QC and downstream analysis of the Rp-Bp results

Rp-Bp includes a number of additional scripts for quality control and downstream analysis.

- Creating read length-specific profiles
- Counting and visualizing reads filtered at each step
- Creating and visualizing read length distributions
- Visualizing read length metagene profiles

Creating read length-specific profiles

As described in the usage instructions, Rp-Bp writes the unsmoothed ORF profiles to a matrix market file. This profile merges reads of all lengths.

The `create-read-length-orf-profiles` script can be used to create profile files which also include counts of individual read lengths.

```
create-read-length-orf-profiles <config> <sample or condition name> <out> [--is-condition]
```

Command line options

- `config`. A yaml config file
- `sample or condition name`. The name of either one of the `riboseq_samples` or `riboseq_biological_replicates` from the config file
- `out`. The output (txt.gz) file, containing the read-length specific profiles. The format is a sparse coordinate format inspired by the matrix market format. See below for details about the output format.
- [`--is-condition`]. If the `sample or condition name` is a condition, that is, if it is a key from `riboseq_biological_replicates`, then this flag must be given.

Additionally, the command can be given logging and slurm options.

Output format

Each line in the output file is a tuple containing the following values.

- `read_length`. The (trimmed) read lengths for this position.
- `orf_num`. An identifier which maps to `orf_num` in the (static) list of ORFs for the reference, `<genome_base_path>/transcript-index/<genome_name>.genomic-orfs.<orf_note>`

- `orf_position`. The base-0 position with respect to the spliced transcript (so `position % 3 == 0` implies the position is in-frame)
- `read_count`. The sum of counts across all replicates for the condition (if `--is-condition` is given) or the single sample (otherwise) after adjusting according to P-sites and removing multimappers.

Counting and visualizing reads filtered at each step

Counting

The `get-all-read-filtering-counts` script counts reads filtered at each step of the preprocessing pipeline.

This script requires `samtools` to be present in `$PATH`.

```
get-all-read-filtering-counts <config> <out> [--num-cpus <num_cpus>]
```

Command line options

- `config`. A yaml config file
- `out`. The output file, in csv.gz format. See below for details.
- [`--num-cpus`]. The script is parallelized at the sample level. If specified, this many samples will be processed at once.

Output format

The output is a "wide" data frame which contains one row for each sample. The fields are as follows.

- `note`. The name of the sample.
- `raw_data_count`. The number of reads in the original fastq files.
- `without_adapters_count`. The number of reads remaining after running `flexbar` to remove adapters and low-quality reads.
- `without_rrna_count`. The number of reads remaining after removing ribosomal and other reads with `bowtie2`.
- `genome_count`. The number of reads with at least one genome alignment.
- `unique_count`. The number of reads with exactly one genome alignment.
- `length_count`. The number of uniquely mapping reads which also have a "periodic" read length, as determined by BPPS.

Visualizing (script)

The `visualize-read-filtering-counts` script visualizes the read counts from `get-all-read-filtering-counts`.

```
visualize-read-filtering-counts <read_counts> <out> [--without-rrna] [--title <title>] [--fo
```

Command line options

- `read_counts`. The output from `get-all-read-filtering-counts`
- `out`. The output image file. The extension should be something recognized by matplotlib, such as `png` or `pdf`.
- [`--without-rrna`]. If this flag is given, then the bar chart will not include reads filtered due to low quality or mapping to ribosomal sequences.
- [`--title`]. A title placed at the top of the plot
- [`--fontsize`]. The fontsize used for most of the text on the plot, including the tick labels (sample names and read counts), axis labels and title.
- [`--legend-fontsize`]. The fontsize to use for the entries in the legend (the filtering steps).
- [`--ymax`]. The maximum number of reads displayed on the y-axis. Typically, this value should be around 10% higher than the largest read count. However, some other value may be more appropriate if one of the samples has many more reads than the others.
- [`--ystep`]. The frequency of tick marks on the y-axis.

Visualizing (ipython notebook)

The `notebooks/preprocessing/create-read-filtering-bar-chart` notebook can be used to visualize the read counts. It functionality is essentially the same as the `visualize-read-filtering-counts` script; however, the properties of the plot, such as the exact location of the legend, are much easier to manipulate in the notebook.

Additionally, the notebook will attempt to use the `riboseq_sample_name_map` from the config file to find "pretty" names for the samples. In particular, this should be a map from the sample name given in the `riboseq_samples` to a string that will be used for the x-tick labels in the plot. If a sample name is not present in the name map, it will be left unchanged.

Control variables

In the third cell, the `config_files`, `alignment_counts_files`, `out_files` and `without_rrna_files` dictionaries must be updated to include the relevant files. The key in the dictionary should be the same for all of the new files.

In the fourth cell, the `data` variable should be changed to the key used in the dictionaries. The other variables (`without-rrna`, etc.) have the same interpretation as for the script.

In the sixth cell, visualization aspects such as the colors, legend location, figure size, etc., can be set using the respective matplot lib options.

Example visualization

Creating and visualizing read length distributions

Creating distributions

The `get-read-length-distribution` script (part of the misc package) counts the number of reads of each length in a given bam file. It can be used to count the read length distribution for both all aligned reads and only uniquely-aligning reads.

N.B. The script handles multi-mappers to ensure they only contribute to the counts once.

```
get-read-length-distribution <bam_1> [<bam_2> ...] -o/--out <length-counts.csv.gz> [-p/--num
```

Command line options

- `bam_i`. The bam files which contain the aligned reads.
- `out`. The output file, in csv.gz format, which contains the counts. See below for the column specifications.
- [`--num-cpus`]. The number of CPUs to use; this many files will be processed at once.

Output format

The output is a "long" ("tidy") data frame with the following fields.

- `basename`. The name of the bam file, excluding the "`.bam`" extension.
- `length`. The read length.
- `count`. The number of reads of that length in the indicated file.

Visualizing the distributions (script)

The `plot-read-length-distribution` script creates a bar chart of the counts from `get-read-length-distribution`.

```
plot-read-length-distribution <distribution> <basename> <out> [--title <title>] [--min-read-
```

Command line options

- `distribution`. The csv file created by `get-read-length-distribution`.
- `basename`. The `basename` to visualize.
- `out`. The output (image) file
- [`--title`]. The title of the plot.
- [`--{min, max}_read_length`]. The minimum and maximum read lengths to include in the plot, inclusive. Defaults: [22, 35]
- [`--ymax`]. The maximum value for the y-axis. Default: 1.5e6+1
- [`--fontsize`]. The size of the fonts for the title, axis labels and ticks

Visualizing the distributions (ipython notebook)

The `notebooks/preprocessing/create-read-length-distribution-bar-chart` notebook can be used to visualize the read counts. Its functionality is essentially the same as `plot-read-length-distribution`; however, the properties of the plot, such as the colors, are much easier to manipulate in the notebook.

Additionally, the notebook will attempt to use the `riboseq_sample_name_map` from the config file to find "pretty" names for the samples. In particular, this should be a map from the sample name given in the `riboseq_samples` to a string that will be used for the x-tick labels in the plot. If a sample name is not present in the name map, it will be left unchanged.

Control variables

All of the relevant control variables in the third cell should point to the appropriate files.

Example visualization

Visualizing read length metagene profiles

As described in the usage instructions, metagene profiles for each read lengths are created as a part of the pipeline. These can be visualized with the `create-read-length-metagene-profile-plot` script. In particular, it shows the reads aligned around the annotated translation initiation and termination sites.

```
create-read-length-metagene-profile-plot <metagene_profile> <length> <out> [--title <title>]
```

Command line options

- `metagene_profile`. The metagene profile file (`<riboseq_data>/metagene-profiles/<sample-name>[.<`
- `length`. The length to visualize
- `out`. The output (image) file
- [`--title`]. A title for the plot
- [`--xlabel-{start,end}`]. Messages to include beneath the x-axis around the initiation and termination sites, respectively. Defaults: "Position of P-site relative to start (nt)\nRed: TIS. Green: TIS -12", "Position of P-site relative to stop (nt)\nBlue: Translation termination"
- [`--ylabel`]. The label for the y-axis. Default: "Read count (starting at bp x)"
- [`--step`]. The step size for the x-axis. Default: 10
- [`--font-size`]. The font size for everything in the plot. Default: 15
- [`--{start,end}-{up,down}stream`]. The position (in bp) to start and end the visualization around the translation initiation (`start`) and terminiation (`end`) sites. **N.B.** The `upstream` options must be negative, and the `downstream` options must be position. Defaults: [-50, 21] (for both)
- [`--use-entire-profile`]. If this option is given, then the entire profile in the file will be used, rather than the positions given by the other options.

There is not currently an ipython notebook to create these plots.

Example visualization