NGS Project

scFv analysis script guide

Last update: July 21, 2016

1. Table of Contents

2. Introduction and Scope

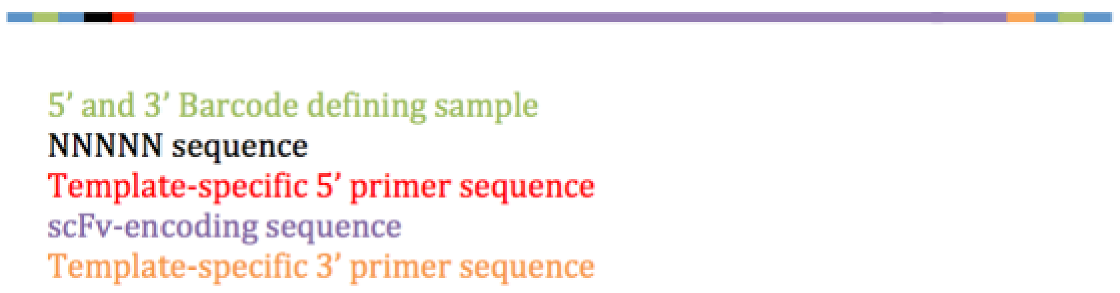
This section will lay out the basic aspects of the project, reiterating the expected data and what the scripts are designed to achieve.

The immediate goal here is to familiarize the users with the assumptions that were taken while developing the tool, as well as to introduce reasoning behind some of the design choices based on the underlying data.

2.1. Explaining the data

The underlying data for this project comes from Illumina MiSeq platform. While thorough review of this platform is out of the scope of this document, a brief introduction will be given in this subsection. For more information, please refer to the user guide for the Illumina MiSeq equipment that is relevant for your experiments.

The scripts developed for this project are designed to work with paired reads from single chain variable fragments (scFv), that are sequenced according to the following scheme.


5' and 3' Barcode defining sample
NNNNN sequence
Template-specific 5' primer sequence
scFv-encoding sequence
Template-specific 3' primer sequence

The output of sequencing is a pair of FASTQ files that are named with a prefix that refers to the particular sample run and suffixed as _1 and _2 to indicate the the 5' (i.e. *forward*) and 3' (*reverse*) reads.

FASTQ file format is originally developed by Wellcome Trust Sanger Institute, and contains a FASTA sequence together with some meta data such as sequencing instrument and quality information. A FASTQ file is composed of entries with are spread over four lines: sequence identifier, sequence, quality score identifier and sequence quality score. See example below:

```
@EAS139:136:FC706VJ:2:5:1000:12850 1:Y:18:ATCACG
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
+
BBBBCCCC?<A?BC?7@@???????DBBA@@@@A@@
```

The quality score line denotes the probability of an erroneous basecalling for each position in the sequence, more precisely put $Q = -10 \log10(p)$. The values are encoded in ASCII base 33 (see Appendix A).

Given several million entries in each file, the FASTQ files are typically rather large (a couple GBs) and more often than not they are compressed with a text compression algorithm like gzip or bzip. These files can be recognized by the .gz or .bz2 extension.

2.2. Aim

The ultimate goal of this project was to streamline the handling and analysis of the NGS-data originating from scFv sequencing. Specifically, to create tools for comparison of the sequences in different samples and evaluation of the selection procedure.

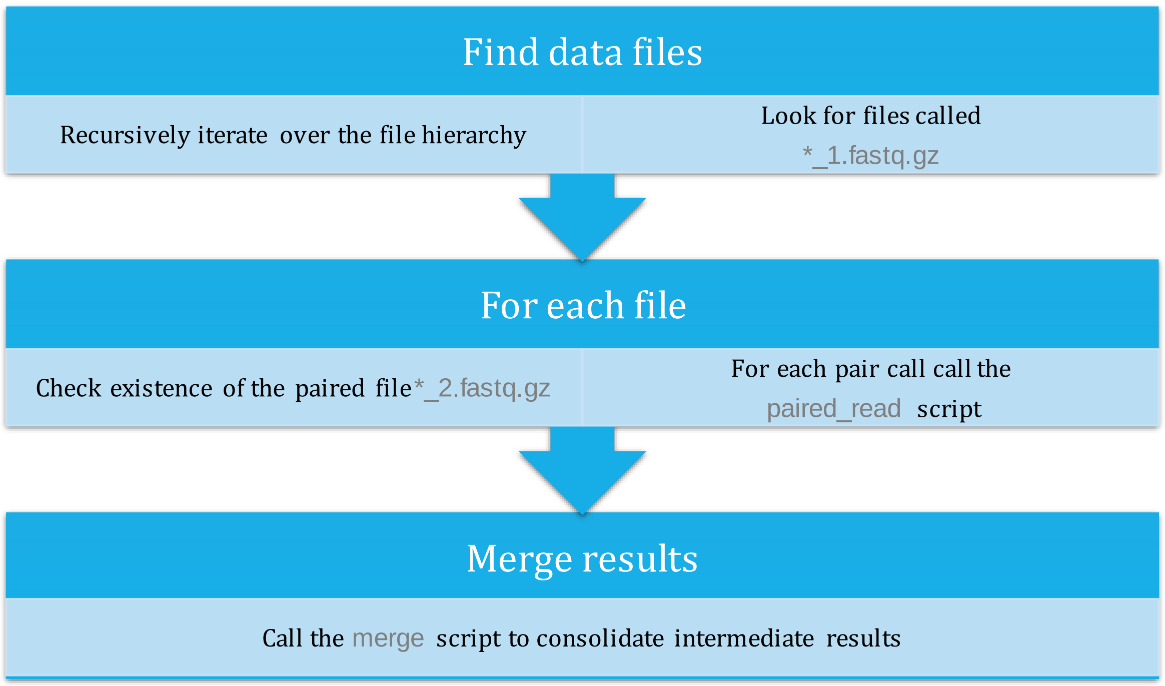
As such, the scripts that were developed for this project will return a consolidated table of sequences, together with their frequency in each sample.

3. Workflow

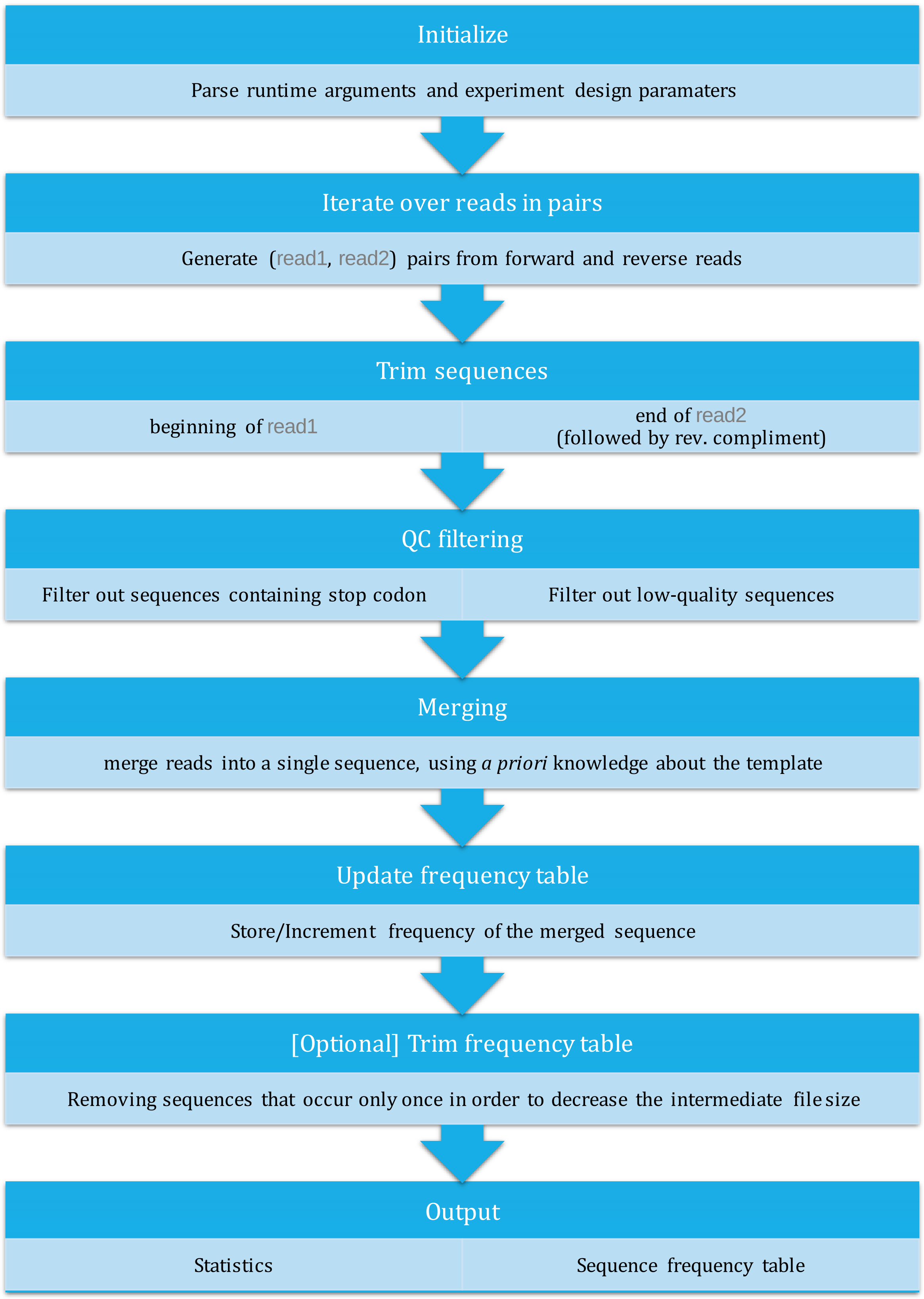
The workflow adopted is divided into three modules, which are presented here. This division of workflow allow for modular development and testing, as well as serving a logical structure to the analysis.

3.1. Main script

File system interface is written as a BASH script, intended to run on a server or a cluster. The idea is to find the data files, and start a child process to process each file pair. Once all child processes complete, a final process is triggered to combine the intermediate results originating from each of the read pairs, these are assumed to be forward and reverse reads, in that order.


Find data files
Recursively iterate over the file hierarchy
Look for files called *_1.fastq.gz
For each file
Check existence of the paired file*_2.fastq.gz
For each pair call call the paired_read script
Merge results
Call the merge script to consolidate intermediate results

3.2. Processing paired reads

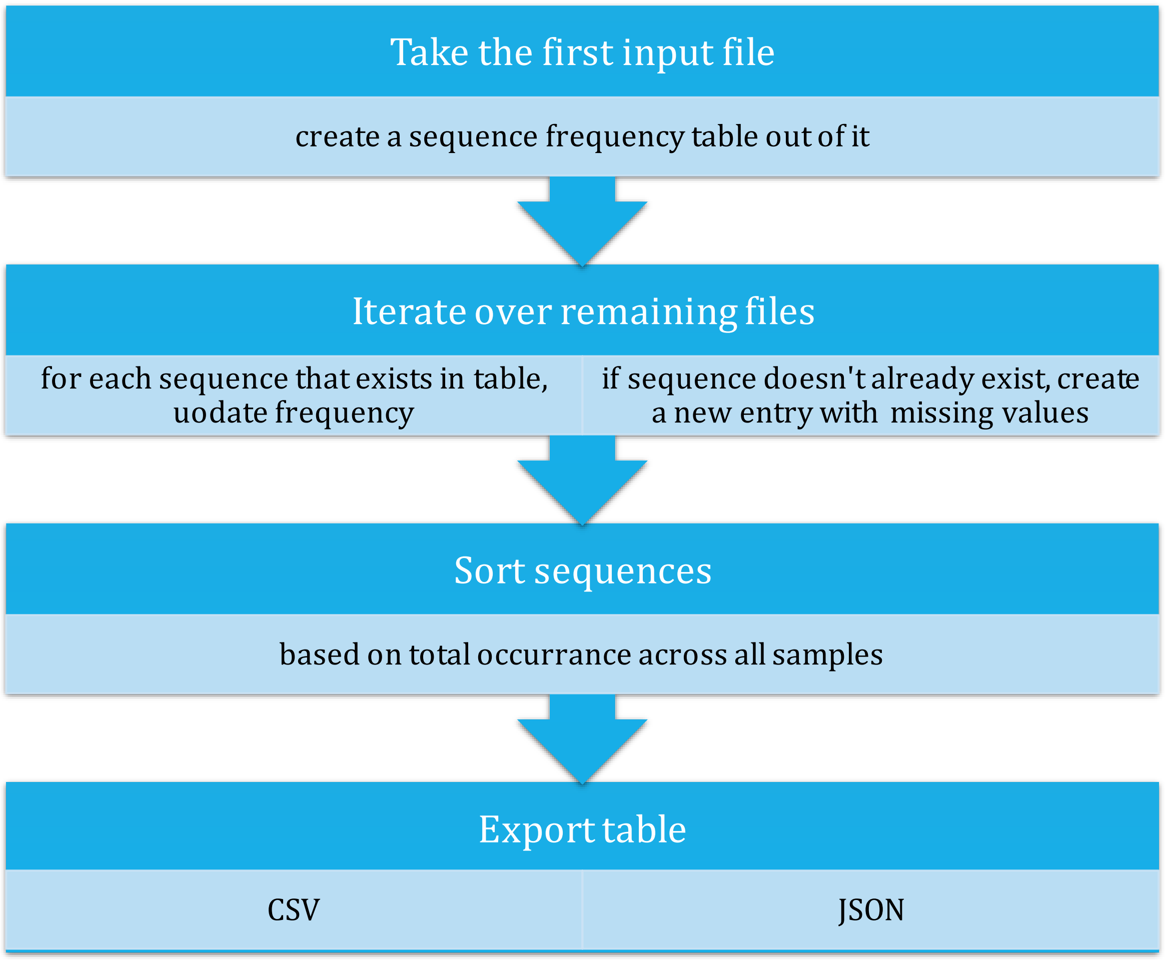

Initialize
Parse runtime arguments and experiment design paramaters
Iterate over reads in pairs
Generate (read1, read2) pairs from forward and reverse reads
Trim sequences
beginning of read1
end of read2 (followed by rev. compliment)
QC filtering
Filter out sequences containing stop codon
Filter out low-quality sequences
Merging
merge reads into a single sequence, using a priori knowledge about the template
Update frequency table
Store/Increment frequency of the merged sequence
[Optional] Trim frequency table
Removing sequences that occur only once in order to decrease the intermediate file size
Output
Statistics
Sequence frequency table

paired_read script is written in Python language. While capable of processing only forward reads as well, this script was designed to process a pair of files containing forward and reverse reads, corresponding to the same original sequence. Furthermore, the script assumes that there are equally many sequences in the files and that they are ordered accordingly. These assumptions generally do hold for Illumina NGS data, thus requiring no additional work by the end user. It is, however, usually a good idea to keep the underlying assumptions in mind.

The output of this script is two files, an intermediate result file containing the occurrence of sequences in that sample (called seqdata_paired.json) and a summary file (called seqtable.out) containing the 100 most common sequence found in that sample.

3.3. Consolidating intermediate results

When each sample folder is processed and frequency tables are created, a secondary Python script, called jsonmerge, is run in order to consolidate the frequency tables. This script takes a number of frequency tables in JSON format as a command-line argument and consolidates these into a single large table.


Take the first input file
create a sequence frequency table out of it
Iterate over remaining files
for each sequence that exists in table, uodate frequency
if sequence doesn't already exist, create a new entry with missing values
Sort sequences
based on total occurrance across all samples
Export table
CSV
JSON

4. Experiment Design

This section contains details regarding the specifics of the algorithms, with respect to the experiment design adopted. The scripts explained in the previous section expect a user-defined experiment design file, written in YAML syntax (see Appendix B).

YAML is a human-readable file format that is commonly used for configuration files due to its simple syntax. The basic idea is to organize files based on line indentation. Below are the five subchapters of the experiment design file.

4.1. Trimming sequences

Sequences may contain artefacts from the primers or "barcodes", that should be removed before doing any sequence comparison. Furthermore, since the read quality varies over the course of the sequence. Thus, it may be beneficial to trim parts of the sequence where no variability is introduced, in order to minimize the number of false negatives by a read error.

For instance, no sequence variability was expected in the CDRL1 and CDRL2 domains of the dataset, which allowed us to trim the second half of the 3' read (corresponding to the V_L side) to avoid any significant read errors giving rise to original and unexpected sequences.

The trimming procedure is defined by the following piece in the experiment design file:

```
# Trimming sequences, ranges given in nucleotide indices
# A single value refers to a start index to the end, two values indicate start and stop
coordinates
Trim:
  fwdread: [27,null]
  revread: [21,150]
```

In the example above, the user has defined the region starting from the 27th base until the end of 5' read (read1), as well as the region between 21st and 150th bases on the 3' read (read2) to be considered as the paired sequences. In the test dataset the first 9 amino acids on read1 were trimmed to allow the sequence to start with the CDRH1. Similarly, half of read2 was trimmed out, due to the decreasing read quality in that region.

It is however important to remember to add any piece sequence that was trimmed out from the middle (end of V_H or beginning of V_L) back into the sequence using the stitching parameters (see Section 4.4)

4.2. Defining CDR regions

In the context of this study large portions of the sequence are expected to be highly conserved, essentially identical to the framework chosen. In that sense it necessary for intended users to indicate where in the sequence the variability is expected to be. This is also important in terms of quality control filtering for the sequences.

Similar to the trimming procedure, the coordinates for the CDR regions are given in indices of base pairs. Unlike the previous step however it is necessary to indicate the read direction associated with any given region, see example below. In the case below, the first region is named "cdrh1" and corresponds to the subsequence between the 1st and 30th nucleotides, **inclusively**, on read1. This corresponds to the first 10 amino acids on the 5' end of the sequence **after** the trimming step. Similarly, the second region is defined between 23rd - 33rd and the third region is defined between 69th – 87th amino acids. In this scenario, the actual CDR region windows were are taken with 1 AA margin on either side.

```
# CDR regions, given as [DIRECTION, START, STOP]
# start and stop values are given in nucleotide indices
# INCLUDING +-1 AA (3 bp) margins
# Add/remove regions as necessary
Regions:
  cdrh1: [5,0,30]
  cdrh2: [5,69,99]
  cdrh3: [5,207,261]
  cdrl3: [3,99,129]
```

The framework used in the test files introduced variability in CDRH1-3 and CDRL3, thus resulting in 4 variable regions. In other experiments, however, this number can be increased or decreased by adding or removing lines into the Regions section of the experiment design file. Note that while individual regions can be named arbitrarily, the section name Region is pre-set and **should not be changed.**

4.3. Filtering and quality control

Read quality is an important aspect of NGS data and thus tools developed in this project take the read quality into account prior to doing any analysis on the sequences. Based on the test data, two steps of filtering are applied to the sequences.

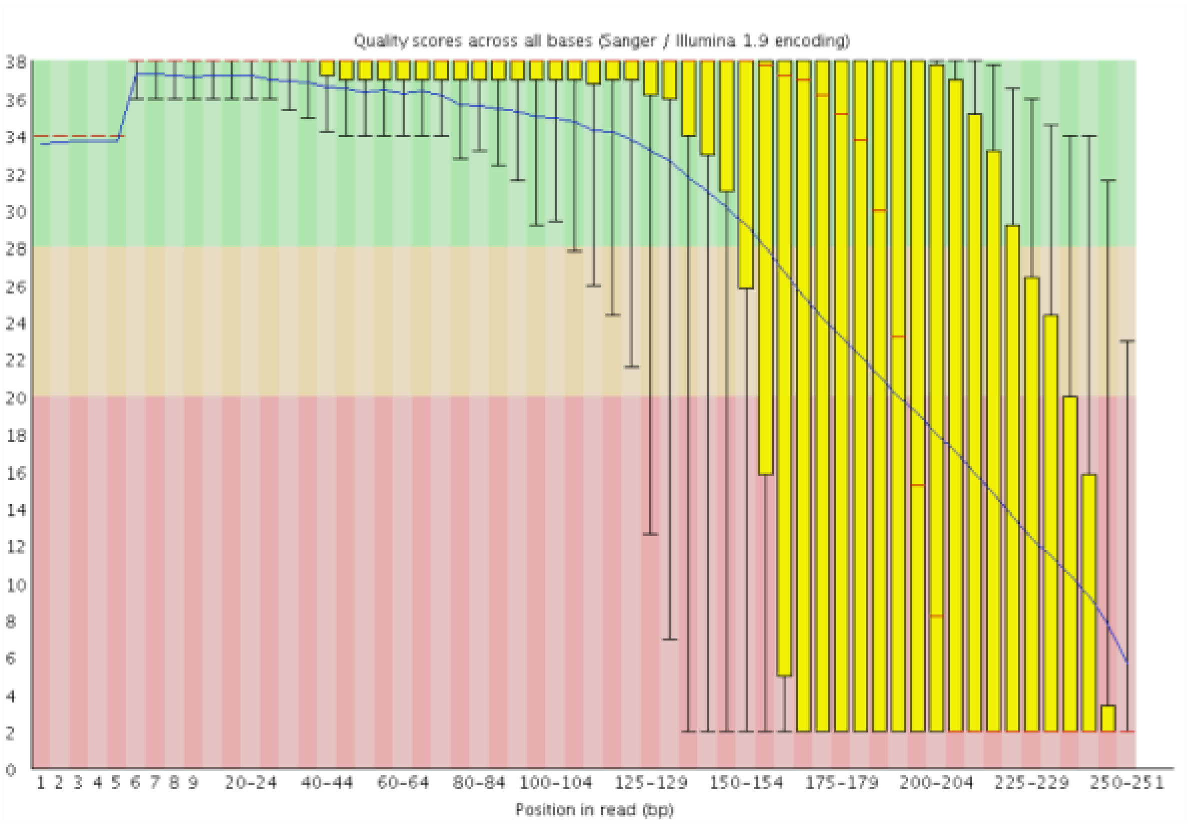
4.3.1. Stop-codon filtering

Some of the sequences may contain stop codons located introduced to control for efficiency of the primer annealing. The basic idea is to have a non-functional single-stranded plasmid, which can then be filtered out in case the annealing process does not favor the primers over the template string.

These sequences are filtered out prior to frequency analysis. Additionally, the occurrence of such sequences decreases with each selection step, based on the test data.

4.3.2. Quality control

The quality of any analysis is reliant on the quality of the input data. Thus in this case the reliability of frequency and enrichment analysis of scFv sequences is dependent on the accuracy of the reads. Furthermore, read quality is a non-linear function dependent on the sequence length. In other words, as the sequencing progresses further along the process the read quality. This is an issue with the sequencing by synthesis (SBS) technique and is often referred to as *loss of base call accuracy* (for more information please see Fuller et al. Nat Biotech 2009).



Three different QC-schemes are implemented in the scripts, based on the proposed workflows presented in a similar tool called pRESTO from Yale School of Medicine. The fundamental idea is the same across all three methods, filtering out sequences that do not hold an average quality level, set by the user. There is however significant difference as to how it's implemented and how strict these three measures of quality are.

The first method is a naïve average of read qualities over the entire sequence. This method is simple and straightforward, however not a very good fit for this type of data, since a large portion of the sequence is expected to have high read quality which can mask much lower quality in smaller regions. For example, consider the following hypothetical read of a 100 base-long sequence below:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25
51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25
76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
25	25	25	25	25	25	25	25	25	25	5	5	5	5	5	5	5	5	5	5	2	2	2	2	2

The first 85 bases have a read quality score of 25, corresponding to an error probability of 0.00316. The quality, however, decreases towards the end of the read, bases 86-95 have a q-score of 5, and the last 5 bases have a quality score of 2. Taking an average q-scores of the entire sequence is approximately 22, which is decent value, corresponding to an error probability of less than 1%.

$$Average(Q) = \frac{85 \times 25 + 10 \times 5 + 5 \times 2}{100} = 21.85$$

However, the error probability is individual for each read. For a whole sequence S, the likely number of errors is the sum of error probabilities:

$$Err(S) \sim 85 \times 0.003 + 10 \times 0.316 + 5 \times 0.631 = 6.583$$

One way to improve this situation is to not look at the entire sequence but rather to the regions that are expected to have any variability, in other words the CDR regions. If we have the same schematic representation of a read, assume that white boxes are framework regions and the red boxes are variable regions of the sequence.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
25	25	25	**25**	**25**	**25**	**25**	**25**	**25**	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	**25**	**25**	**25**	**25**	**25**	**25**	25	25	25	25
51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25
76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
25	25	25	25	25	25	25	25	25	**25**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	5	2	2	2	2	2

By *slicing* the red regions and checking the quality of those regions only, the masking effect of the averaging can be mitigated to some extent. Considering only the red boxes, corresponding to the variable regions; the average quality score is 16.8 which is slightly worse than the standard cut-off value of 20, corresponding to a 1% error rate. With a QC threshold of 20, this sequence would be filtered out as a low-quality sequence.

$$Average(Q) = \frac{13 \times 25 + 9 \times 5}{22} = 16.82$$

While slightly better than an average over the whole sequence, this approach still does not address the issue of accuracy loss towards the end of the sequence. This is an important issue, as the CDRH3 region is typically where the drop in quality occurs. Since the goal is to evaluate how many times a particular sequence occurs in a sample, it is paramount to capture as much as possible of the biological variation and as little as possible of the technical variation.

The final and strictest quality control measure that is implemented, considers each CDR region (defined by the user, see Defining CDR Regions) individually and checks whether or not these regions pass the QC threshold. In the example above, both the first and second regions have average read qualities of 25, while the last region has an average read quality of 7. These values are then compared to the QC threshold value individually. Thus if the threshold is set to 20, this sequence would be filtered out as a low-quality sequence.

The experiment design file contains two fields that are to be set by the user; a threshold value (default: 20) and a filtering method. The filtering methods are given as “ave” for whole sequence average, “combi” for combined CDR average, or “cdr” for individual QC for each CDR region.

```
# Quality control settings
# Alternative values for filtering are "cdr", "combi" or "ave"
# In case of typos or invalid values, filtering method defaults to average quality filtering
Quality:
  threshold: 20
  filtering: cdr
```

It is worth reminding that when using the cdr method for filtering low quality sequences, the pair of sequences are filtered based on the CDR regions defined in the experimental design file. So in the example given Chapter 4.2, the first read in each pair will be filtered considering the 3 regions defined, and the second read is filtered considering the only region defined in the experiment design. The pair is filtered out if either one of the reads fail quality control.

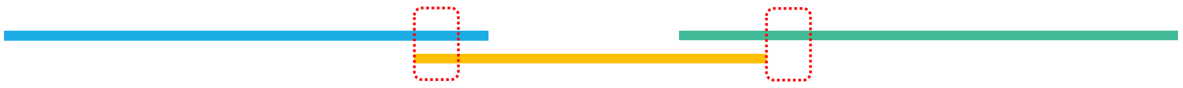
4.4. Stitching together reads

Once the paired reads are filtered for quality control, they are merged together in order to generate a single, complete sequence for frequency analysis. This is done by using *a priori* information the template used to generate the single-chain antibody fragments. This procedure is not straight-forward, however, and there are several pitfalls that require special attention.

In an ideal world, the only potential problem is that CDR regions can vary in length and thus the exact location where (*i*) read1 ends and (*ii*) read2 starts is not certain. These issues are relatively easy to address by linking the reads using a partially overlapping linker string.

```
# Stitching the paired reads
# Linker string is used to join the ends 5' -> 3'
# Anchors are short strings to which the linker will be attached;
# they are expected to be immediately connected to the linker
Stitching:
  linker:
DYWGQGTLVTVSSGGGGSGGGGSGGGGSDIQMTQSPSSLSASVGDRVTITCRASQSISSYLNWYQ
QKPGKAPKLLIYAASSLQS
  f_anchor: DYW
  r_anchor: GVP
```

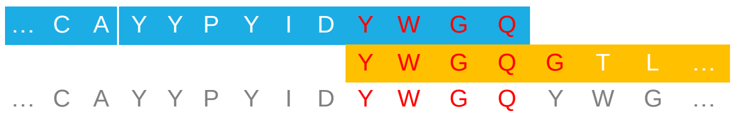
The overlapping subsequences are called “anchor” sequences (shown in dashed red outline) and these are used to locate the start and stop of the linker sequence. Note that the linker string (yellow) is overlapping with read1 (blue) using the first anchor sequence, and continues on read2 (green) **up to but not including** the second anchor sequence. This setup prevents most problems with variable length CDR regions.



We do not live in an ideal world, however, and base calling step of the sequencing is not perfect. That being the case there are several issues that arise with the merging of the sequences.

One such issue is the loss of base calling accuracy towards the end of read1. Given the higher rate of error in this region, it’s likely that first anchor sequence (defined as DYW in the example above) cannot be found in the end of read1. The choice of amino acids for anchor sequence is thus a compromise between type 1 and type 2 errors. A short anchor sequence, such as a single amino acid residue is susceptible to errors if the particular residue occurs somewhere else before the intended stitching location (such as the CDRH3 region), resulting in a sequence that does not really exist.

A longer sequence of 4-5 amino acids, on the other may be problematic since the entire anchor sequence might not exist in the end of read1. This is an issue in the case of a clone with longer CDRH regions. To illustrate this with an example if read1 ends with the sequence CAYYPYIDYWGQ and the anchor sequence is YWGQG the following scenario will occur and an erroneous sequence is generated. This is due to the fact that the sequence YWGQG is cannot be located on read1, and thus the linker string is added to the end of read1 in its entirety.



The same problems may occur on read2 as well, analogous to the example depicted above. Based on test data, 3 amino acid long anchor sequences present a reasonable trade-off. It's highly unlikely that a 3-aa long subsequence occurs more than once, especially if the anchor string contains a rare amino acid such as tryptophan.

A similar outcome occurs if a read error occurs at the stitching regions where the anchor sequence is altered. For instance, sequences that look like CARYYVYPYIDDWGQ or CARYYVYPYIDSWGQ will not match with the anchor sequence DYW and thus will result in unnatural sequences. This is a minor problem, however, as this type of read errors are stochastic and will likely not affect the frequency analysis significantly, any effect in occurrence of a particular sequence should be negligible.

It is essentially difficult to develop a more complex stitching algorithm, since it is difficult to distinguish read errors from biological variability associated with these regions. A more tolerant stitching algorithm would consequentially be dependent on heuristics which are likely to vary from project to project, and thus limit the utility of the scripts.

4.5. Sample labeling

Last portion of the experiment design file, called Samples, allows the user to label the samples. This is just a small feature to make the resultant tables more readable. The key: value pairs given, where the key is the folder name for each sample, and the value is the label associated with the sample.

5. Technical Requirements

5.1. Hardware

There are no real requirements for these scripts to function. However, as the input files are large and contain millions of sequences, the process takes a relatively long time. Furthermore, processing each pair of files is a processor intensive task.

That being the case, each invocation of paired_read script is independent and is suitable for parallel processing. Thus it is not only suitable, but also preferable to run these scripts on a modern multi-core environment. All the tests are run on the UPPMAX cluster, allowing for simultaneous processing of each pair of files. However, any modern computer with as many cores as number of samples would do just as fine.

To give an approximate figure on resources consumed during the tests, 12 cores were utilized for approximately 25-30 minutes, requiring approximately 2GBs of RAM storage in the process. Note that in these tests, each read file contained a couple of million sequences. For larger input files, the process time *should* scale linearly.

5.2. Software

As described in the Workflow section, the main script is written in BASH and the other two scripts are written in Python 2.7 language, a translated version for Python 3 also exists. As such, the use of these scripts require interpreters for these two languages, both of which are included in default installation of any Linux or Mac OSX system.

Please note that it is possible to install Python and Bash on Windows computers as well, however that is beyond the scope of this article, since an overwhelming majority of the modern server or cluster architectures are built on Unix/Linux environment. Those who are interested in getting the bash script to work on windows should consider installing Cygwin for a Linux-like shell environment.

The python scripts also require Biopython package as a dependency, which in turn requires NumPy package. When installing from scratch consider a package manager such as conda in order to resolve any dependencies automatically. For additional scripts for data managing, installing R and RStudio is advised.

6. Appendix

A. Table of ASCII quality scores

Q-score	Probability	Accuracy	ASCII (value, char)	
0	1	0,000%	33	!
1	0,794328235	20,567%	34	"
2	0,630957344	36,904%	35	#
3	0,501187234	49,881%	36	$
4	0,398107171	60,189%	37	%
5	0,316227766	68,377%	38	&
6	0,251188643	74,881%	39	'
7	0,199526231	80,047%	40	(
8	0,158489319	84,151%	41	)
9	0,125892541	87,411%	42	*
10	0,1	90,000%	43	+
11	0,079432823	92,057%	44	,
12	0,063095734	93,690%	45	-
13	0,050118723	94,988%	46	.
14	0,039810717	96,019%	47	/
15	0,031622777	96,838%	48	0
16	0,025118864	97,488%	49	1
17	0,019952623	98,005%	50	2
18	0,015848932	98,415%	51	3
19	0,012589254	98,741%	52	4
20	0,01	99,000%	53	5
21	0,007943282	99,206%	54	6
22	0,006309573	99,369%	55	7
23	0,005011872	99,499%	56	8
24	0,003981072	99,602%	57	9
25	0,003162278	99,684%	58	:
26	0,002511886	99,749%	59	;
27	0,001995262	99,800%	60	<
28	0,001584893	99,842%	61	=
29	0,001258925	99,874%	62	>
30	0,001	99,900%	63	?
31	0,000794328	99,921%	64	@
32	0,000630957	99,937%	65	A
33	0,000501187	99,950%	66	B
34	0,000398107	99,960%	67	C
35	0,000316228	99,968%	68	D
36	0,000251189	99,975%	69	E
37	0,000199526	99,980%	70	F
38	0,000158489	99,984%	71	G
39	0,000125893	99,987%	72	H
40	0,0001	99,990%	73	I
41	7,94328E-05	99,992%	74	J
42	6,30957E-05	99,994%	75	K

B. Experiment design in YAML

The entire experiment design file used for test data. Indentation is important, as it describes relationship between different fields. Lines starting with a # are comments (highlighted in green) and sections are separated long repeats of # sign for convenience of the user.

```
### This file holds runtime configuration for the scFv parsing script
### Blocks are annotated with comments for convenience
#####################################################################
# Trimming sequences, ranges given in nucleotide indices
# A single value refers to a start index to the end, two values indicate start and stop coordinates
Trim:
  fwdread: [27,null]
  revread: [21,150]
#####################################################################
# CDR regions, given as [DIRECTION, START, STOP]
# start and stop values are given in nucleotide indices INCLUDING +-1 AA (3 bp) margins
# Add/remove regions as necessary
Regions:
  cdrh1: [5,0,30]
  cdrh2: [5,69,99]
  cdrh3: [5,207,261]
  cdrl3: [3,99,129]
#####################################################################
# Quality control settings
# Alternative values for "filtering" are "cdr", "combi" or "ave" (see documentation)
# In case of typos or invalid values, filtering method defaults to average quality filtering
Quality:
  threshold: 20
  filtering: cdr
#####################################################################
# Stitching the paired reads
# Linker string is used to join the ends 5'-> 3'
# Anchors are short strings to which the linker will be attached;
# they are expected to be immediately connected to the linker
Stitching:
  linker:
DYWGQGTLVTVSSGGGGSGGGGSGGGGSDIQMTQSPSSLSASVGDRVTITCRASQSISSYLNWYQQKPGKAP
KLLIYAASSLQS
  f_anchor: DYW
  r_anchor: GVP
#####################################################################
# Sample IDs, in the same order (alphabetic) as the folders are listed
Samples:
  P3003_2001: Library
  P3003_2002: GPS2-1
  P3003_2003: GPS8-1
  P3003_2004: PIK3-1
  P3003_2005: GPS2-2
  P3003_2006: GPS8-2
  P3003_2007: PIK3-2
  P3003_2008: GPS2-3
  P3003_2009: GPS8-3
  P3003_2010: PIK3-3
  P3003_2011: GPS2-4
  P3003_2012: GPS8-4
```

C. Command-line cheatsheet

Below are some of the most commonly used commands in a terminal that will likely be useful in this context.

Command	Usage

pwd	Stands for “print working directory”, shows where in the file hierarchy you currently are
cd	Stands for “change directory”. When run without any additional arguments it takes you to your home directory. Otherwise changes the working directory to the given argument, e.g. “cd Projects/NGS” looks for a folder called Projects under the current working directory, if found, looks for NGS folder under that directory. If found, the current working directory is changed, otherwise error message is printed.
ls	This command lists the contents of the current working directory unless it’s run with an argument. For instance, “ls Projects/NGS” will list the contents of NGS folder under Projects, instead.