The results from `fastqc` will be saved in a named `genome1.fastq.zip` in the same directory (type 'dir' to list all the files).

Transfer the file to your windows account, unzip it and open it in your browser (by double clicking on the file `fastqc_report.html`). What many sequences did the file contain? How are the quality scores distributed over the reads? How is the G/C-content of the reads?

Repeat the quality control for the other two files (`genome2.fq` and `genome3.fq`) and examine the results. How does the quality compare between the samples? Which sample has the worst quality? Are there any other differences?

To remove nucleotides that have a too low quality, the data needs to be processed. This will, in this computer exercise, be done using a software call `fastx`, which can remove reads or part of reads based on their quality scores. The software takes a FASTQ-file, a set of parameters and generates a new FASTQ-file containing only the reads that were passing a pre-defined quality cut-off. `fastx` is a Linux command line program, similarly to `fastqc`.

We will use program `fastq_quality_filter` from the `fastx` suite to filter the reads. This command takes two parameters, the quality score threshold and the minimum percent of nucleotides that needs to be above the score threshold. Thus, any read which does not satisfy these parameters will be removed from the file (i.e. has more than 'minimum percent of nucleotides' with a quality score less than the threshold). We will use a quality score threshold of 30 and set the minimum percent of nucleotides to 80, i.e.

```
fastq_quality_filter -i genome1.fq -o genome1.filtered.fq -q 30 -p 80 -Q64
```

Here, -i specifies the input file and -o the output file. The flag -Q64 sets the encoding of the quality scores used for the FASTQ-files we are using in this exercise.

Process all three genomes. Then rerun `fastqc` on the filtered data files. We can do this in a single step using

```
fastqc genome[123].filtered.fq
```

Do you see any differences? How many sequences were removed from each of the files? Did the read length change?

Optional: Rerun the preprocessing steps using different parameters and view the results in fastqc. What happens if you use more strict or less strict values?

Exercise 3: Mapping the reads to a reference

In order to compare the genomes from the three samples, each of them needs to be aligned to a reference genome. In this exercise, we will use a wild-type reference of *Escherichia* coli called strain K12 MG1655. The sequence of the **reference can be downloaded** using `wget` by typing

```
wget http://bioinformatics.math.chalmers.se/courses/MVE510/reference_Ecoli_K12_MG1655.fasta
```

Alignment of the read against the reference will be done using a software called BWA (stands for Burrows-Wheeler Alignment Tool). BWA works in two stages, where it first finds seeds by using suffix arrays and the Burrow-Wheeler transform. The read is then extended around the seed using the Smith-Waterman algorithm. In order to do find the seeds, we first need to **calculate the index**, consisting of the BWT and the corresponding suffix array for the reference genome. This can be done by typing

```
bwa index reference_Ecoli_K12_MG1655.fasta
```

After the index has been created, BWA can align reads to the reference. Mapping of the reads to the reference can then be done by proving BWA with the reference and the file with the sequence reads.

```
bwa mem reference_Ecoli_K12_MG1655.fasta genome1.filtered.fq > genome1.sam
```

The word 'mem' decides which specific BWA algorithm to use (BWA contains other algorithms not covered by this course). BWA write the resulting file on the standard output (the terminal) and the `> genome1.sam` in the end tells the shell to redirect the results into a file called `genome1.sam`. This file will be in SAM-format, which stands for 'sequence alignment'. Even though SAM-files are rather complex, they can in text-format and **can be viewed**, e.g.

```
less -S genome1.sam
```

(Hint: in the less text viewer, you can navigate with the error keys and quit by pressing `d`).

Does any of the SAM-file make sense to you? Do you see any relevant information?

Exercise 4: Viewing the results in Integrative Genome Viewer (IGV)

Integrative Genome Viewer (IGV) is a software for visualizing results from read mapping. IGV is installed in the Windows system. Before you transfer our SAM-files to Windows systems they need to be converted to a format that IGV can read. This is done with a command line program called `samtools` by **converting the text format into a more efficient and sorted binary BAM-file** (the 'B' in BAM stands for 'binary') which contains the reads in the order of their mapped starting position on the genome. We **create an additional index** which IGV uses for fast access of information in the BAM file. Write

```
samtools sort genome1.sam > genome1.sorted.bam
samtools index genome1.sorted.bam
```

Transfer the resulting files `genome1.sorted.bam` and `genome1.sorted.bam.bai` to your Windows account and start IGV. Before you load the files into IGV you need to load an annotation file corresponding to the reference genome we used. The annotation files are available at http://bioinformatics.math.chalmers.se/courses/MVE510/K12.genome.zip (it can be downloaded using a web browser). Save *K12.genome.zip* on your Windows account at the same place as the previous files and unzip it. In IGV, **load first the annotation file `K12.genome`** by selecting Genomes->Load Genome Form File. After that you can **load your alignment** by selecting File->Load from File and then select `genome1.sorted.bam`.

Describe what you see. Are the reads organized in any particular way? Can you say anything about the coverage? Do you see any sequencing errors? Note that you need to zoom in before any information is shown (using the "+"-sign in the top right corner).

Step 2 – Identification of mutations

In this part of the exercise we will focus on identifying mutations present in the three bacterial isolates. This will require analysis of each position in the three genomes. For this analysis, we will use R on the Windows platform. The alignment done in the previous step of the computer exercise was done on a subset to ensure small files and fast computations (and thus as little waiting time as possible). In the rest of the exercise we will instead use the full data.

Exercise 5: Loading the data into R

In this part of the exercise we will, however, work with data from the entire dataset. This data is available in R datafiles generated directly from the SAM-files produced by BWA (identical to what you did previously). Use a web browser to download the complete datafiles from http://bioinformatics.math.chalmers.se/courses/MVE510/ to a directory in your Windows account. Start Rstudio and change its directory to the place where you downloaded the files. You can then **load the R data files** using the `load` command (or open in from the files panel in RStudio).

```
load("genome1.rdata")
```

Repeat the process for the two other data files. One loaded, you will get three new objects (you can check this by using `ls()`) called genome1, genome2 and genome3. Familiarize yourself with these object. How are they organized? Note that the object are quite large so the commands `class`, `dim`, `length` and `head` may be useful here.

An important tip for this part of the exercise

The three data objects are quite large which means that some of the calculations done in this part of the exercise make take some time. This is a very common complication when working with big and complex data. A *strong* recommendation is therefore to create a subset of the first genome which you can use to test the code you write. For example, write

```
genome1.subset <- genome1[1:1000,]
ref.subset <- reference[1:1000]
```

to store the first 1000 positions in of the first genome in the variable `genome1.subset` and the first 1000 positions of the reference in `ref.subset`. Processing genome1.subset will be much faster than the full genome since it only contains the first 1,000 positions. This makes it suitable to verify that the code that you are writing is working properly. Once your code work, you can use it on the complete data.

Exercise 6: Coverage

The positional coverage is the total number of sequence reads covering a specific genomic position. For genome 1, **use `apply` together with `sum` to calculate the coverage for each position**. Add the coverage as a new colum `cov` to the genome1 `data.frame`. **Calculate also the mean coverage** over the entire genome.

Does the coverage vary? Why? What is the maximum coverage? Plot the results for a region covering 1,000 positions. Why is it good to have a high coverage?

Exercise 7: Error rate

Sequencing errors are common, even after the data has been quality assessed. Calculate, for each position, the proportion of reads that does not match the reference in **genome1**. How many position has at least one read with a mismatch? Visualize the proportion of mismatching reads over a region covering 1,000 positions.

Hint: This exercises can be solved by a **for-loop** looping over the number of positions in the genome. Remember that the **for-loop** in R has the following syntax

```
# Syntax for for-loop in R
genome1.length <- nrow(genome1)  # length of genome1

# Allocate an empty vector for matches
genome1$matches <- vector(length=genome1.length, mode="double")

for (pos in 1:genome1.length){
  # Picks the number of reads that has the same nucleotide as the reference
  genome1$matches[pos] <- genome1[pos,reference[pos]]
}
```

The code above will, for each position, pick out the number of reads that has the same nucleotide as the reference. Not that this is only possible due to the rownames of genome1 – for each position the column with the same name as the reference will be used (use `rownames` on the `genome1` data object to see what they are).

Optional: Instead of a loop, you can also use `apply` with a custom function. Often, using `apply` instead of loops runs much faster in R and creates more compact code. The following code is equivalent to the loop above.

```
genome1$matches <- apply(genome1, 1, function(row) row[reference[row[1]]])
```

Exercise 8: A test for single nucleotide polymorphisms (SNPs)

Next, our aim is to **identify positions for which the observed SNPs are caused by differences between the genomes and not sequencing errors**. This means, that the reads carrying the SNP will differ from the reference genome they are aligned to. We will use a statistical test to detect these positions. The test will assume that sequencing errors appear randomly and independently int the reads and nucleotides. Let the random variable Y_i be the number of matches at position i, i.e. the number of reads that have the same nucleotide as the reference. Furthermore, let N_i be the coverage at and p_i be the probability for observing a nucleotide different that the reference at position i. For positions where there are no mutations, we expect p_i to be small describing only the sequencing errors (p_{error}). For positions where we have a mutation, p_i is instead expected to be large (close to 1 since bacteria are haploid). We will therefore use a test to assess if the null hypothesis

$$H_0 : p_i = p_{\text{error}}$$

versus the alternative hypothesis

$$H_1 : p_i > p_{\text{error}}$$

Under the assumption that H_0 is true, Y_i can be shown to follow a binomial distribution with parameters p_{error} and N_i.

Why is this the case and which assumptions are necessary for this to be true?

We can then, based on our observed data y_i test H_0 against H_1 by calculating the p-value as

$$pvalue = \text{Prob}\left(Y_i > y_i\right) = \sum_{j=y_i+1}^{N_i} p_{Y_j}\left(j\right).$$

This test is called a binomial test and is implemented in R in the function `binom.test`. Use the R help to **read about the `binom.test` function and the parameters** it requires.

Implement then an R-function that takes the number of mismatches and the coverage at a specific position and returns a p-value according to the test above. We assume that the sequencing error probability p_{error} is 5%.

Hint: Be sure to set the correct alternative hypothesis *H_1* in `binom.test`. You should also take care that your function can handle positions which have a coverage of zero (since such positions are present in the data). An **if**-statement in the beginning of your function can be used to handle such positions.

Exercise 9: Screening the genome for SNPs

Apply the function you wrote in the previous exercise to each position in the genome. Again, this can easily be solved using a **for-loop** (or more efficiently in using the apply function). Save the output of your test function (the p-values) in a columns called `pval` in the genome `data.frame`. After you are done, use `order` to **identify the most significant p-values**.

Are there any positions that show evidence of mutations? What is a good p-value cut-off for selecting significant positions? Is there any risk of setting the p-value cut-off to high?

Repeat the analysis for all three genomes. Which of the genomes has the highest number of significant SNPs?

Note that this analysis may take some time (~10 minutes per genome) so make sure that it works properly before you apply it to the full genome.

Exercise 10: Interpretation of the SNPs – what do they mean?

Do any of the isolates carry any mutation that makes them resistant to antibiotics? To answer this question for each genome, we **determine where the three most significant SNPs in each isolate are located**. This can be done by first looking up the reference genome at the NCBI GenBank database at https://www.ncbi.nlm.nih.gov/nuccore/NC_000913.3. This page shows the full annotation of the reference genome we have used. Once you have found out what gene is located at the position of the mutation, you can click on the corresponding GeneID (line that says `/db_xref=GeneID:XXXXX`). At the new page, you will find a genome browser showing the annotations. By zooming in to 100%, you can view both the DNA and corresponding amino acid sequence. The easiest way to get the codon where the mutation is located, calculate the difference between the starting point of the gene and the location of the mutation and divide that by three. Be aware that these pages contains a lot of information so take your time.

One you know about the gene, the codon that is mutated and its alternative codon, use PubMed (https://www.ncbi.nlm.nih.gov/pubmed) to **search for relevant literature**.

Can you conclude that any of the isolates are resistant to antibiotics? Which isolate and what kind of antibiotic?