

Refactoring Riboviz Analysis Code: *A Personal Journey*

24 June 2020

Flic Anderson

The Wallace Lab, University of Edinburgh

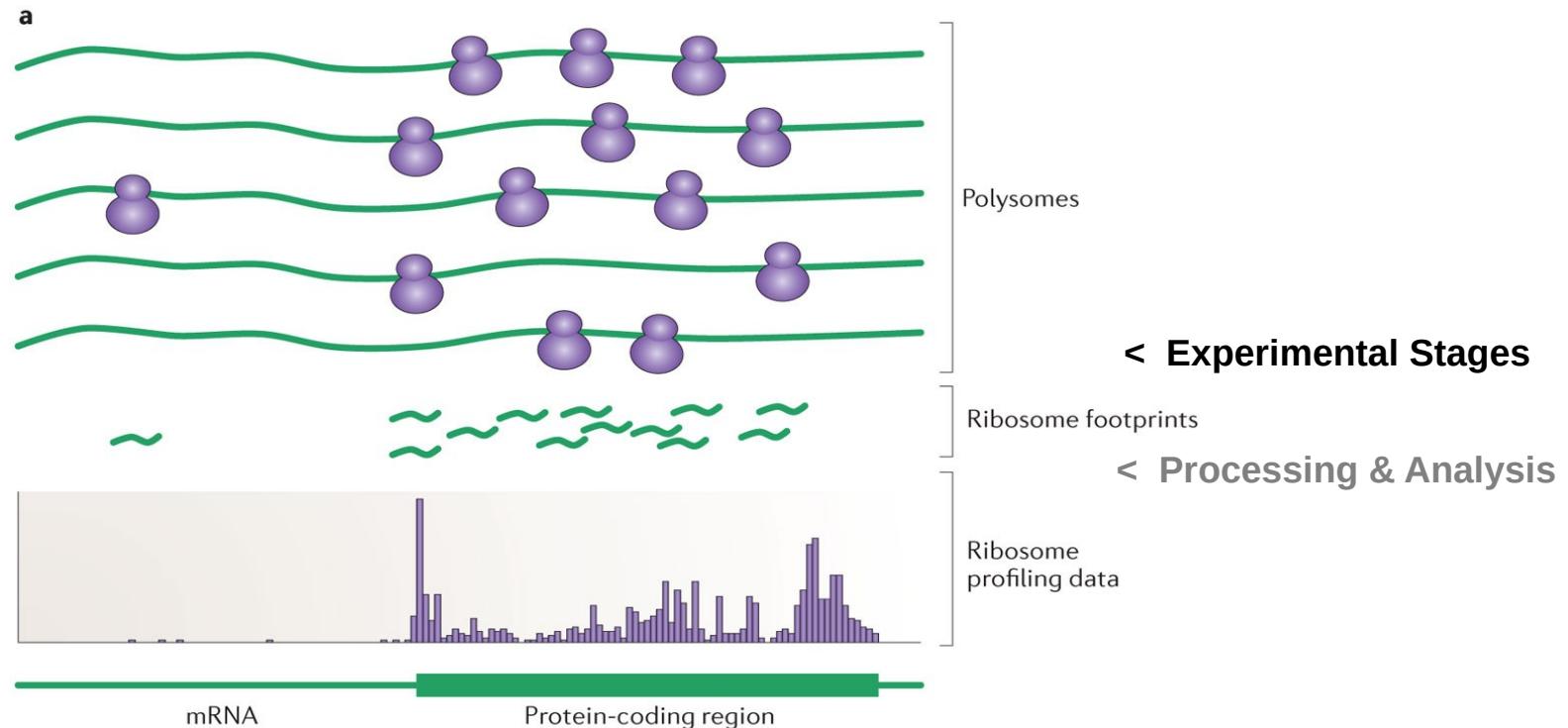
Outline

- Riboviz 101
- Ribosome Profiling Process
- Riboviz Workflow
- Analysis Code Refactoring
 - Example
 - Tips
 - Fails
- Call to action!
- Future Priorities

Riboviz 101

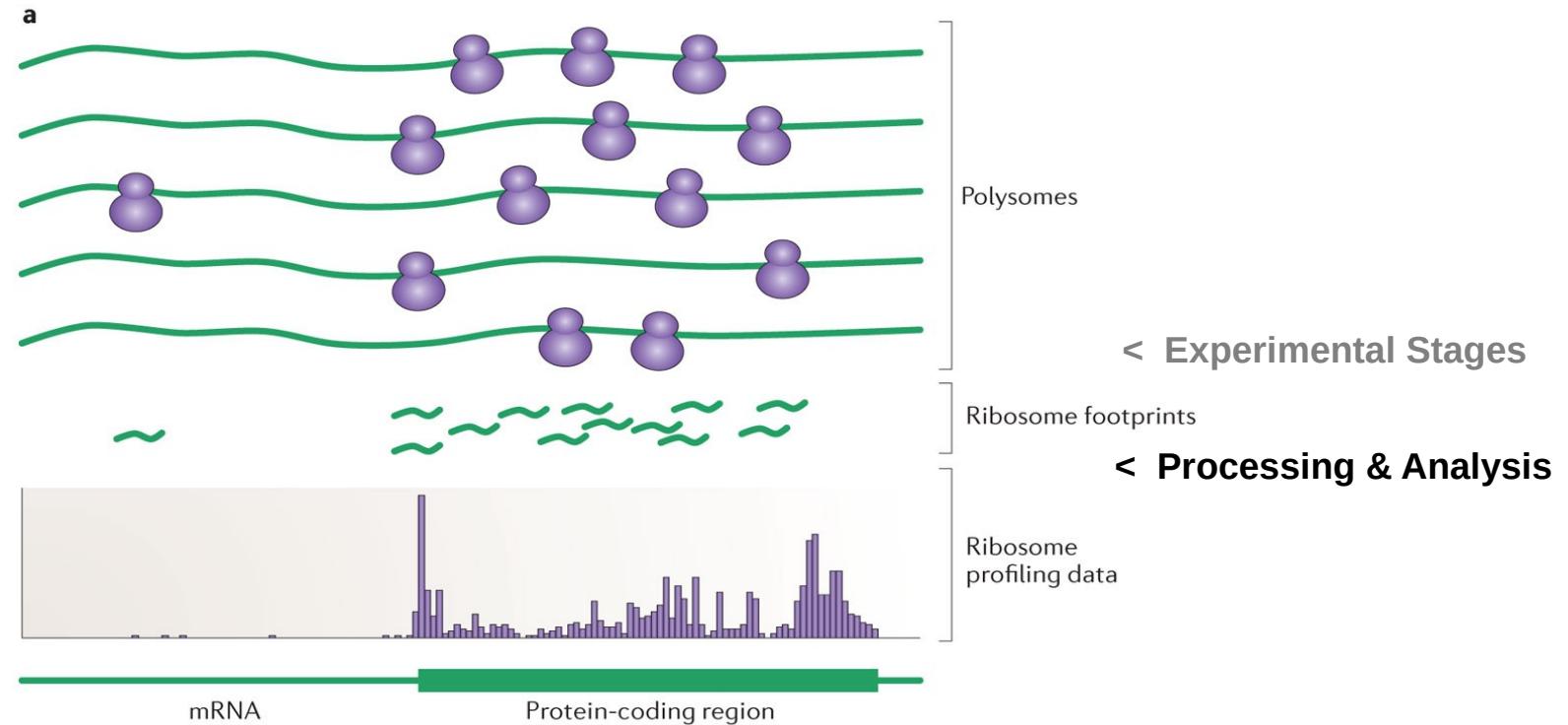
- **Riboviz processes & analyses ribosome profiling data**
- Ribosome profiling data helps unlock details of **active translation**: *mechanics of translation, regulation methods, translational efficiency*
- Developing/improving riboviz = **more researcher time** for biological questions rather than tinkering with pipelines & bespoke analysis code...

Polysomes to Footprints



Ingolia (2014). "Ribosome profiling: new views of translation, from single codons to genome scale". Nature Reviews. Genetics. 15 (3): 205–13. doi:10.1038/nrg3645

Footprints to Profiling Data



Ingolia (2014). "Ribosome profiling: new views of translation, from single codons to genome scale". Nature Reviews. Genetics. 15 (3): 205–13. doi:10.1038/nrg3645

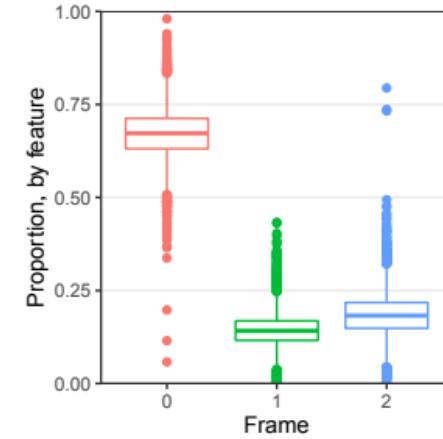
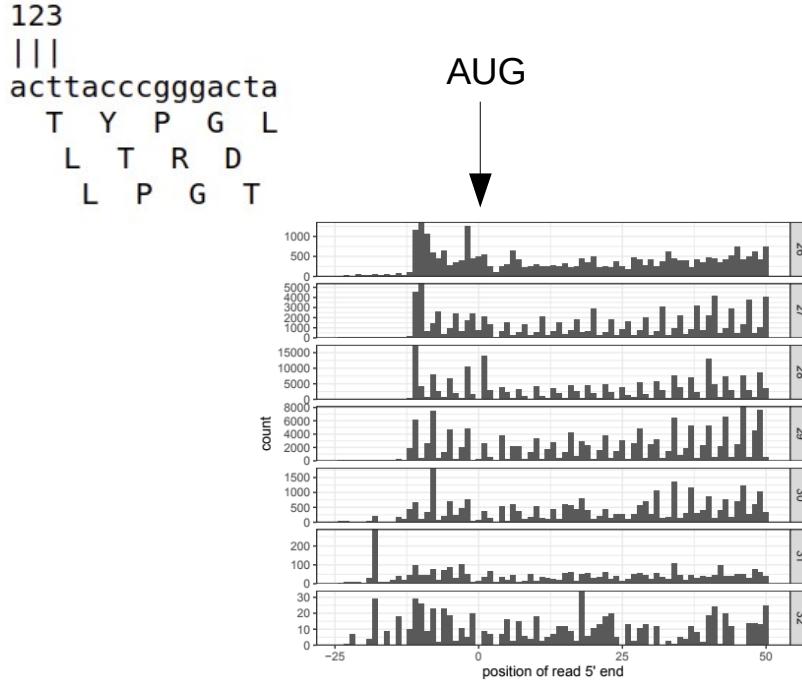
Analysis of Ribosome Profiling Data

- Looking for **3-NT periodicity**: ribosomes moving along transcript 1 codon at a time
- Most reads map to **coding regions** (98.8% in Ingolia et al 2009)
- Reasonable read-lengths (e.g. know should look for appx **~28-30NT**)
- Looking for most reads to be in **one frame**

Ribosome Data & Reading Frame

reading frame:

first reading frame
second reading frame
third reading frame



G-Sc_2014: WTnone

Footprints to Ribosome Profiling Data

- **Processing:** *lots of steps*
 - Removing adapter sequences
 - Remove UMIs (Unique Molecular Identifiers) & barcodes if present
 - Demultiplex / Deduplicate reads if required
 - Need to filter out contaminant reads
 - Align reads to transcriptome
- **Analysis:** *more steps*
 - Analyse & quantify data:
 - Create outputs (including for quality-control, further analysis)

Riboviz Workflow: Inputs

Organism Specific

Transcript Sequences
.fasta

Genome / Transcriptome
Features
.gff

Contaminant Sequences
(rRNA)
.fasta

(Additional
Organism-Specific
Data)

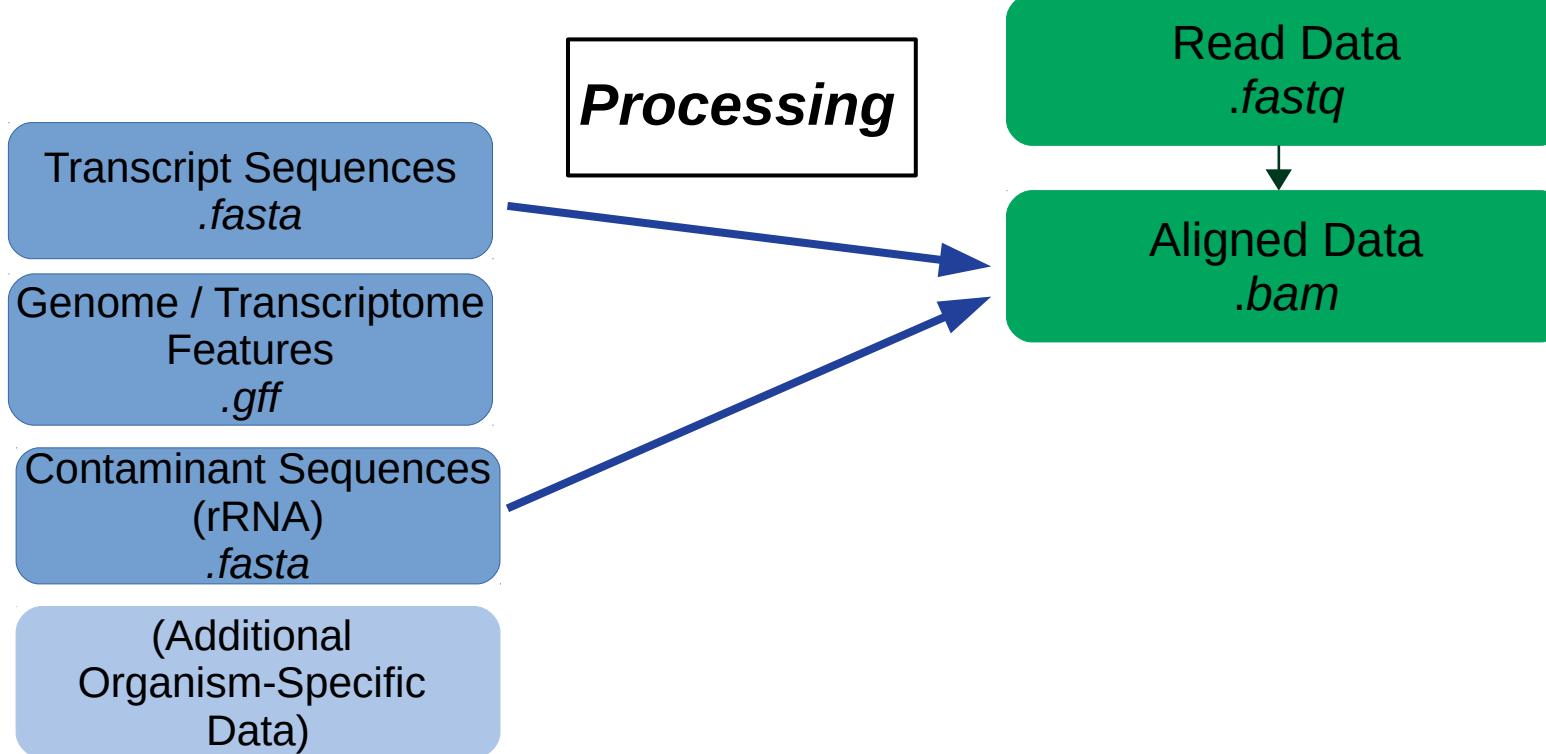
Sample Specific

Read Data
.fastq

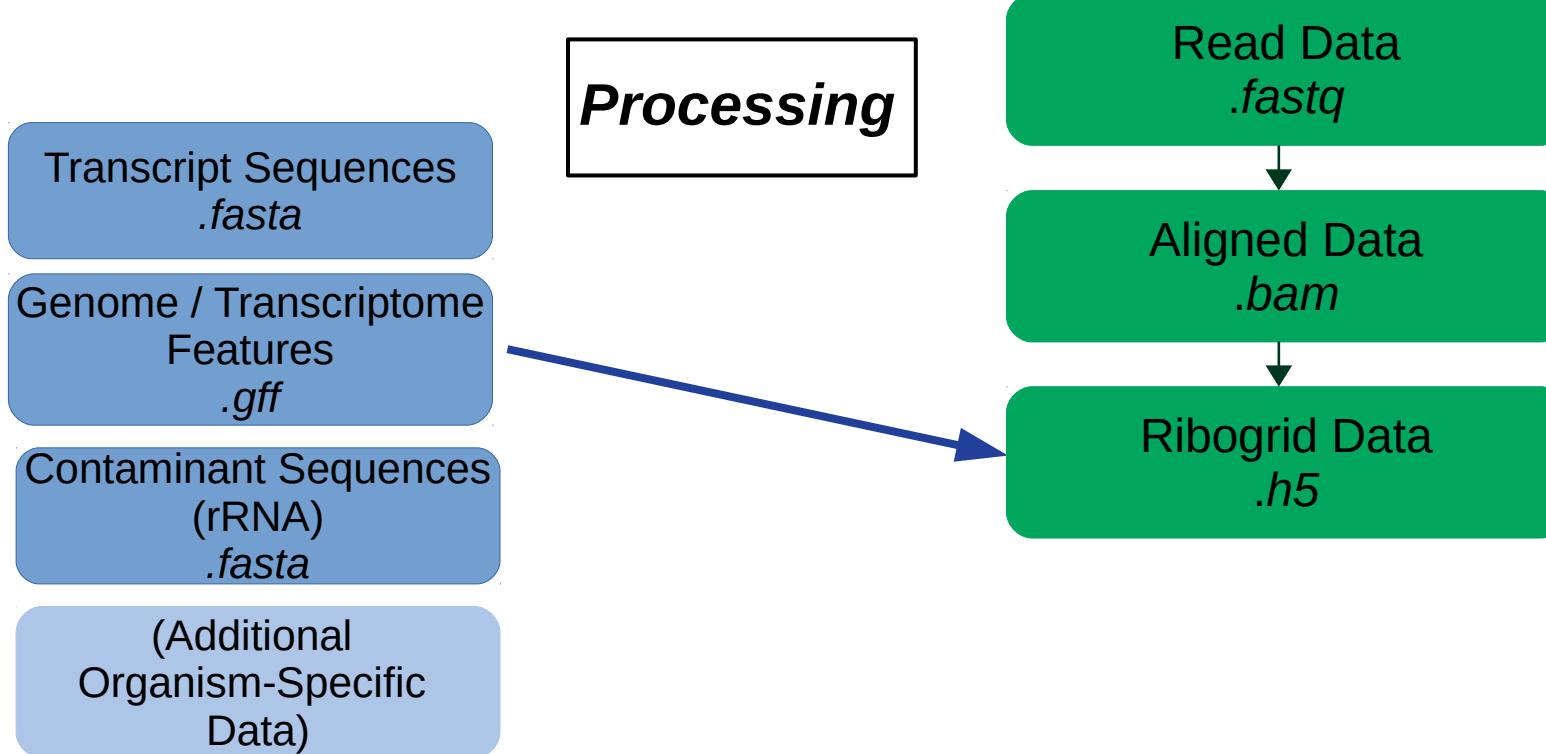
Configuration File
.yaml

Configuration File lists all files & parameters needed to run RiboViz

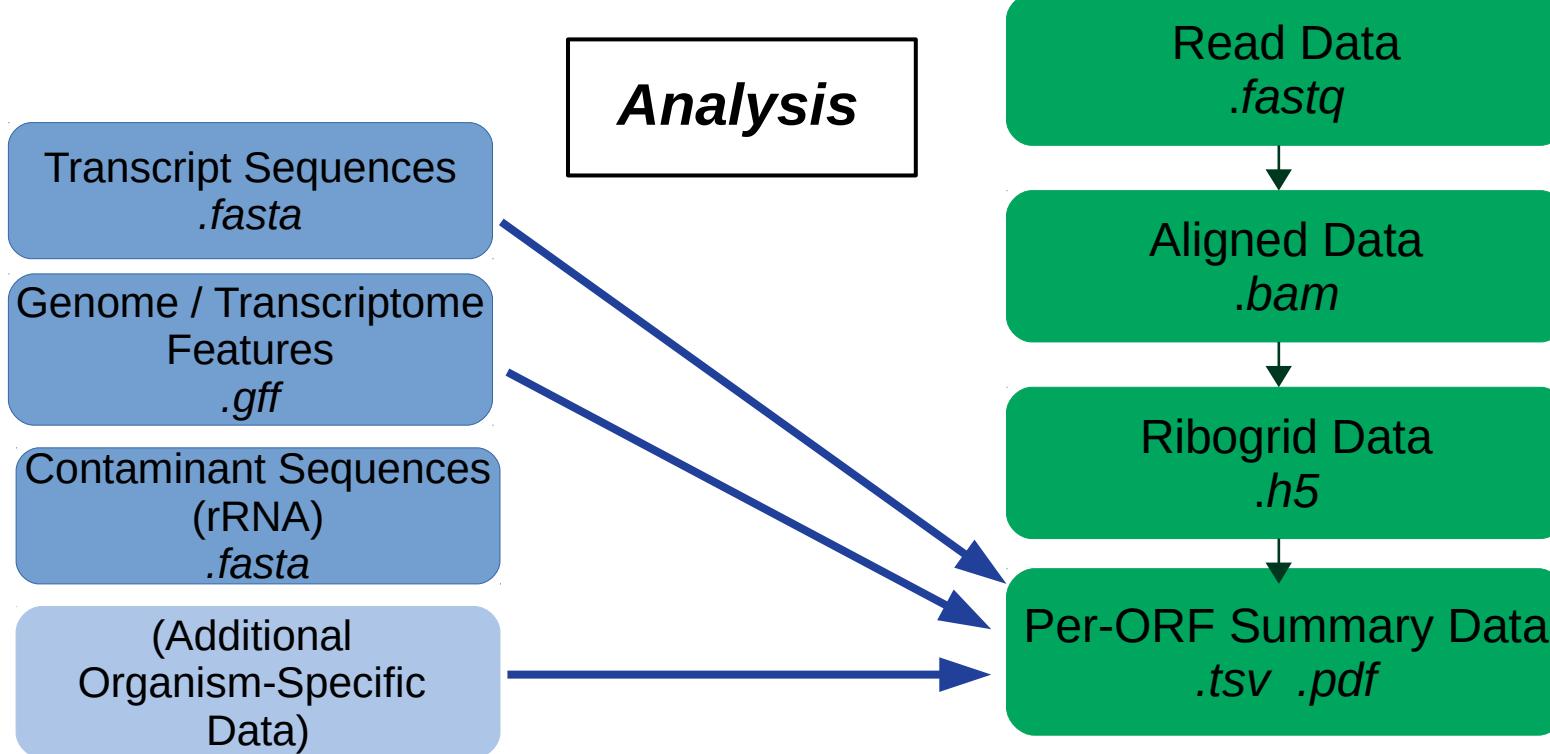
Riboviz Workflow



Riboviz Workflow



Riboviz Workflow



Analysis

`generate_stats_figs.R`:

Inputs:

- sample read data from .h5 file
- organism-specific transcriptome data from .gff & .fa
- parameters from configuration .yaml file

Generates summary statistics, analysis plots & quality control plots

Outputs:

- .pdf plots
- .tsv files

Refactoring Analysis Code

`generate_stats_figs.R`

Complex Analysis Code

1000+ lines

Lots of different processes & analyses

Defining Functions

Using Functions

Multiple dialects of R

Debugging Nightmare!

Refactor to Big Chunks

Setup Code (libraries etc)

> 3-NT (3-Nucleotide) Periodicity

Length of Mapped Reads

Biases in Nucleotide Composition

Calculate Read Frame per ORF

Position Specific Distribution of Reads

TPMs of Genes

TPMs Correlations With Gene Features

Then... to Medium Chunks

Calculate 3NT Periodicity Function

> Plot 3NT Periodicity Function

Output 3NT Periodicity Function

v

Then... to Small Chunks

Prep Output Function

Run/Do Output Function

v

Achieve Code Zen!

Anatomy of a Code Chunk...

- 3NT Periodicity Big Code Chunk:
 - Calculate! Get start positions & read counts at each position for each gene from the .h5 file, calculate periodicity from this matrix of positions & counts
 - Plots! {ggplot2}
 - Save plots as .pdf
 - Write information out as .tsv (includes provenance info)

```
313 ThreeNucleotidePeriodicity <- function(gene_names, dataset, hd_file, gff_df) {  
314  
315     # check for 3nt periodicity  
316     print("Starting: Check for 3nt periodicity globally")  
317  
318     # CalculateThreeNucleotidePeriodicity():  
319     three_nucleotide_periodicity_data <- CalculateThreeNucleotidePeriodicity(gene_names = gene_names, dataset = dataset, hd_file =  
320  
321     # PlotThreeNucleotidePeriodicity()  
322     three_nucleotide_periodicity_plot <- PlotThreeNucleotidePeriodicity(three_nucleotide_periodicity_data)  
323  
324     # NOTE: repeated from inside CalculateThreeNucleotidePeriodicity() as preferred not to return multiple objects in list (hassle  
325     gene_poslen_counts_5start_df <- AllGenes5StartPositionLengthCountsTibble(gene_names = gene_names, dataset= dataset, hd_file =  
326  
327     # run PlotStartCodonRiboGrid()  
328     start_codon_ribogrid_plot <- PlotStartCodonRiboGrid(gene_poslen_counts_5start_df)  
329     # creates plot object  
330  
331     # run SaveStartCodonRiboGrid():  
332     SaveStartCodonRiboGrid(start_codon_ribogrid_plot)  
333  
334     # run PlotStartCodonRiboGridBar():  
335     start_codon_ribogrid_bar_plot <- PlotStartCodonRiboGridBar(gene_poslen_counts_5start_df)  
336     # creates plot object  
337  
338     # run SaveStartCodonRiboGridBar():  
339     SaveStartCodonRiboGridBar(start_codon_ribogrid_bar_plot)  
340  
341     # run SavePlotThreeNucleotidePeriodicity():  
342     SavePlotThreeNucleotidePeriodicity(three_nucleotide_periodicity_plot)  
343  
344     # run WriteThreeNucleotidePeriodicity():  
345     WriteThreeNucleotidePeriodicity(three_nucleotide_periodicity_data)  
346  
347     print("Completed: Check for 3nt periodicity globally")  
348  
349 } # end ThreeNucleotidePeriodicity() function definition  
350 # run ThreeNucleotidePeriodicity():  
351 ThreeNucleotidePeriodicity(gene_names, dataset, hd_file, gff_df)
```

```

180
181 CalculateThreeNucleotidePeriodicity <- function(gene_names, dataset, hd_file, gff_df){
182
183   # get gene and position specific total counts for all read lengths
184   gene_poslen_counts_5start_df <- AllGenes5StartPositionLengthCountsTibble(gene_names = gene_names, dataset =
185
186   gene_poslen_counts_3end_df <- AllGenes3EndPositionLengthCountsTibble(gene_names = gene_names, dataset = da
187
188   # summarize by adding different read lengths
189   gene_pos_counts_5start <- gene_poslen_counts_5start_df %>%
190     group_by(Pos) %>%
191     summarize(Counts = sum(Counts))
192   # gives:
193   # > str(gene_pos_counts_5start)
194   # Classes 'tbl_df', 'tbl' and 'data.frame': 75 obs. of 2 variables:
195   # $ Pos : int -24 -23 -22 -21 -20 -19 -18 -17 -16 -15 ...
196   # $ Counts: int 285 318 307 386 291 347 840 330 475 355 ...
197
198   gene_pos_counts_3end <- gene_poslen_counts_3end_df %>%
199     group_by(Pos) %>%
200     summarize(Counts = sum(Counts))
201   # gives:
202   # > str(gene_pos_counts_3end)
203   # Classes 'tbl_df', 'tbl' and 'data.frame': 75 obs. of 2 variables:
204   # $ Pos : int -49 -48 -47 -46 -45 -44 -43 -42 -41 -40 ...
205   # $ Counts: int 19030 13023 50280 19458 12573 46012 19043 13282 36968 20053 ...
206
207   three_nucleotide_periodicity_data <- bind_rows(
208     gene_pos_counts_5start %>% mutate(End = "5'"),
209     gene_pos_counts_3end %>% mutate(End = "3'"))
210   ) %>%
211   mutate(End = factor(End, levels = c("5'", "3'")))
212   # gives:
213   # > str(three_nucleotide_periodicity_data)
214   # Classes 'tbl_df', 'tbl' and 'data.frame': 150 obs. of 3 variables:
215   # $ Pos : int -24 -23 -22 -21 -20 -19 -18 -17 -16 -15 ...
216   # $ Counts: int 285 318 307 386 291 347 840 330 475 355 ...
217   # $ End : Factor w/ 2 levels "5'","3'": 1 1 1 1 1 1 1 1 1 1 ...
218
219   return(three_nucleotide_periodicity_data)
220
221 } # end CalculateThreeNucleotidePeriodicity() definition
222 # gives:
223 # CalculateThreeNucleotidePeriodicity(gene_names = gene_names, dataset = dataset, hd_file = hd_file, gff_
224 # # A tibble: 150 x 3
225 #   Pos Counts End
226 #   <int> <int> <fct>
227 #     1    -24    285 5'
228 #     2    -23    318 5'
229 #     3    -22    307 5'
230 #     4    -21    386 5'
231 #     5    -20    291 5'
232 #     6    -19    347 5'
233 #     7    -18    840 5'
234 #     8    -17    330 5'
235 #     9    -16    475 5'
236 #    10    -15    355 5'
237 # ... with 140 more rows
238

```

```

238
239 # define PlotThreeNucleotidePeriodicity() function with reasonable arguments
240 PlotThreeNucleotidePeriodicity <- function(three_nucleotide_periodicity_data){
241
242   # Plot
243   three_nucleotide_periodicity_plot <- ggplot(
244     three_nucleotide_periodicity_data,
245     aes(x = Pos, y = Counts)) +
246     geom_line() +
247     facet_wrap(~End, scales = "free") +
248     labs(x = "Nucleotide Position", y = "Read counts")
249
250   return(three_nucleotide_periodicity_plot)
251
252 } # end PlotThreeNucleotidePeriodicity() definition
253
254 # potentially replace/tweak plot_ribogrid() to follow StyleGuide
255 PlotStartCodonRiboGrid <- function(gene_poslen_counts_5start_df){
256   # function to do the ribogrid & ribogridbar plots?
257   # ribogrid_5start
258   start_codon_ribogrid_plot <- plot_ribogrid(gene_poslen_counts_5start_df)
259   return(start_codon_ribogrid_plot)
260 } # end PlotStartCodonRiboGrid() definition
261
262 SaveStartCodonRiboGrid <- function(start_codon_ribogrid_plot){
263   # function to do the ribogrid & ribogridbar plots?
264   # ribogrid_5start
265   start_codon_ribogrid_plot %>%
266     ggsave(
267       filename = file.path(output_dir, paste0(output_prefix, "startcodon_ribogrid.pdf")),
268       width = 6, height = 3
269     )
270   #return() # no return as writing-out
271 } # end SaveStartCodonRiboGrid() definition
272
273 PlotStartCodonRiboGridBar <- function(gene_poslen_counts_5start_df){
274   start_codon_ribogrid_bar_plot <- barplot_ribogrid(gene_poslen_counts_5start_df)
275   return(start_codon_ribogrid_bar_plot)
276 } # end PlotStartCodonRiboGridBar() definition
277
278 SaveStartCodonRiboGridBar <- function(start_codon_ribogrid_bar_plot){
279   start_codon_ribogrid_bar_plot %>%
280     ggsave(
281       filename = file.path(output_dir, paste0(output_prefix, "startcodon_ribogridbar.pdf")),
282       width = 6, height = 5
283     )
284   #return() # no return as writing-out
285 } # end SaveStartCodonRiboGridBar() definition
286
287 SavePlotThreeNucleotidePeriodicity <- function(three_nucleotide_periodicity_plot) {
288   # Save plot and file
289   ggsave(
290     three_nucleotide_periodicity_plot,
291     filename = file.path(output_dir, paste0(output_prefix, "3nt_periodicity.pdf"))
292   )
293   # return() # NO RETURN as writing out
294 } # end of function definition SavePlotThreeNucleotidePeriodicity()
295

```

```
...
296 WriteThreeNucleotidePeriodicity <- function(three_nucleotide_periodicity_data) {
297   tsv_file_path <- file.path(output_dir, paste0(output_prefix, "3nt_periodicity.tsv"))
298   write_provenance_header(path_to_this_script, tsv_file_path)
299   write.table(
300     three_nucleotide_periodicity_data,
301     file = tsv_file_path,
302     append = T,
303     sep = "\t",
304     row = F,
305     col = T,
306     quote = F)
307   # return()? NO RETURN
308 } # end of function definition WriteThreeNucleotidePeriodicity()
309
```

Refactoring Tips

- Break it down into **chunks!**
- **Regression tests** are your friend
- Develop functions in your main script & then move into **separate functions script**
- Keep it specific: **issue-focussed working**
 - ... Decide how you'll know you're finished...

Refactoring LMFs

- **Issue proliferation**... Getting lost amongst different issues & losing sight of the main goals
- Schroedinger's **debugging** issues!
- **Package::function()** is very useful!
- ... Remember to **tell collaborators** about & document the new packages you add to the code to solve problems...

Fails are not ALWAYS a failure!

- <3 Regression tests!
- Which output is the **correct** output? How do we know?
- How to decide when to make a **new issue** or keep working on a problem?
- **Rollback...**
- Importance of **code testing**... & understandable code!

“Putting the YOU into USER”

- User Testing & User Interviews – *Sam!*
- Existing Documentation Feedback – *Siyin* & others already helping with this :)
- Leave the Code Docs to the Robots? – {roxygen2} – hoping to chat to *Edward* about this!
- General Comparisons - what features of other software doing similar things do you find useful/unhelpful? - **Everyone!!!**

“Testable, Reliable CompYOUtation”

- **Code Testing:**
 - Helping integrate functions from *Ania & Siyin’s* projects: want to have good testable code...
 - **{testthat} R package** – would be great to chat with *Xuejia* on how this works
- **More data!**
 - This lab: *Rosey?*
 - Others? Let’s talk :)

Priorities

- **Tests** for the analysis code
- generate_stats_figs.R **finishing touches**: Documentation / better output format / styling
- **New datasets** run & added to example datasets repository
- Integrate **new functions** from Ania & Siyin's project work
- **User focus**
- Q: How does riboviz compare with **other tools**
- Q: Do we have the right **statistics** for diagnostics?

Thanks / Acknowledgements

- BBSRC-NSF funded project
- Collaborative project:
 - Edward Wallace: *The Wallace Lab*, The University of Edinburgh.
 - + Siyin Xue, Ania Kurowska
 - Premal Shah, John Favate, Tongji Xing: *The Shah Lab*, Rutgers University.
 - Liana Lareau, Amanda Mok: *The Lareau Lab*, University of California, Berkeley.
 - Kostas Kavousannakis, Mike Jackson: *EPCC*, The University of Edinburgh.
 - Oana Carja, Joshua Plotkin: The University of Pennsylvania

Questions?

EXTRA SLIDES

Process ribosome profiling sample data

If sample files (`fq_files`) are specified, then the workflow processes the sample files as follows:

1. Read configuration information from YAML configuration file.
2. Build hisat2 indices if requested (if `build_indices: TRUE`) using `hisat2 build` and save these into the index directory (`dir_index`).
3. Process each sample ID-sample file pair (`fq_files`) in turn:
 - i. Cut out sequencing library adapters (`adapters`) using `cutadapt` .
 - ii. Extract UMIs using `umi_tools extract` , if requested (if `extract_umis: TRUE`), using a UMI-tools-compliant regular expression pattern (`umi-regexp`). The extracted UMIs are inserted into the read headers of the FASTQ records.
 - iii. Remove rRNA or other contaminating reads by alignment to rRNA index files (`rrna_index_prefix`) using `hisat2` .
 - iv. Align remaining reads to ORFs index files (`orf_index_prefix`). using `hisat2` .
 - v. Trim 5' mismatches from reads and remove reads with more than 2 mismatches using `trim_5p_mismatch` .
 - vi. Output UMI groups pre-deduplication using `umi_tools group` if requested (if `dedup_umis: TRUE` and `group_umis: TRUE`)
 - vii. Deduplicate reads using `umi_tools dedup` , if requested (if `dedup_umis: TRUE`)
 - viii. Output UMI groups post-deduplication using `umi_tools group` if requested (if `dedup_umis: TRUE` and `group_umis: TRUE`)
 - ix. Export bedgraph files for plus and minus strands, if requested (if `make_bedgraph: TRUE`) using `bedtools genomecov` .
 - x. Write intermediate files produced above into a sample-specific directory, named using the sample ID, within the temporary directory (`dir_tmp`).
 - xi. Make length-sensitive alignments in compressed h5 format using `bam_to_h5.R` .
 - xii. Generate summary statistics, and analyses and QC plots for both RPF and mRNA datasets using `generate_stats_figs.R` . This includes estimated read counts, reads per base, and transcripts per million for each ORF in each sample.
 - xiii. Write output files produced above into an sample-specific directory, named using the sample ID, within the output directory (`dir_out`).
4. Collate TPMs across results, using `collate_tpms.R` and write into output directory (`dir_out`). Only the results from successfully-processed samples are collated.
5. Count the number of reads (sequences) processed by specific stages if requested (if `count_reads: TRUE`).



```
11
12 # Handle interactive session behaviours or use get_Rscript_filename():
13 if (interactive()) {
14   # Use hard-coded script name and assume script is in "rscripts"
15   # directory. This assumes that interactive R is being run within
16   # the parent of rscripts/ but imposes no other constraints on
17   # where rscripts/ or its parents are located.
18   this_script <- "generate_stats_figs.R"
19   path_to_this_script <- here("rscripts", this_script)
20   source(here::here("rscripts", "provenance.R"))
21   source(here::here("rscripts", "read_count_functions.R"))
22 } else {
23   # Deduce file name and path using reflection as before.
24   this_script <- getopt::get_Rscript_filename()
25   path_to_this_script <- this_script
26   source(here::here("rscripts", "provenance.R"))
27   source(here::here("rscripts", "read_count_functions.R"))
28 }
29
```

Updates! UX

- Users:
 - Building user base?
 - Needs / wants?
 - Oven-ready datasets*?

* <https://github.com/riboviz/example-datasets>

Updates! UX

- Support & Documentation:
 - Existing docs suitable?
 - Document outputs better?
 - Translation UK workshop?

Updates! Testing

- General Testing:
 - New feature development ongoing
 - Bug fixes
 - Code reviews happening
 - New datasets

Updates! Testing

- Methods Testing:
 - Regression tests
 - Expected outputs: simulated data.

First Things First: Setup

- Initial Setup Code:
 - load libraries
 - handle interactive session behaviours if required*
 - source provenance & functions scripts
 - load parameters passed in from main riboviz workflow
 - read in key files (.gff, .fa, .h5)

Refactoring Riboviz Analysis Code: *A Personal Journey*

24 June 2020

Flic Anderson

The Wallace Lab, University of Edinburgh

1 / 27

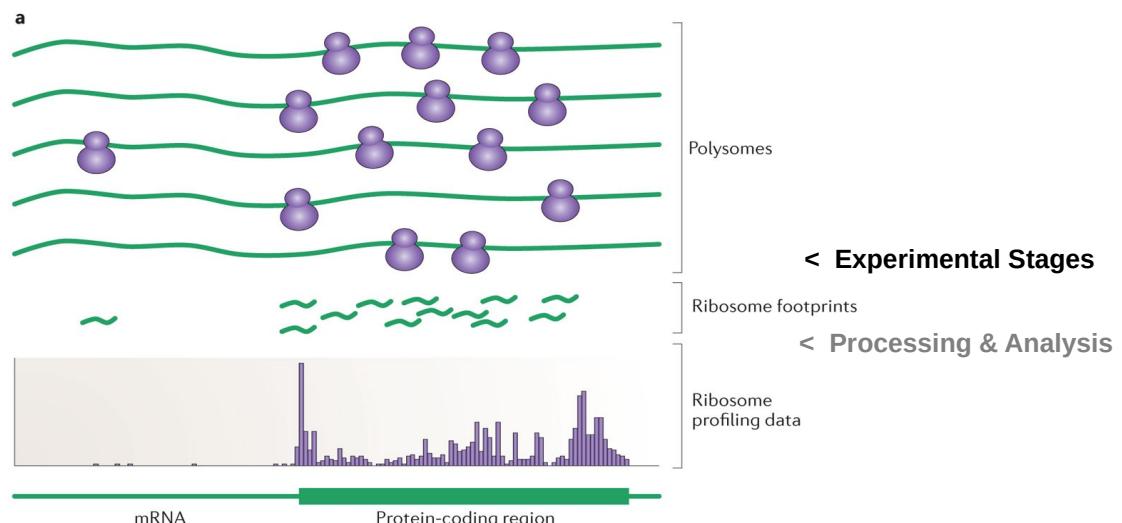
Outline

- Riboviz 101
- Ribosome Profiling Process
- Riboviz Workflow
- Analysis Code Refactoring
 - Example
 - Tips
 - Fails
- Call to action!
- Future Priorities

Riboviz 101

- Riboviz processes & analyses ribosome profiling data
- Ribosome profiling data helps unlock details of **active translation**: *mechanics of translation, regulation methods, translational efficiency*
- Developing/improving riboviz = **more researcher time** for biological questions rather than tinkering with pipelines & bespoke analysis code...

Polysomes to Footprints

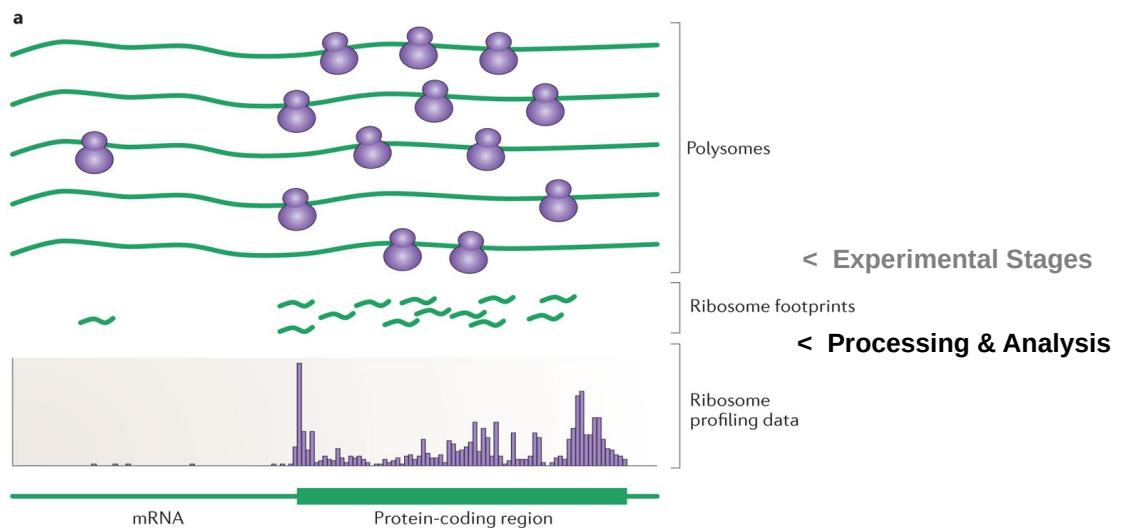


Ingolia (2014). "Ribosome profiling: new views of translation, from single codons to genome scale". Nature Reviews. Genetics. 15 (3): 205–13. doi:10.1038/nrg3645
4 / 27

- Lyse the cells to get at the mRNA molecules bound to ribosomes.
- 'Stop' the translation process: e.g. with cycloheximide or other means
- Digest the non-protected RNA using a nuclease
- Strip away the ribosomes and proteins
- Size-select for these previously 'masked' fragments of mRNA
- Add adapters
- Reverse-transcribe to complimentary DNA
- Amplify
- Sequence

Ingolia NT, Ghaemmaghami S, Newman JR, Weissman JS (2009). "Genome-wide analysis in vivo of translation with nucleotide resolution using ribosome profiling". Science. 324 (5924): 218–23. doi:10.1126/science.1168978

Footprints to Profiling Data



Ingolia (2014). "Ribosome profiling: new views of translation, from single codons to genome scale". Nature Reviews. Genetics. 15 (3): 205–13. doi:10.1038/nrg3645
5 / 27

Take these reads & submit to further processing & analysis to get plots & information.

Analysis of Ribosome Profiling Data

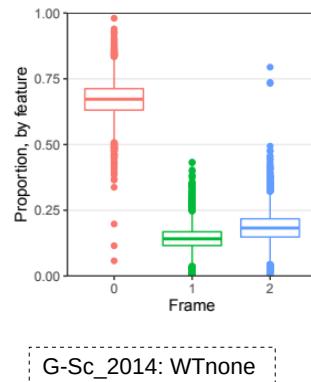
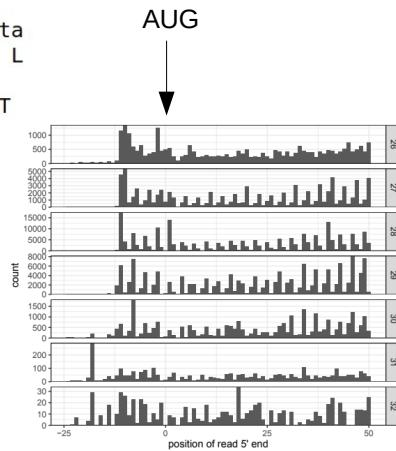
- Looking for **3-NT periodicity**: ribosomes moving along transcript 1 codon at a time
- Most reads map to **coding regions** (98.8% in Ingolia et al 2009)
- Reasonable read-lengths (e.g. know should look for appx **~28-30NT**)
- Looking for most reads to be in **one frame**

Ribosome Data & Reading Frame

reading frame:

first reading frame
second reading frame
third reading frame

123
|||
acttacccgggacta
T Y P G L
L T R D
L P G T



Reading Frame: <https://www.ncbi.nlm.nih.gov/Class/MLACourse/Original8Hour/Genetics/readingframe.html>
Data: Guydosh & Green (2014). "Dom34 rescues ribosomes in 3' untranslated regions." Cell. 156 (5): 950-62. doi: 10.1016/j.cell.2014.02.006.

7 / 27

Ribosome profiling data is really useful for investigating translational mechanics particularly because of the information it provides about the active open reading frame!

Can locate the start codon 'AUG' (from annotation position information from a .gff or sequence info) & determine what should be the reading frame based on that.

Can determine which reading frame(s) are being actively translated from the counts of reads sequenced which match each frame. Can look at if this is altered by particular characteristics of the gene or sequence for example.

Footprints to Ribosome Profiling Data

- **Processing:** *lots of steps*
 - Removing adapter sequences
 - Remove UMIs (Unique Molecular Identifiers) & barcodes if present
 - Demultiplex / Deduplicate reads if required
 - Need to filter out contaminant reads
 - Align reads to transcriptome
- **Analysis:** *more steps*
 - Analyse & quantify data:
 - Create outputs (including for quality-control, further analysis)

Riboviz Workflow: Inputs

Organism Specific

Transcript Sequences
.fasta

Genome / Transcriptome
Features
.gff

Contaminant Sequences
(rRNA)
.fasta

(Additional
Organism-Specific
Data)

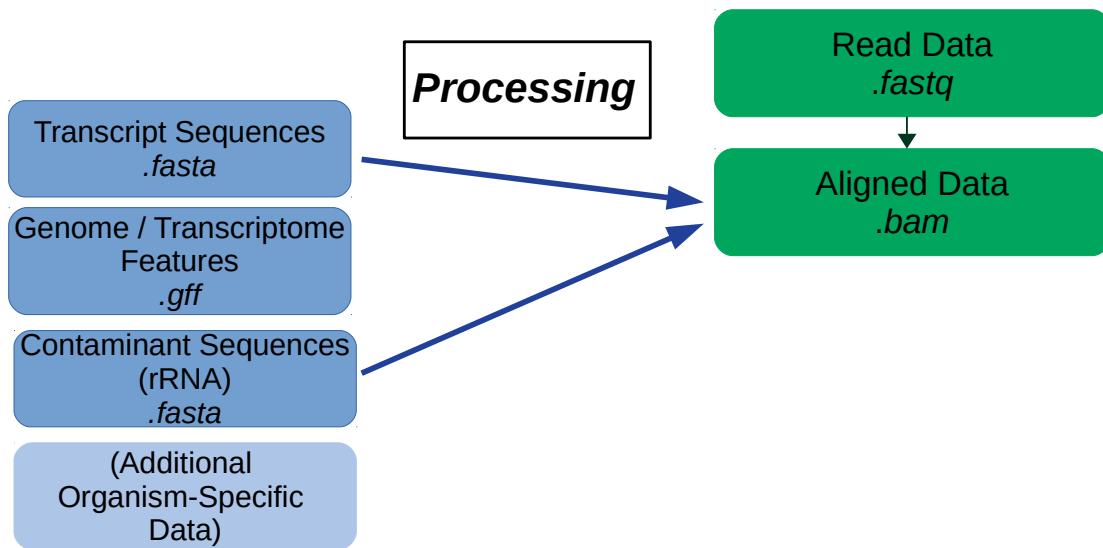
Sample Specific

Read Data
.fastq

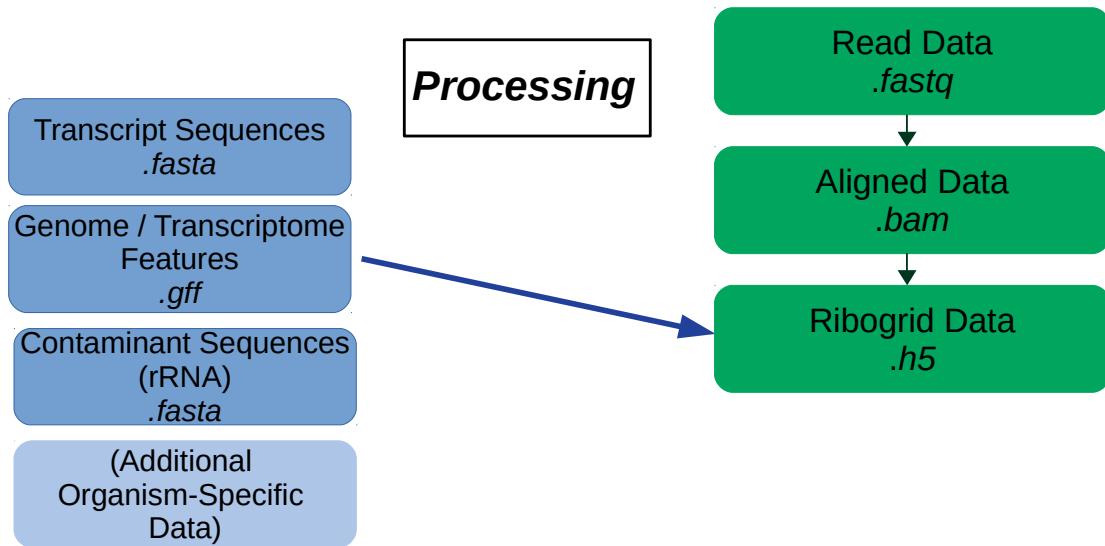
Configuration File
.yaml

Configuration File lists all files & parameters needed to run RiboViz

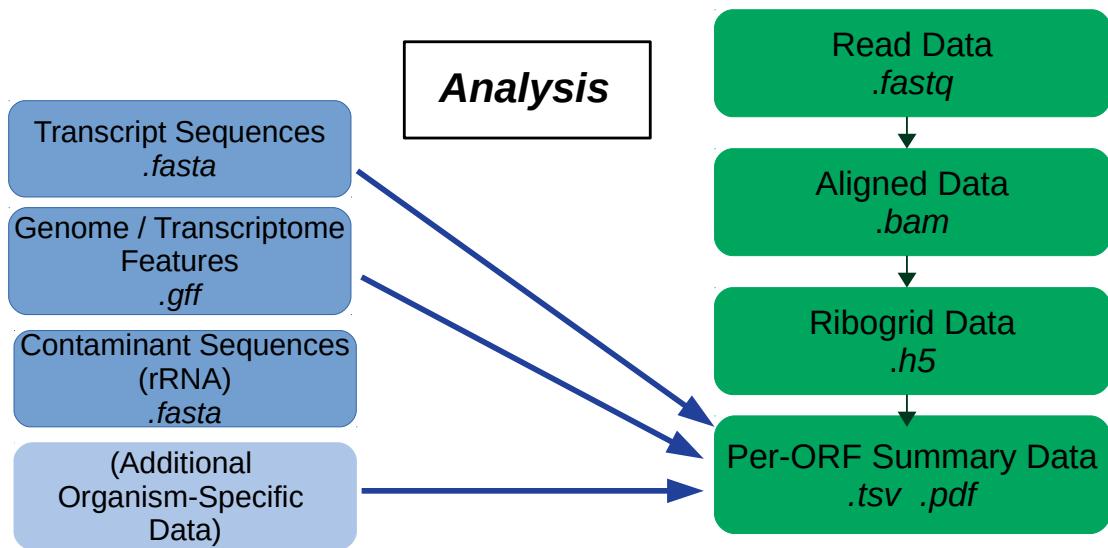
Riboviz Workflow



Riboviz Workflow



Riboviz Workflow



Analysis

generate_stats_figs.R:

Inputs:

- sample read data from .h5 file
- organism-specific transcriptome data from .gff & .fa
- parameters from configuration .yaml file

Generates summary statistics, analysis plots & quality control plots

Outputs:

- .pdf plots
- .tsv files

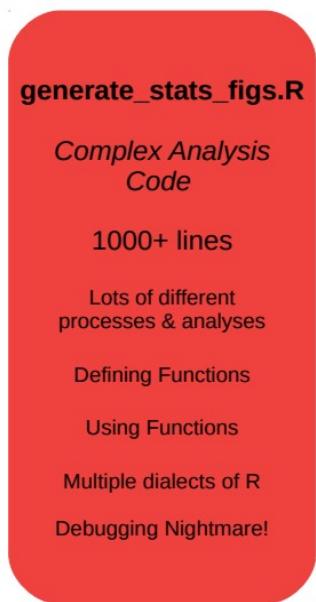
1 key script which handles this:
`generate_stats_figs.R`

WHAT IT DOES:

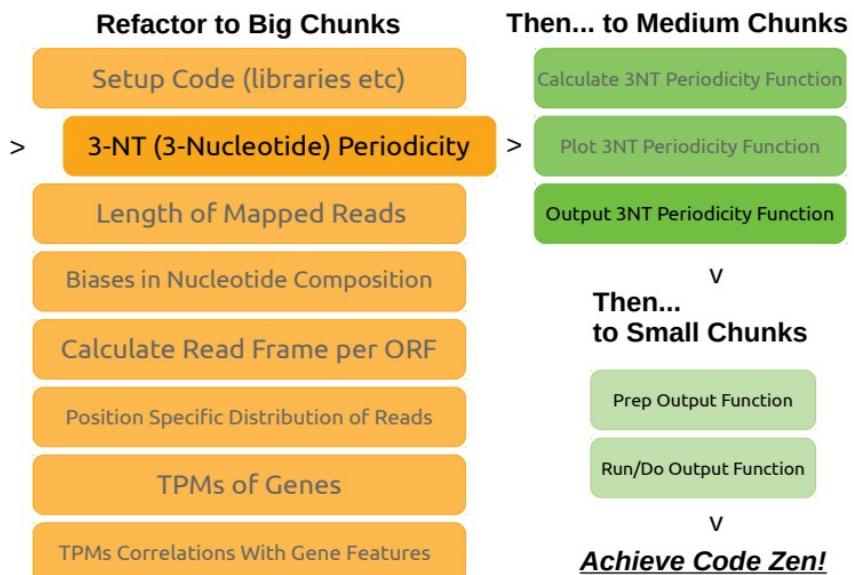
Generates summary statistics, analysis plots &
quality control plots in a sample

It's run once per sample as part of the riboviz
workflow & creates the same (comparable!) outputs
for each.

Refactoring Analysis Code



08 Apr 2020



1 / 27

- Setup Code

LARGE CHUNKS:

- 3nt periodicity
- Lengths all mapped reads
- Biases in nucleotide composition
- Calculate read frame for every orf
- RPF position specific distribution of reads
- mRNA position specific distribution of reads
- TPMs of genes
- TPMs correlations with gene features

MEDIUM CHUNKS:

- CalculateFunction - computational code
- PlotFunction - plot creation
- SaveFunction - PDF writeouts of plots
- WriteFunction - .TSV file creation

Anatomy of a Code Chunk...

- 3NT Periodicity Big Code Chunk:
 - Calculate! Get start positions & read counts at each position for each gene from the .h5 file, calculate periodicity from this matrix of positions & counts
 - Plots! {ggplot2}
 - Save plots as .pdf
 - Write information out as .tsv (includes provenance info)

```

313 ThreeNucleotidePeriodicity <- function(gene_names, dataset, hd_file, gff_df) {
314
315   # check for 3nt periodicity
316   print("Starting: Check for 3nt periodicity globally")
317
318   # CalculateThreeNucleotidePeriodicity():
319   three_nucleotide_periodicity_data <- CalculateThreeNucleotidePeriodicity(gene_names = gene_names, dataset = dataset, hd_file =
320
321   # PlotThreeNucleotidePeriodicity()
322   three_nucleotide_periodicity_plot <- PlotThreeNucleotidePeriodicity(three_nucleotide_periodicity_data)
323
324   # NOTE: repeated from inside CalculateThreeNucleotidePeriodicity() as preferred not to return multiple objects in list (hassle
325   gene_poslen_counts_5start_df <- AllGenes5StartPositionLengthCountsTibble(gene_names = gene_names, dataset= dataset, hd_file =
326
327   # run PlotStartCodonRiboGrid()
328   start_codon_ribogrid_plot <- PlotStartCodonRiboGrid(gene_poslen_counts_5start_df)
329   # creates plot object
330
331   # run SaveStartCodonRiboGrid():
332   SaveStartCodonRiboGrid(start_codon_ribogrid_plot)
333
334   # run PlotStartCodonRiboGridBar():
335   start_codon_ribogrid_bar_plot <- PlotStartCodonRiboGridBar(gene_poslen_counts_5start_df)
336   # creates plot object
337
338   # run SaveStartCodonRiboGridBar():
339   SaveStartCodonRiboGridBar(start_codon_ribogrid_bar_plot)
340
341   # run SavePlotThreeNucleotidePeriodicity():
342   SavePlotThreeNucleotidePeriodicity(three_nucleotide_periodicity_plot)
343
344   # run WriteThreeNucleotidePeriodicity():
345   WriteThreeNucleotidePeriodicity(three_nucleotide_periodicity_data)
346
347   print("Completed: Check for 3nt periodicity globally")
348
349 } # end ThreeNucleotidePeriodicity() function definition
350 # run ThreeNucleotidePeriodicity():
351 ThreeNucleotidePeriodicity(gene_names, dataset, hd_file, gff_df)

```

16 / 27

3-NT Periodicity Big chunk code.

- **Easy to read** (maybe not on this screen, sorry)
- Functions & objects **different styling** (not my focus just now in this talk but lots of interesting / helpful stuff to say about it)
- **Print statements** help show where we're at in the logs if something goes wrong
- If things DO go wrong, **error messages benefit from middle-size functions with clear naming**
- Aiming to go with nice tidyverse stuff: e.g. consistent variable names
- **{magrittr} pipes (%>%)** where they help make code more understandable vs nested(functions(in(functions)))
- **Returning things explicitly** using return() to avoid unexpected behaviour! Also helps highlight the important objects.
- Writing out as .tsv also super helpful because **regression tests can't compare content of .pdfs**, just if .pdfs exist, but CAN compare values in .tsv files – if anything changes, it's easy to know!
- **NOTE:** the medium level functions are defined before this chunk or it doesn't work :)

```

188 CalculateThreeNucleotidePeriodicity <- function(gene_names, dataset, hd_file, gff_df){
189
190   # get gene and position specific total counts for all read lengths
191   gene_poslen_counts_5start_df <- AllGenes5StartPositionLengthCountsTibble(gene_names = gene_names, dataset = dataset)
192
193   # summarize by adding different read lengths
194   gene_pos_counts_5start <- gene_poslen_counts_5start_df %>%
195     group_by(Pos) %>%
196     summarize(Counts = sum(Counts))
197
198   # gives:
199   # > str(gene_pos_counts_5start)
200   # Classes 'tbl_df', 'tbl' and 'data.frame': 75 obs. of 2 variables:
201   # $ Pos : int -24 -23 -22 -21 -20 -19 -18 -17 -16 -15 ...
202   # $ Counts: int 285 318 307 386 291 347 849 338 475 355 ...
203
204   gene_pos_counts_3end <- gene_poslen_counts_3end_df %>%
205     group_by(Pos) %>%
206     summarize(Counts = sum(Counts))
207
208   # gives:
209   # > str(gene_pos_counts_3end)
210   # Classes 'tbl_df', 'tbl' and 'data.frame': 75 obs. of 2 variables:
211   # $ Pos : int -49 -48 -47 -46 -45 -44 -43 -42 -41 -40 ...
212   # $ Counts: int 198930 13923 56280 19458 12573 46912 19043 13282 36968 20053 ...
213
214   three_nucleotide_periodicity_data <- bind_rows(
215     gene_pos_counts_5start %>% mutate(End = "5"),
216     gene_pos_counts_3end %>% mutate(End = "3")
217   ) %>%
218     mutate(End = factor(End, levels = c("5", "3")))
219
220   # gives:
221   # > str(three_nucleotide_periodicity_data)
222   # Classes 'tbl_df', 'tbl' and 'data.frame': 150 obs. of 3 variables:
223   # $ Pos : int -24 -23 -22 -21 -20 -19 -18 -17 -16 -15 ...
224   # $ Counts: int 285 318 307 386 291 347 849 338 475 355 ...
225   # $ End : Factor w/ 2 levels "5","3": 1 1 1 1 1 1 1 1 1 1 ...
226
227   return(three_nucleotide_periodicity_data)
228 }
229
230 # end CalculateThreeNucleotidePeriodicity() definition
231 # gives:
232 # > str(CalculateThreeNucleotidePeriodicity(gene_names = gene_names, dataset = dataset, hd_file = hd_file, gff_df = gff_df))
233 # A tibble: 150 x 3
234 #   Pos    Counts End
235 #   <int> <int> <fct>
236 #     1      285 5'
237 #     2      318 5'
238 #     3      307 5'
239 #     4      386 5'
240 #     5      291 5'
241 #     6      347 5'
242 #     7      849 5'
243 #     8      330 5'
244 #     9      475 5'
245 #    10      355 5'
246 # ... with 140 more rows
247
248

```

17 / 27

CALCULATE (medium) block does all the heavy lifting.

This defined above the ‘big’ block function (necessary).

Lots of functions used here which are defined in (small blocks in) read_count_functions.R script & sourced in - this keeps the structure of this script quite easy to read.

NOTE: I found it handy to put example output as comments to give an idea of the objects returned & what they might look like. Quite useful while refactoring, but maybe not necessary longer-term & can be removed with a styling pass.

```

238 # define PlotThreeNucleotidePeriodicity() function with reasonable arguments
239 PlotThreeNucleotidePeriodicity <- function(three_nucleotide_periodicity_data){
240
241   # Plot
242   three_nucleotide_periodicity_plot <- ggplot(
243     three_nucleotide_periodicity_data,
244     aes(x = Pos, y = Counts)) +
245     geom_line() +
246     facet_wrap(~End, scales = "free") +
247     labs(x = "Nucleotide Position", y = "Read counts")
248
249   return(three_nucleotide_periodicity_plot)
250 }
251 # end PlotThreeNucleotidePeriodicity() definition
252
253 # potentially replace/tweak plot_ribogrid() to follow StyleGuide
254 PlotStartCodonRiboGrid <- function(gene_poslen_counts_5start_df){
255
256   # Function to do the ribogrid & ribogridbar plots?
257   # ribogrid_Start
258   start_codon_ribogrid_plot <- plot_ribogrid(gene_poslen_counts_5start_df)
259   return(start_codon_ribogrid_plot)
260 }
261 # end PlotStartCodonRiboGrid() definition
262
263 SaveStartCodonRiboGrid <- function(start_codon_ribogrid_plot){
264
265   # Function to do the ribogrid & ribogridbar plots?
266   # ribogrid_Start
267   start_codon_ribogrid_plot %>%
268     ggsave(
269       filename = file.path(output_dir, paste0(output_prefix, "startcodon_ribogrid.pdf")),
270       width = 6, height = 3
271     )
272   #return() # no return as writing-out
273 }
274 # end SaveStartCodonRiboGrid() definition
275
276 SaveStartCodonRiboGridBar <- function(start_codon_ribogrid_bar_plot){
277
278   start_codon_ribogrid_bar_plot %>%
279     ggsave(
280       filename = file.path(output_dir, paste0(output_prefix, "startcodon_ribogridbar.pdf")),
281       width = 6, height = 5
282     )
283   #return() # no return as writing-out
284 }
285 # end SaveStartCodonRiboGridBar() definition
286
287 SavePlotThreeNucleotidePeriodicity <- function(three_nucleotide_periodicity_plot) {
288
289   # Save plot and file
290   ggsave(
291     three_nucleotide_periodicity_plot,
292     filename = file.path(output_dir, paste0(output_prefix, "3nt_periodicity.pdf"))
293   )
294   # return() # NO RETURN as writing out
295 }
296 # end of function definition SavePlotThreeNucleotidePeriodicity()

```

18 / 27

PLOT & SAVE blocks

Same kind of thing: medium block where we've got plotting code for each type of plot as a separate function.

'Design choices': when multiple blocks look quite similar (e.g. SaveStartCodonRiboGrid + SaveStartCodonRiboGridBar which save out ggplot objects) can be difficult to know when to make 1 function with differing parameters, or keep as 2 separate functions which do almost the same thing...

Decided in this case to keep separate & avoid losing time 'over-optimising', also easy to read so possibly not necessary to fix what ain't broke :)

```
~~~  
296 WriteThreeNucleotidePeriodicity <- function(three_nucleotide_periodicity_data) {  
297   tsv_file_path <- file.path(output_dir, paste0(output_prefix, "3nt_periodicity.tsv"))  
298   write_provenance_header(path_to_this_script, tsv_file_path)  
299   write.table(  
300     three_nucleotide_periodicity_data,  
301     file = tsv_file_path,  
302     append = T,  
303     sep = "\t",  
304     row.names = F,  
305     col.names = T,  
306     quote = F)  
307   # return()? NO RETURN  
308 } # end of function definition WriteThreeNucleotidePeriodicity()  
309
```

19 / 27

WRITE medium block

Nice legible chunk of code: we've got a function with lots of parameters – `write.table()` - written across multiple lines to avoid going over recommended 80 chars length

Makes it easy to read & quick to identify/change any of those parameters.

Good to do that generally!

Note: `#return()?` comment handy note to self while I was thinking about what each block should output

Note: I included some “`# end of function ...`” comments at the end after the last ‘`}`’ as it lets me see the end of that block clearly – something that really helped working on a lot of long code blocks. Stops me getting lost while scrolling, too!

Again, could be easily removed during a re-style.

Refactoring Tips

- Break it down into **chunks!**
- **Regression tests** are your friend
- Develop functions in your main script & then move into **separate functions script**
- Keep it specific: **issue-focussed working**
... Decide how you'll know you're finished...

20 / 27

Think about WWMJD: What Would Mike Jackson Do?

Break it down: Big chunks, Medium chunks, Small chunks, success!

Otherwise you spend hours looking bleakly at a page of impenetrable code & have no idea how to make a dent in it...

Regression Tests: SO useful at telling you something has suddenly stopped working or gives different answers!

Functions in another script, called from the ‘main’ one. Avoids visual clutter, keeps the script easy to read & understand. Work in main first: easier to write them in the same script then move rather than troubleshooting everything + environment issues

Specific **issue tickets** & issue **branches** keep you focussed on ONE thing: open branch, do work, merge branch, close issue, move on.

A good issue ticket tells you how you'll **know when it's done**, also... Or it's easy to keep tinkering forever... Refactoring turns into styling & then into adding more testing & so on...

Refactoring LMFs

- **Issue proliferation**... Getting lost amongst different issues & losing sight of the main goals
- Schroedinger's **debugging** issues!
- **Package::function()** is very useful!
- ... Remember to **tell collaborators** about & document the new packages you add to the code to solve problems...

21 / 27

Issue proliferation: I got lost and delayed as I tried fixing other issues as they came up & the issues THOSE issues created and... MADNESS. Avoid.

Interactive function behaviour: some functions behave differently when run interactively vs in a sourced script... Makes debugging SUPER tricky!

Package::function() You WILL forget which snazzy package that important function came from otherwise

Also you will probably forget add it to the documentation and tell people to install it and There Will Be Errors.

Fails are not ALWAYS a failure!

- <3 Regression tests!
- Which output is the **correct** output? How do we know?
- How to decide when to make a **new issue** or keep working on a problem?
- **Rollback...**
- Importance of **code testing**... & understandable code!

22 / 27

If the original code wasn't tested, how do we know which output is *correct* when we're comparing original output to new output from new code?

Currently we've got 1 section of 'old' code which we've had to roll back to because 'new' code produces different outputs & we need to investigate this!

This is now a new issue, instead of holding back the rest of the refactoring work.

Importance of code testing... & understandable code!

“Putting the YOU into USER”

- **User Testing & User Interviews** – *Sam!*
- Existing **Documentation Feedback** – *Siyin* & others already helping with this :)
- Leave the **Code Docs** to the Robots? – {roxygen2} – hoping to chat to *Edward* about this!
- **General Comparisons** - what features of other software doing similar things do you find useful/unhelpful? - **Everyone!!!**

23 / 27

User Testing & Interviews:

- First impressions
- Ease/difficulties of use
- Analyses you want to see
- What doesn't work / isn't helpful?

Getting folks to feedback on & improve existing documentation: learned quite a lot from Ania, & now Siyin, & other new users!

Documentation using {roxygen2}:

- Better code documentation is better for users (if nothing else, it ensures developers really THINK about their code before releasing it into the wild!!)

What processing/analysis software tools do you use now that you find useful?

- What features => useful / annoying?
- Any generalisations we can take to riboviz work?

“Testable, Reliable CompYOUtation”

- **Code Testing:**

- Helping integrate functions from *Ania & Siyin's* projects: want to have good testable code...
- **{testthat} R package** – would be great to chat with *Xuejia* on how this works

- **More data!**

- This lab: *Rosey?*
- Others? Let's talk :)

Priorities

- **Tests** for the analysis code
- generate_stats_figs.R **finishing touches**: Documentation / better output format / styling
- **New datasets** run & added to example datasets repository
- Integrate **new functions** from Ania & Siyin's project work
- **User focus**
- Q: How does riboviz compare with **other tools**
- Q: Do we have the right **statistics** for diagnostics?

25 / 27

- Would be good to compare riboviz with other tools & get a feeling for how easy they are to use, what outputs they give & whether their results 'match' - something I'd like to do ahead of a riboviz paper!
- Hard to know without knowing what more users expect from riboviz and what questions they're trying to answer with it. More idea of this as we get more users onboard (esp. user interviews etc)
-

Thanks / Acknowledgements

- BBSRC-NSF funded project
- Collaborative project:
 - Edward Wallace: *The Wallace Lab*, The University of Edinburgh.
 - + Siyin Xue, Ania Kurowska
 - Premal Shah, John Favate, Tongji Xing: *The Shah Lab*, Rutgers University.
 - Liana Lareau, Amanda Mok: *The Lareau Lab*, University of California, Berkeley.
 - Kostas Kavousannakis, Mike Jackson: *EPCC*, The University of Edinburgh.
 - Oana Carja, Joshua Plotkin: The University of Pennsylvania

Questions?

EXTRA SLIDES

Process ribosome profiling sample data

If sample files (`fq_files`) are specified, then the workflow processes the sample files as follows:

1. Read configuration information from YAML configuration file.
2. Build hisat2 indices if requested (if `build_indices: TRUE`) using `hisat2 build` and save these into the index directory (`dir_index`).
3. Process each sample ID-sample file pair (`fq_files`) in turn:
 - i. Cut out sequencing library adapters (`adapters`) using `cutadapt` .
 - ii. Extract UMIs using `umi_tools extract` , if requested (if `extract_umis: TRUE`), using a UMI-tools-compliant regular expression pattern (`umi_regex`). The extracted UMIs are inserted into the read headers of the FASTQ records.
 - iii. Remove rRNA or other contaminating reads by alignment to rRNA index files (`rrna_index_prefix`) using `hisat2` .
 - iv. Align remaining reads to ORFs index files (`orf_index_prefix`). using `hisat2` .
 - v. Trim 5' mismatches from reads and remove reads with more than 2 mismatches using `trim_5p_mismatch` .
 - vi. Output UMI groups pre-deduplication using `umi_tools group` if requested (if `dedup_umis: TRUE` and `group_umis: TRUE`)
 - vii. Deduplicate reads using `umi_tools dedup` , if requested (if `dedup_umis: TRUE`)
 - viii. Output UMI groups post-deduplication using `umi_tools group` if requested (if `dedup_umis: TRUE` and `group_umis: TRUE`)
 - ix. Export bedgraph files for plus and minus strands, if requested (if `make_bedgraph: TRUE`) using `bedtools genomecov` .
 - x. Write intermediate files produced above into a sample-specific directory, named using the sample ID, within the temporary directory (`dir_tmp`).
 - xi. Make length-sensitive alignments in compressed h5 format using `bam_to_h5.R` .
 - xii. Generate summary statistics, and analyses and QC plots for both RPF and mRNA datasets using `generate_stats_figs.R` . This includes estimated read counts, reads per base, and transcripts per million for each ORF in each sample.
 - xiii. Write output files produced above into an sample-specific directory, named using the sample ID, within the output directory (`dir_out`).
4. Collate TPMs across results, using `collate_tpms.R` and write into output directory (`dir_out`). Only the results from successfully-processed samples are collated.
5. Count the number of reads (sequences) processed by specific stages if requested (if `count_reads: TRUE`).

<https://github.com/riboviz/riboviz/blob/develop/docs/user/prep-riboviz-operation.md>

29 / 27

*

```
11
12 # Handle interactive session behaviours or use get_Rscript_filename():
13 if (interactive()) {
14   # Use hard-coded script name and assume script is in "rscripts"
15   # directory. This assumes that interactive R is being run within
16   # the parent of rscripts/ but imposes no other constraints on
17   # where rscripts/ or its parents are located.
18   this_script <- "generate_stats.R"
19   path_to_this_script <- here("rscripts", this_script)
20   source(here::here("rscripts", "provenance.R"))
21   source(here::here("rscripts", "read_count_functions.R"))
22 } else {
23   # Deduce file name and path using reflection as before.
24   this_script <- getopt::get_Rscript_filename()
25   path_to_this_script <- this_script
26   source(here::here("rscripts", "provenance.R"))
27   source(here::here("rscripts", "read_count_functions.R"))
28 }
29
```

30 / 27

get_Rscript_filename() useful function which produces the name of the script being run.

It doesn't act the same way if it's being run interactively, which

Updates! UX

- Users:
 - Building user base?
 - Needs / wants?
 - Oven-ready datasets*?

* <https://github.com/riboviz/example-datasets>

31 / 27

Previously spoke about Challenges: UX & Testing Reliability. Now checking in on those.

UX:

- Building user base I'm not sufficiently active enough on that, but our collaborators are currently running datasets with version 2.0-beta ahead of v2.0 release!
- New users (inc Siyin!) posting questions & issues on github – that's a good sign of the start of a building community!
- What do users need/want? Haven't survey'd anyone yet... Qs for Sam! :)
- Onboarding with oven-ready datasets? YES! Example-datasets repo, PR for a Cryptococcus dataset Siyin has just run on the Eddie cluster Progress!

Updates! UX

- Support & Documentation:
 - Existing docs suitable?
 - Document outputs better?
 - Translation UK workshop?

32 / 27

UX contd.: Support & Docs:

- Had comments from Ania, now Siyin reading & giving feedback, adding to & improving.
- New documentation like running on Eddie (UoE-specific, certainly, but really useful to know how to set up on any cluster, potentially help other researchers do the same on theirs?)
- Documentation of outputs? Still on my TODO list, but NOW I've worked through refactoring, it's going to be easier to document. Also, #32 led to fixing #155 (.h5 issue) & now fixed the .Rmd issue, so can start to work on getting markdown output & win a 1-page .html option ?
- Translation UK sadly postponed until 2021...
- Version 2.0-beta? Out now, & working towards v2.0 release! Will include the refactored code!

Updates! Testing

- General Testing:
 - New feature development ongoing
 - Bug fixes
 - Code reviews happening
 - New datasets

33 / 27

- New features like Nextflow workflow implementation alongside (sort of original) python implementation!
- .h5 file access more reliable & can now be parallelised (e.g. plays better with clusters like Eddie)
- Pull requests reviewed before being merged (more than one person checks it works), lots of this happened recently – very exciting to be part of tbh
- New datasets through means new potential data weirdness that might break riboviz. Want to know it works over a wide range of different organisms/ data sizes/etc & will work for other groups' data & not just this lab.
 - e.g. Mouse data? Drosophila? Bacteria? Human? Transcriptome vs Genome files...

Updates! Testing

- Methods Testing:
 - Regression tests
 - Expected outputs: simulated data.

34 / 27

As I mentioned, the regression tests have been super useful at showing what's different between old & new code

But, this has underlined that we didn't have tests for the code in the first place. Some investigations to be done on some of the analysis code to confirm whether 'new' code is more correct or not...?

{testthat} unit testing R package for future? Would be really helpful to implement this for the generate_stats_figs.R code to make sure it DOES do what we expect! (Plus it promises to "make testing as fun as possible", so who could say no?!)

Close to getting the Lareau Lab simulated ribosome profiling data through riboviz: going to be interesting to see what this shows us about riboviz!

First Things First: Setup

- Initial Setup Code:
 - load libraries
 - handle interactive session behaviours if required*
 - source provenance & functions scripts
 - load parameters passed in from main riboviz workflow
 - read in key files (.gff, .fa, .h5)

35 / 27

- Didn't need too much work
- Some libraries in main script, most in another sourced script (`read_count_functions.R`)
- `SuppressMessages(library(...))` avoids lots of loading messages
- Talk a bit about the interactive session behaviours issue shortly
- Parameters come from configuration `.yaml` file & are passed in to `generate_stats_figs.R` via riboviz pipeline (python script or nextflow workflow)
- Parameters handled using option parsing, allows us to have set defaults
- Read in: .gff info on sequence annotations, .fa has the coding sequences, .h5 has the sample reads