5PSeq data analysis pipeline

List of software:
bioinfo-tools
bcl2fastq
FastQC
MultiQC
Cutadapt
Samtools
Umi_tools
STAR(or BOWTIE)
Python 3 with libraries: plastid, pysam, numpy, cython, dill, matplotlib, scipy
(already on UPPMAX, but for local use type in bash "sudo pip install scipy, pysam, numpy, cython, dill, matplotlib, plastid")

1. Sequencing raw data:

Bash script to run on UPPMAX. Transforms and de-multiplexes raw sequencing output into individual sample-separated *.fastq* files.
Comment : essential to provide *.csv containing all indexes with the query.

```
#!/bin/bash -l
#SBATCH -p node -n 20
#SBATCH -A snic2017-7-122
#SBATCH      --mail-type ALL --mail-user your.address@scilifelab.se
#SBATCH -t 1:00:00 -J 180630_bcl2fastq_fastqc -o 180630_bcl2fastq_fastqc.txt

module load bioinfo-tools
module load bcl2fastq

bcl2fastq -R
/crex1/proj/sllstore2017018/private/RawData/180630_NB501365_0239_AHKFW5BGX7 -o
/crex1/proj/sllstore2017018/nobackup/projects/5PSeq_VB/fastq --sample-sheet
180630_SampleSheet_Susan.csv --no-lane-splitting --barcode-mismatches 1

module load FastQC

cd /crex1/proj/sllstore2017018/nobackup/projects/5PSeq_VB/fastq
for i in *.fastq.gz; do echo $i; fastqc -o
/crex1/proj/sllstore2017018/nobackup/projects/5PSeq_VB/FastQC -t 20 $i; done
```

Comment:
When we previously used this setting for demultiplexing, we had some samples separated incorrectly, resulting in some <u>non-biological</u> "cross-contamination". Suggesting to try –barcode-mismatches 0 in the future

2. Post-processing *.fastq

This was ran locally on my computer, which works well for smaller genomes like yeast or bacteria. A different pipeline could be used in UPPMAX for large genomes/bigger libraries.
Extract archives in custom directory. From the directory, run following bash script:

Comment: sequence specific for the adapter used in the sequencer, and in this case single-end reads.

```
mkdir cutadapt
for f in *.fastq; do cutadapt -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC --match-read-wildcards -m 10 -o cutadapt/$f $f -j 4 ; done

cd cutadapt
mkdir umitools
```

Comment : extract 8 nt at the beginning of transcript to move UMI to header

```
for f in *.fastq; do umi_tools extract --bc-pattern NNNNNNNN --stdin $f --stdout umitools/$f -L logfile_$f.txt; done
cd umitools
mkdir fastqc
```

Comment: Fastqc after trimming the UMI headers, to be used for evaluation of mapping results

```
for f in *.fastq; do fastqc $f -o ./fastqc ; done
```

Comment: STAR used for aligning, although other aligners would work just as well or even better depending on the experiment.

```
mkdir star
for f in *.fastq; do STAR --runThreadN 4 --readFilesIn $f --genomeDir ../../../../genome --outSAMtype BAM SortedByCoordinate  --outFileNamePrefix star/$f --outFilterMultimapNmax 1 --alignIntronMax 50 --limitBAMsortRAM 1433281461; done
```

Comment: here we use almost default settings for alignment with STAR. We do not allow multi-mapping reads to be aligned, and max length of introns can be adjusted according to species. Generally we want to allow very short introns always in order to map reads which may contain indels. Yeast tend to have real introns, and those can be fairly long, so when aligning yeast, I usually allow 1000 base introns.

```
multiqc .
cd star
for f in *.bam; do samtools index $f; done
```

Comment: sort mapped reads as umi_tools dedup requires index

```
mkdir dedup
for f in *.bam; do umi_tools dedup -I $f -S dedup/$f -L $f_log.txt; done
cd dedup
for f in *; do samtools index $f; done
```

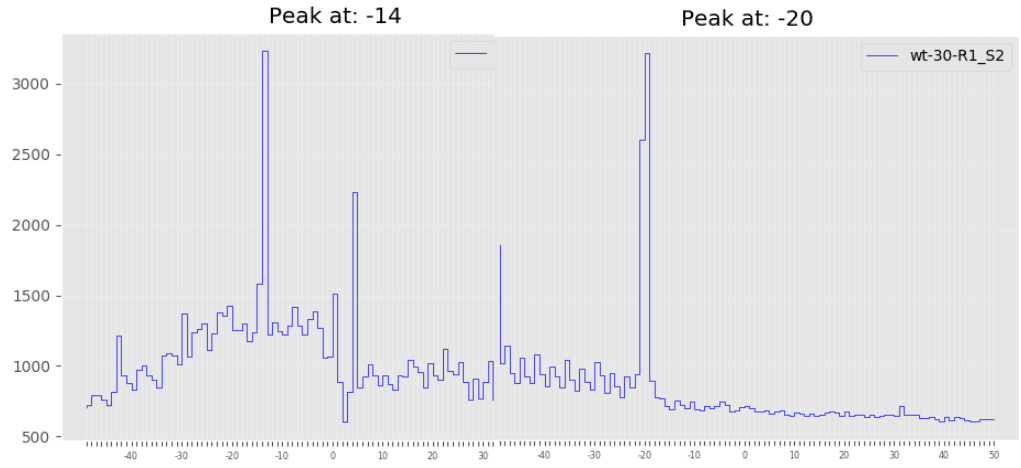
Comment: indexing required for files to be read by plastid

---

3. Custom pipeline with Plastid (python)

Plastid is a useful Python library which was created for analysis of ribosome profiling data. Some of the command line utilities in Plastid are very good for the kind of analysis we are interested in, but are written for ribosome profiling data (which includes fragments of 20-50nt length)
I have written several all-purpose custom scripts based on Plastid library which are more suitable for 5'PSeq data. These scripts support multi-threading, one sample per thread when analyzed in batch.

coverage_boundries.py
Creates plots showing metagene coverage X nucleotides around translation start and end sites. The script aligns all transcripts at the start and end, which allows us to see the genome-wide coverage. The peaks are calculated relative to the coding transcript's annotated boundries, so relative to first nucleotide of first codon and first nucleotide AFTER stop codon.



Arguments:

Path to directory where sample is, and path to existing directory where to save plots
--sample_dir
--output_dir

Path to annotation files (fasta and gtf/gff)
--genome_fasta
--annotation_file

List of genes which we want to include in analysis (one per line)
--gene_set

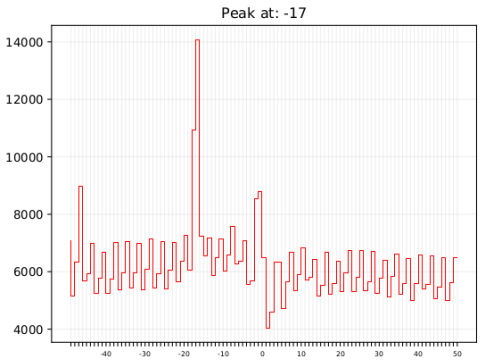
How many cores to use (one per sample at a time)
--cores

Span to extend the annotated coordinates (to see coverage beyond translation start site)
--offset

Normalize by transcript. This would cause high and low coverage transcripts to contribute equally
--normalize

coverage_codon_frame.py

Creates plots (ans saves raw value vectors) showing metagene coverage X nucleotides around every codon and amino-acid (excluding the first codon, as this can introduce biases specific to translation initiation). Calculates peak relative to the first nucleotide in codon plotted at 0.
This can show genome-wide trends of ribosome dynamics without requiring deep coverage of individual transcript.
Specifying –frame 0, --frame 1 or --frame 2 could allow us to look at the coverage in out-of frame codons (usually provide no significant pattern)



Coverage relative to TYR aminoacid

Arguments:

Path to directory where sample is, and path to existing directory where to save plots
--sample_dir
--output_dir

Path to annotation files (fasta and gtf/gff)
--genome_fasta
--annotation_file

List of genes which we want to include in analysis (one per line)
--gene_set

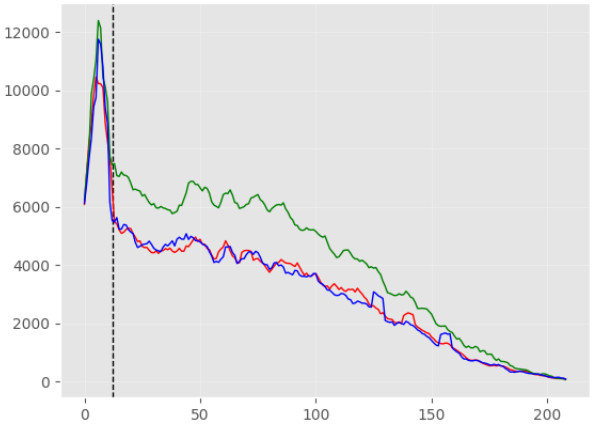
How many cores to use (one per sample at a time)
--cores

Spaning window which to plot
--offset

Normalize by transcript. This would cause high and low coverage transcripts to contribute equally
--normalize

reading_frames_split.py

Creates plots of smoothed metagene coverage specific to nucleotide position in the codon, aligned to start or end of the transcript. The scale labels are incorrect, but the vertical black line indicates transcript start or end. This plot is useful to find frameshifts which are a genome-wide trend, or in individual genes with very high coverage



Arguments:

Filter out transcripts based on length
--longest
--shortest

Path to directory where sample is, and path to existing directory where to save plots

--sample_dir
--output_dir

Path to annotation files (fasta and gtf/gff)
--genome_fasta
--annotation_file

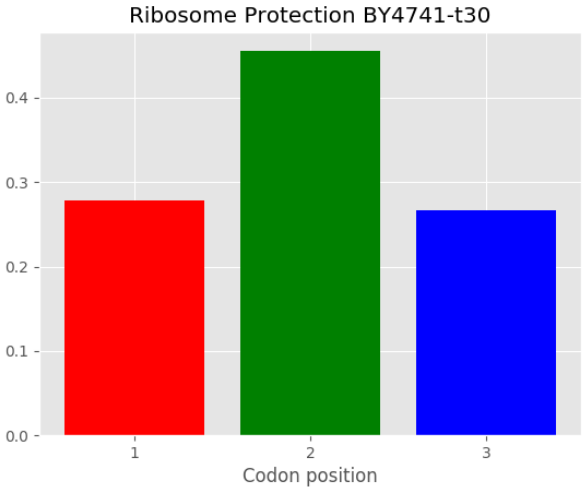
List of genes which we want to include in analysis (one per line)
--gene_set

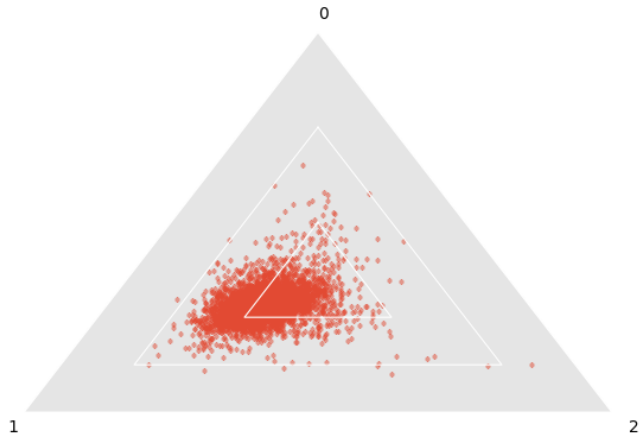
How many cores to use (one per sample at a time)
--cores

Span to extend the annotated coordinates (to see coverage beyond translation start site)
--offset

Normalize by transcript. This would cause high and low coverage transcripts to contribute equally
--normalize

triangle_genes.py
Similar to ***reading_frames_split.py***, this script provides a plot reflecting of coverage of nucleotides in each position of codon. Each dot represents a transcript, and this helps visualize heterogeneity and frame preference of our library. The bar plot shows the average of all codons in the genome.





Arguments:

Path to directory where sample is, and path to existing directory where to save plots
--sample_dir
--output_dir

Path to annotation files (fasta and gtf/gff)
--annotation_file

List of genes which we want to include in analysis (one per line)
--gene_set

How many cores to use (one per sample at a time)
--cores

Span to extend the annotated coordinates (to see coverage beyond translation start site)
--offset

Normalize by transcript. This would cause high and low coverage transcripts to contribute equally
--normalize

These scripts can be ran from command-line and **require some additional files** to be provided

Annotation GTF or GFF and genome sequence in *.fasta format (get them from NCBI Genomes > Genbank/RefSeq annotation).
These are used to map the reads to genome, but also needed for plastid to translate genomic coordinates to local transcript coordinates, and match the genetic code to those coordinates.

Line-delimited gene list
Generally, non-protein coding genes like rRNA/tRNAs and ncRNAs introduce a lot of noise in our analysis which we want to avoid. We can provide scripts with a list of genes which we know or predict are protein-coding for our research purposes.
The gene list should include the same reference ID as found under “transcript id” in GTF/GFF annotation. There is a lot of discrepancy in how those are annotated between genome assemblies and experiments, so I devised a universal solution which should work for most cases.
Depending whether we work with gtf or gff, we would run a script which parses the annotated protein-coding identifiers in the exact format which plastid can recognize. For GTF we can parse directly from the gtf file, while for GFF we would need to parse the “Feature table” file from the same RefSeq/Genbank assembly.
get_coding_gtf.py
get_coding_gff.py

Multiple specie samples
It is very straight-forward to map to multi-species genome.

Merge genome fastas using bash cat command.
Merge genomic GFF using bash cat command.
Extract coding gene list from each genome separately.
Create STAR index using the merged files
Map the sample to the merged file.
Run command-line scripts each time for each different gene list.

This should work without a hitch, but be mindful as STAR genome generation needs modified indexing settings when our genome contains a lot of contigs