piquant Documentation

Release 1.0

Owen Dando

Jun 23, 2017

Contents

piquant is a pipeline to help assess the accuracy of quantification of transcripts from RNA-sequencing data.

Contents:

CHAPTER 1

Introduction

piquant is a pipeline to help assess the accuracy of the quantification of transcripts from RNA-sequencing data.

RNA-sequencing has become an important technique in cellular biology for characterising and quantifying the transcriptome, and many bioinformatics methods have been developed to reconstruct transcripts from RNA-seq data and then estimate their abundances. Gene expression estimates calculated by these methods have been shown to be relatively robust. However, at the level of transcripts, problems arising from the ambiguous origin of short RNA-seq reads and from bias in their sequence composition are compounded, and thus estimates of isoform abundance may be less accurate. It is therefore useful to be able to assess the conditions under which different transcriptome quantification tools perform well or more poorly, and how the many optional parameter choices available for each tool may affect their performance.

piquant is a pipeline of python scripts to help assess the accuracy of transcriptome quantification tools. In its first stage, RNA-seq reads are simulated from a starting set of transcripts with known abundances, under specified combinations of sequencing parameters: for example, different read lengths and sequencing depths, single- and paired-end reads, reads with or without sequencing errors, and reads with or without sequence bias. In the second stage, a number of transcriptome quantification tools (or the same tool with different optional parameter choices) estimate isoform abundances for each set of simulated reads. Finally, the isoform expression estimates calculated by each tool for each data set are compared to the known transcript abundances used to generate the reads. The comparative accuracy of expression estimates calculated by each tool can then be assessed as sequencing parameters change, or for different groups of transcripts segregated by particular transcript classification measures, via a range of automatically generated statistics and graphs.

CHAPTER 2

Overview

For a poster overview of *piquant*, see `here`.

The *piquant* pipeline consists of three main stages:

1. Simulate RNA-seq reads under specified combinations of sequencing parameters.
2. Run a number of transcriptome quantification tools (or the same tool with different optional parameter choices) on each set of simulated reads to estimate isoform abundances.
3. Generate statistics and graphs to assess and compare the performance of each quantification tool.

All three stages of the pipeline can be run via different commands of the `piquant` script. They are described in more detail below.

Simulate reads

Simulation of RNA-seq reads proceeds in two steps. In the first, run via the `piquant` command `prepare_read_dirs`, directories are prepared in which reads will be simulated; each directory corresponds to a particular combination of sequencing parameters:

- depth of sequencing
- length of reads
- single- or paired-end reads
- reads with or without errors
- reads with or without sequence bias
- strand-specific or unstranded reads
- presence or absence of background read "noise"

In the second step, RNA-seq reads are simulated. Each directory created in the first step contains a script which, when run, will use the *Flux Simulator* RNA-seq experiment simulator *[FluxSimulator]* to generate an expression profile for transcripts, then simulate reads for those transcripts according to the specified combination of sequencing

parameters. This script can be run directly; however, using the `piquant` command `create_reads`, reads for several combinations of sequencing parameters can be simulated at once as a batch.

The `piquant` command `check_reads` provides an easy way to check that the read simulation processes completed correctly for specified combinations of sequencing parameters.

Quantify transcripts

Quantification of transcripts proceeds in three steps. In the first, run via the `piquant` command `prepare_quant_dirs`, directories are prepared in which quantification results will be produced. For each combination of sequencing parameters for which reads were simulated, there will be such a directory for each quantification tool (or, alternatively, for each different combination of quantification tool parameters that are being assessed).

In the second step, the `piquant` command `prequantify` runs, for each quantification tool, commands that only need to be executed once, regardless of how many different sets of simulated reads are being used for quantification. For example, such commands might include creating a *Bowtie* *[Bowtie]* index for the genome to which reads will be mapped, or deriving FASTA sequences for the transcripts whose abundance is being measured.

Finally, transcript abundances are estimated using specified transcriptome quantification tools. Each directory created by the command `prepare_quant_dirs` contains a script which, when run, will use a particular tool to estimate isoform expression for a particular set of simulated reads. As for the case of creating reads, this script can be run directly if necessary; however, the `piquant` command `quantify` allows a number of such scripts to be run simultaneously as a batch.

The `piquant` command `check_quant` provides an easy way to check that the transcript quantification processes completed correctly for specified combinations of tools and read sequencing parameters.

Assess quantification accuracy

In the final stage of the pipeline, run via the `piquant` command `analyse_runs`, data describing quantification accuracy for specified combinations of sequencing parameters and quantification tools are assembled, and statistics and graphs are generated by which comparative performance can be assessed. In addition, by default, graphs are produced comparing the time and memory resource usage of the different quantification tools during the prequantification and quantification steps.

Requirements

The *piquant* pipeline is implemented as a set of Python scripts and modules; it has currently been tested against Python versions 2.7.6 and 3.4.0 running under Ubuntu 12.04.4 LTS and 14.04.1 LTS.

In order to simulate reads, the *Flux Simulator* RNA-seq experiment simulator is required to be installed, and the `flux-simulator` executable be added to the executable path (e.g. via the Unix PATH variable). *piquant* has been tested with Flux Simulator version 1.2.2.

By default, *piquant* has the ability to run six different quantification tools:

- *Cufflinks*: *[Cufflinks]*
- *RSEM*: *[RSEM]*
- *eXpress*: *[eXpress]*
- *Sailfish*: *[Sailfish]*

- *Salmon*: *[Salmon]*
- *Kallisto*: *[Kallisto]*

and these tools are required to be installed, and their relevant executables added to the executable path, if they are to be used within *piquant*. The pipeline has been tested with the following versions of these quantification tools:

- *Cufflinks*: version 2.2.1
- *RSEM*: version 1.2.19
- *eXpress*: version 1.5.1
- *Sailfish*: version 0.8.0
- *Salmon*: version 0.5.1
- *Kallisto*: version 0.42.4

In addition, the use of each quantification tool within the *piquant* pipeline has additional dependencies, which are enumerated below:

- *Cufflinks*: *Bowtie [Bowtie]* and *TopHat [TopHat]* are required to map simulated reads to the genome.
- *RSEM*: *Bowtie* is required by RSEM to map simulated reads to the transcriptome.
- *eXpress*: *Bowtie* is required to map simulated reads to the transcriptome. In this case, *piquant* creates transcriptome sequences for mapping using a tool from the *RSEM* package (`rsem-prepare-reference`).
- *Sailfish*, *Salmon* and *Kallisto*: *piquant* again uses `rsem-prepare-reference` from the *RSEM* package to create reference transcriptome sequences.

piquant has been tested with *Bowtie* version 1.0.0 and *TopHat* version 2.0.10.

Attention: *TopHat* does not currently execute under Python 3. Hence, if *piquant* is being run in a virtual environment in which the command `python` invokes Python 3, the main *TopHat* script must be altered so as to invoke Python 2. This can be done by altering the first line of the *TopHat* script to read `#!/usr/bin/env python2`.

Finally, the recording of time and memory usage by quantification tools requires that the GNU `time` command is available at `/usr/bin/time`. Resource usage recording can be turned off by specifying the `--nousage` option to the `prepare_quant_dirs` and `analyse_runs` *piquant* commands.

CHAPTER 3

Installation

Attention: As *piquant* has a number of dependencies on other Python packages, it is **strongly** recommended to install in an isolated environment using the virtualenv tool. The virtualenvwrapper tool makes managing multiple virtual environments easier.

Create and work in a virtual environment for *piquant* using the `virtualenvwrapper` tool:

```
mkproject piquant
```

Clone the *piquant* GitHub repository into this environment:

```
git clone https://github.com/lweasel/piquant.git .
```

Install the *piquant* package and scripts, and their Python package dependencies, into the virtual environment by running:

```
pip install .
```

in the tool's top level directory. Note that it may take some time to install and build the dependencies.

Run unit tests for *piquant* using the command:

```
py.test test
```

CHAPTER 4

`piquant` commands

Stages of the *piquant* pipeline are executed via the following commands of the `piquant` executable:

- Simulating reads
  - `prepare_read_dirs`
  - `create_reads`
  - `check_reads`
- Quantifying transcript expression
  - `prepare_quant_dirs`
  - `prequantify`
  - `quantify`
  - `check_quant`
- Producing statistics and graphs
  - `analyse_runs`

Further information on each command is given in the sections below. Note first, however, that the commands share a number of common command line options.

Common options

The following command line options control which combinations of sequencing parameters and quantification tools the particular `piquant` command will be executed for. The value of each option should be a comma-separated list:

- `--read-length`: A comma-separated list of integer read lengths for which to simulate reads or perform quantification.
- `--read-depth`: A comma-separated list of integer read depths for which to simulate reads or perform quantification.

- `--paired-end`: A comma-separated list of "False" or "True" strings indicating whether read simulation or quantification should be performed for single- or paired-end reads or both.
- `--errors`: A comma-separated list of "False" or "True" strings indicating whether read simulation or quantification should be performed without or with sequencing errors introduced into the reads, or both.
- `--bias`: A comma-separated list of "False" or "True" strings indicating whether read simulation or quantification should be performed without or with sequence bias introduced into the reads, or both.
- `--stranded`: A comma-separated list of "False" or "True" strings indicating whether reads should be simulated as coming from an unstranded or strand-specific RNA-seq protocol, or both.
- `--noise-perc`: A comma-separated list of positive integers. Each indicates a percentage of the main sequencing depth; in each case a set "noise transcripts" will be sequenced to this depth. A value of zero indicates that no noise reads will be simulated.
- `--quant-method`: A comma-separated list of quantification methods for which transcript quantification should be performed. By default, *piquant* can quantify via the methods "Cufflinks", "RSEM", "Express", "Sailfish", "Salmon" and "Kallisto". (Note that this option is not relevant for the simulation of reads).

Except in the case of the `--quant-method` option when simulating reads, values for each of these options *must* be specified; otherwise `piquant` will exit with an error. For ease of use, however, the options can also be specified in an options file, via the common command line option `--options-file` (indeed, any command-line option can be specified in this file). Such an options file should take the form of one option and its value per-line, with option and value separated by whitespace, e.g.:

```
--quant-method Cufflinks,RSEM,Express,Sailfish
--read-length 35,50,75,100
--read-depth 10,30
--paired-end False,True
--errors False,True
--bias False
--stranded True
--noise-perc 0,5,10
```

As options can be specified both in an options file, and via individual command line options, in case of conflict the values specified on the command line override those in the options file.

`piquant` commands also share the following additional common command line options:

- `--log-level`: One of the strings "debug", "info", "warning", "error" or "critical" (default "info"), determining the maximum severity level at which log messages will be written to standard error.
- `--options-file`: Specifies the path to a file containing options as described above.

Prepare read directories (`prepare_read_dirs`)

The `prepare_read_dirs` command is used to prepare the directories in which RNA-seq reads will subsequently be simulated - one such directory is created for each possible combination of sequencing parameters determined by the options `--read-length`, `--read-depth`, `--paired-end`, `--errors`, `--bias`, `--stranded` and `--noise-perc`, and each directory is named according to its particular set of sequencing parameters. For example, with the following command line options specified:

- `--read-length`: 50
- `--read-depth`: 30
- `--paired-end`: False,True
- `--errors`: False,True

- `--bias`: False,True
- `--stranded`: False
- `--noise_perc`: 0

eight read simulation directories will be created:

- `30x_50b_se_no_errors_unstranded_no_bias_no_noise`: i.e. 30x sequencing depth, 50 base-pairs read length, unstranded protocol, no background noise, single-end reads, no read errors or sequence bias
- `30x_50b_se_errors_unstranded_no_bias_no_noise`: i.e. 30x sequencing depth, 50 base-pairs read length, unstranded protocol, no background noise, single-end reads, with read errors, no sequence bias
- `30x_50b_se_no_errors_unstranded_bias_no_noise`: i.e. 30x sequencing depth, 50 base-pairs read length, unstranded protocol, no background noise, single-end reads, no read errors, with sequence bias
- `30x_50b_se_errors_unstranded_bias_no_noise`: i.e. 30x sequencing depth, 50 base-pairs read length, unstranded protocol, no background noise, single-end reads, with read errors and sequence bias
- `30x_50b_pe_no_errors_no_unstranded_bias_no_noise`: i.e. 30x sequencing depth, 50 base-pairs read length, unstranded protocol, no background noise, paired-reads, no read errors or sequence bias
- `30x_50b_pe_errors_no_unstranded_bias_no_noise`: i.e. 30x sequencing depth, 50 base-pairs read length, unstranded protocol, no background noise, paired-end reads, with read errors, no sequence bias
- `30x_50b_pe_no_errors_unstranded_bias_no_noise`: i.e. 30x sequencing depth, 50 base-pairs read length, unstranded protocol, no background noise, paired-end reads, no read errors, with sequence bias
- `30x_50b_pe_errors_unstranded_bias_no_noise`: i.e. 30x sequencing depth, 50 base-pairs read length, unstranded protocol, no background noise, paired-end reads, with read errors and sequence bias

Within each read simulation directory, three files are always written:

- `flux_simulator_main_expression.par`: A *Flux Simulator* *[FluxSimulator]* parameters file suitable for creating a transcript expression profile.
- `flux_simulator_main_simulation.par`: A *Flux Simulator* parameters file suitable for simulating RNA-seq reads according to the created transcript expression profile.
- `run_simulation.sh`: A Bash script which, when executed, will use *Flux Simulator* and the above two parameters files to simulate reads for the appropriate combination of sequencing parameters.

In addition, if "background noise" reads are being simulated (i.e. the value of the `--noise-perc` option is greater than zero), the following two additional files are written:

- `flux_simulator_noise_expression.par`: A *Flux Simulator* parameters file suitable for creating a transcript expression profile for the set of transcripts that will be used to simulate background noise.
- `flux_simulator_noise_simulation.par`: A *Flux Simulator* parameters file suitable for simulating RNA-seq reads according to the created noise transcript expression profile.

Note that it is possible to execute the `run_simulation.sh` script directly; however by using the `piquant` command `create_reads`, sets of reads for several combinations of sequencing parameters can be created simultaneously as a batch (see *Create reads* below).

In addition to the command line options common to all `piquant` commands (see *Common options* above), the `prepare_read_dirs` command takes the following additional options:

- `--reads-dir`: The parent directory into which directories in which reads will be simulated will be written. This directory will be created if it does not already exist (default: output).
- `--transcript-gtf`: The path to a GTF formatted file describing the main set of transcripts to be simulated by *Flux Simulator*. This GTF file location must be supplied. Note that the GTF file should only contain features

of feature type "exon", and that every exon feature should specify both "gene_id" and "transcript_id" among its attributes.

- `--noise-transcript-gtf`: The path to a GTF formatted file describing a set of transcripts that will be used to simulated background noise. This GTF file location needs only be specified if background noise is being simulated (ie. for values of `--noise-perc` other than zero); however, in these cases it must be specified. The same requirements as to GTF file format apply as above for the option `--transcript-gtf`.
- `--genome-fasta`: The path to a directory containing per-chromosome genome sequences in FASTA-formatted files. This directory location must be supplied.
- `--num-molecules`: *Flux Simulator* parameters will be set so that the initial pool of main transcripts contains this many molecules. Note that although it depends on this value, the number of fragments in the final library from which reads will be sequenced is also a complicated function of the parameters at each stage of *Flux Simulator*'s sequencing process. This parameter should be set high enough that the number of fragments in the final library exceeds the number of reads necessary to give any of the sequencing depths required. If the initial number of molecules is not great enough to create the required number of reads, the `run_simulation.sh` script will exit with an error (default: 30,000,000).
- `--num-noise-molecules`: *Flux Simulator* parameters will be set so that the initial pool of noise transcripts contains this many molecules; this parameter should be set high enough that the number of fragments in the final noise simulation library exceeds the number of reads necessary to give any required sequencing depth (default: 2,000,000).
- `--nocleanup`: When run, *Flux Simulator* creates a number of large intermediate files. Unless `--nocleanup` is specified, the `run_simulation.sh` Bash script will be constructed so as to delete these intermediate files once read simulation has finished.

Create reads (`create_reads`)

The `create_reads` command is used to simulate RNA-seq reads via the `run_simulation.sh` scripts that have been written by the `prepare_read_dirs` command (see *Prepare read directories* above). For each possible combination of sequencing parameters determined by the options `--read-length`, `--read-depth`, `--paired-end`, `--errors`, `--bias`, `--stranded` and `--noise-perc`, the appropriate `run_simulation.sh` script is launched as a background process, ignoring hangup signals (via the `nohup` command). After launching the scripts, `piquant` exits.

In addition to the command line options common to all `piquant` commands (see *Common options* above), the `create_reads` command takes the following additional options:

- `--reads-dir`: The parent directory in which directories in which reads will be simulated have been written (default: output).

For details on the process of read simulation executed via `run_simulation.sh`, see *Simulating reads*.

Check reads were successfully created (`check_reads`)

The `check_reads` command is used to confirm that simulation of RNA-seq reads via `run_simulation.sh` scripts successfully completed. For each possible combination of sequencing parameters determined by the options `--read-length`, `--read-depth`, `--paired-end`, `--error`, `--bias`, `--stranded` and `--noise-perc`, the relevant read simulation directory is checked for the existence of the appropriate FASTA or FASTQ files containing simulated reads. A message is printed to standard error for those combinations of sequencing parameters for which read simulation has not yet finished, or for which simulation terminated unsuccessfully.

In the case of unsuccessful termination, the file `nohup.out` in the relevant simulation directory contains the messages output by both *Flux Simulator* and the *piquant* scripts that were executed, and this file can be examined for the source of error.

In addition to the command line options common to all `piquant` commands (see *Common options* above), the `check_reads` command takes the following additional options:

- `--reads-dir`: The parent directory in which directories in which reads were simulated are located (default: output).

Prepare quantification directories (`prepare_quant_dirs`)

The `prepare_quant_dirs` command is used to prepare the directories in which transcript quantification will take place - one such directory is created for each possible combination of sequencing and quantification parameters determined by the options `--read-length`, `--read-depth`, `--paired-end`, `--error`, `--bias`, `--stranded`, `--noise-perc` and `--quant-method`, and each directory is named according to its particular set of parameters. For example with the following command line options specified:

- `--quant-method`: Cufflinks, RSEM, Express, Sailfish
- `--read-length`: 50
- `--read-depth`: 30
- `--paired-end`: False,True
- `--error`: True
- `--bias`: True
- `--stranded`: False
- `--noise-perc` 10

eight quantification directories will be created:

- `Cufflinks_30x_50b_se_errors_stranded_bias_noise-10x`: i.e. 30x read depth, 50 base-pairs read length, unstranded protocol, noise transcripts at 10% of the main read depth (i.e. at 0.1 * 30 = 3x sequencing depth), single-end reads with both errors and bias, with transcripts quantified by Cufflinks.
- `Cufflinks_30x_50b_pe_errors_stranded_bias_noise-10x`: i.e. 30x read depth, 50 base-pairs read length, unstranded protocol, noise transcripts at 10% of the main read depth, paired-end reads with both errors and bias, with transcripts quantified by Cufflinks.
- `RSEM_30x_50b_se_errors_stranded_bias_noise-10x`: i.e. 30x read depth, 50 base-pairs read length, unstranded protocol, noise transcripts at 10% of the main read depth, single-end reads with both errors and bias, with transcripts quantified by RSEM.
- `RSEM_30x_50b_pe_errors_stranded_bias_noise-10x`: i.e. 30x read depth, 50 base-pairs read length, unstranded protocol, noise transcripts at 10% of the main read depth, paired-end reads with both errors and bias, with transcripts quantified by RSEM.
- `Express_30x_50b_se_errors_stranded_bias_noise-10x`: i.e. 30x read depth, 50 base-pairs read length, unstranded protocol, noise transcripts at 10% of the main read depth, single-end reads with both errors and bias, with transcripts quantified by eXpress.
- `Express_30x_50b_pe_errors_stranded_bias_noise-10x`: i.e. 30x read depth, 50 base-pairs read length, unstranded protocol, noise transcripts at 10% of the main read depth, paired-end reads with both errors and bias, with transcripts quantified by eXpress.

- `Sailfish_30x_50b_se_errors_stranded_bias_noise-10x`: i.e. 30x read depth, 50 base-pairs read length, unstranded protocol, noise transcripts at 10% of the main read depth, single-end reads with both errors and bias, with transcripts quantified by Sailfish.
- `Sailfish_30x_50b_pe_errors_stranded_bias_noise-10x`: i.e. 30x read depth, 50 base-pairs read length, unstranded protocol, noise transcripts at 10% of the main read depth, paired-end reads with both errors and bias, with transcripts quantified by Sailfish.

Within each quantification directory, a single file is written:

- `run_quantification.sh`: A Bash script which, when executed, will use the appropriate tool and simulated RNA-seq reads to quantify transcript expression.

As is the case when simulating reads, it is possible to execute the `run_quantification.sh` script directly; however, by using the `piquant` command `quantify`, quantification for several combinations for sequencing parameters and quantification tools can be executed simultaneously as a batch (see *Perform quantification* below).

In addition to the command line options common to all `piquant` commands (see *Common options* above), the `prepare_quant_dirs` command takes the following additional options:

- `--reads-dir`: The parent directory in which directories in which reads were simulated are located (default: output).
- `--quant-dir`: The parent directory into which directories in which quantification will be performed will be written. This directory will be created if it does not already exist (default: output).
- `--transcript-gtf`: The path to a GTF formatted file describing the transcripts from which reads were simulated by *Flux Simulator*. This GTF file location must be supplied. The transcripts GTF file should be the same as was supplied to the `prepare_read_dirs` command (see *Prepare read directories* above).
- `--genome-fasta`: The path to a directory containing per-chromosome genome sequences in FASTA-formatted files. This directory location must be supplied. The genome sequences should be the same as were supplied to the `prepare_read_dirs` command.
- `--num-threads`: Multi-threaded quantification methods will use this number of threads (default: 1).
- `--nocleanup`: When run, quantification tools may create a number of output files. Unless `--nocleanup` is specified, the `run_quantification.sh` Bash script will be constructed so as to delete all of these, except those essential for *piquant* to calculate the accuracy with which quantification has been performed.
- `--nousage`: By default, *piquant* will collect time and memory resource usage statistics for the execution of quantification tools. This is done via the GNU `time` command, which is assumed to reside at `/usr/bin/time`. If the GNU `time` command is not available at this location, or resource usage statistics are not desired, specifying this option will disable their collection.
- `--plot-format`: The file format in which graphs produced during the analysis of this quantification run will be written to - one of "pdf", "svg" or "png" (default "pdf").
- `--grouped-threshold`: When producing graphs of statistics plotted against groups of transcripts determined by a transcript classifier (see *Transcript classifiers*), only groups with greater than this number of transcripts will contribute to the plot.
- `--error-fraction-threshold`: When producing graphs, transcripts whose estimated TPM (transcripts per million) is greater than this percentage higher or lower than their real TPM are considered above threshold for the "error fraction" statistic (default: 10).
- `--not-present-cutoff`: Prior to any statistics being calculated, all real and estimated TPM values below this cut-off value are truncated to zero, to avoid biasing analyses with differences between very low real and estimated TPM values that are likely of little biological interest (default 0.1).

Prepare for quantification (`prequantify`)

Some quantification tools may require some action to be taken prior to quantifying transcript expression which, however, only needs to be executed once for a particular set of transcripts and genome sequences - for example, preparing a *Bowtie [Bowtie]* index for the genome, or creating transcript FASTA sequences. The `piquant` command `prequantify` will execute these pre-quantification actions for any quantification tools specified by the command line option `--quant-method`.

Note that prequantification can, if necessary, be run manually for any particular quantification tool by executing the appropriate `run_simulation.sh` script with the `-p` command line option.

Perform quantification (`quantify`)

The `quantify` command is used to quantify transcript expression via the `run_quantification.sh` scripts that have been written by the `prepare_quant_dirs` command (see *Prepare quantification directories* above). For each possible combination of parameters determined by the options `--read-length`, `--read-depth`, `--paired-end`, `--error`, `--bias`, `--stranded`, `noise-perc`, and `--quant-method`, the appropriate `run_quantification.sh` script is launched as a background process, ignoring hangup signals (via the `nohup` command). After launching the scripts, `piquant` exits.

For details on the process of quantification executed via `run_quantification.sh`, see *Quantifying expression*.

Check quantification was successfully completed (`check_quant`)

The `check_quant` command is used to confirm that quantification of transcript expression via `run_quantification.sh` scripts successfully completed. For each possible combination of parameters determined by the options `--read-length`, `--read-depth`, `--paired-end`, `--error`, `--bias`, `--stranded`, `--noise-perc` and `--quant-method`, the relevant quantification directory is checked for the existence of the appropriate output files of the quantification tool that will subsequently be used for assessing quantification accuracy. A message is printed to standard error for those combinations of parameters for which quantification has not yet finished, or for which quantification terminated unsuccessfully.

In the case of unsuccessful termination, the file `nohup.out` in the relevant quantification directory contains the messages output by both the quantification tool and the *piquant* scripts that were executed, and this file can be examined for the source of error.

Analyse quantification results (`analyse_runs`)

The `analyse_runs` command is used to gather data and calculate statistics, and to draw graphs, pertaining to the accuracy of quantification of transcript expression. Statistics are calculated, and graphs drawn, for those combinations of quantification tools and sequencing parameters determined by the options `--read-length`, `--read-depth`, `--paired-end`, `--error`, `--bias`, `--stranded`, `--noise-perc` and `--quant-method`. In addition, by default, graphs are produced comparing the time and memory usage of the different quantification tools during the prequantification and quantification steps.

For more details on the statistics calculated and the graphs drawn, see *Assessing quantification performance*.

In addition to the command line options common to all `piquant` commands (see *Common options* above), the `analyse_runs` command takes the following additional options:

- `--quant-dir`: The parent directory into which directories in which quantification was performed were written.
- `--stats-dir`: The path to a directory into which statistics and graph files will be written. The directory will be created if it does not already exist.
- `--plot-format`: The file format in which graphs produced during analysis will be written to - one of "pdf", "svg" or "png" (default "pdf").
- `--grouped-threshold`: When producing graphs of statistics plotted against groups of transcripts determined by a transcript classifier, only groups with greater than this number of transcripts will contribute to the plot.
- `--nousage`: Specify this option if graphs of resource usage are not desired to be produced. Note that if this option was specified when preparing quantification directories, it should also be specified here.

CHAPTER 5

Simulating reads

For each particular combination of sequencing parameters - sequencing depth, read length, single- or paired-end reads, lack or presence of errors and bias, strandedness and noise depth - reads are simulated by running the `run_simulation.sh` script in the relevant directory that has been created by the `piquant` command `prepare_read_dirs`.

Running `run_simulation.sh` results in the following main steps being executed:

Create expression profiles

Flux Simulator *[FluxSimulator]* is used to create an expression profile (a `.pro` file) for the supplied set of main transcripts. This profile defines the set of expressed transcripts, and the relative abundances of those transcripts, from which reads will subsequently be simulated. If the noise depth is greater than zero, then an expression profile for the supplied set of noise transcripts is also created.

For more information on the model and algorithm used by *Flux Simulator* to create expression profiles, see the *Flux Simulator* website.

Calculate required number of reads

Given a particular read length and (approximate) desired sequencing depths, a certain number of reads will need to be simulated for both the main and noise transcript sets. These numbers are calculated by the support script `calculate_reads_for_depth` (see *Calculate reads required for sequencing depth* for more details) and the *Flux Simulator* simulation parameters files, `flux_simulator_main_simulation.par` and `flux_simulator_noise_expression.par`, are updated accordingly.

Simulate reads

Next, *Flux Simulator* is used to simulate the required number of reads for the desired sequencing depths, according to the previously created transcript expression profiles. Note that depending on the number of reads being simulated, this step can take considerable time.

Note that:

- Reads are not simulated from the poly-A tails of transcripts (this behaviour is controlled by the *Flux Simulator* parameters `POLYA_SHAPE` and `POLYA_SCALE`), as the multi-mapping of such reads was found to cause problems for certain quantification tools (for more details on *Flux Simulator*'s transcript modifications, see here).
- If sequencing errors have been specified, such errors are simulated with *Flux Simulator*'s 76bp error model; the simulator scales this error model appropriately for the length of reads being produced (for more details on *Flux Simulator*'s error models, see here).
- PCR amplification of fragments, controlled by the *Flux Simulator* parameter `PCR_DISTRIBUTION`, is disabled (for more details on *Flux Simulator*'s simulation of PCR, see here).
- The *Flux Simulator* parameter `UNIQUE_IDS` is set to ensure that, in the case of paired-end reads, read names match for the reads of each pair, excluding the '/1' and '/2' suffix identifiers - this behaviour is required for some quantification tools. Note that with this option set, the reads are effectively stranded, since the first read of each pair ('/1') always originates from the sense strand, and the second ('/2') from the anti-sense strand. For more details on the `UNIQUE_IDS` parameter, see here. (*n.b.* in the case of single-end reads, the reads produced are unstranded).

Check reads

The FASTA or FASTQ files produced by read simulation are checked to ensure that the required number of main and noise reads have been created. If, in either case, the required number of reads are not present, the `run_simulation.sh` exits with an error.

Join and shuffle reads

If both main and noise reads have been simulated (i.e. if the noise depth is greater than zero), then the two FASTA or FASTQ files produced are concatenated.

Note that some transcript quantification tools require reads to be presented in a random sequence. However the reads output by *Flux Simulator* have an inherent order, and hence reads are also randomly shuffled at this stage.

Fix strandedness

For single-end reads, the reads produced by `FluxSimulator` come from either the sense or antisense strand. Hence, if a stranded protocol is being simulated, the support script `fix_antisense_reads` (see *Fix antisense reads* for more details) is used to reverse complement any reads derived from the antisense strand.

For paired-end reads, reads are already effectively stranded, originating from the forward transcript strand. Hence, if an unstranded protocol is being simulated, the support script `randomise_read_strands` (see randomise_read_strands for more details) is used to randomly reassign pairs of paired-end reads such that the first read now corresponds to the antisense strand.

Apply sequence bias

In a real RNA-seq experiment, there are many sources of potential bias, some only poorly understood, that may lead to non-uniform coverage of expressed transcripts by sequenced reads; for example the biases in nucleotide composition at the beginning of reads sequenced in certain Illumina protocols, as described by Hansen *et al.* *[Hansen]*.

If sequencing bias has been specified, then the support script `simulate_read_bias` (see *Simulate sequence bias in reads* for more details) is executed to approximate one form of such bias. A position weight matrix is used to preferentially select reads with a nucleotide composition at their beginning similar to that observed by Hansen *et al.*

Finalise output files

Finally, the reads output by *Flux Simulator* are put into a form suitable for downstream transcript quantification. The result of running `run_simulation.sh` is one or two FASTA or FASTQ files containing the simulated reads:

- For single-end reads, with no read errors specified, one FASTA file is output (`reads_final.fasta`).
- For single-end reads, with read errors, one FASTQ file is output (`reads_final.fastq`).
- For paired-end reads, with no read errors specified, two FASTA files are output (`reads_final.1.fasta` and `reads_final.2.fasta`).
- For paired-end reads, with read errors, two FASTQ files are output (`reads_final.1.fastq` and `reads_final.2.fastq`).

CHAPTER 6

Quantifying expression

For each particular combination of sequencing parameters - sequencing depth, read length, single- or paired-end reads, lack or presence of errors and bias, strandedness and noise sequencing depth - and each quantification tool, transcript quantification is performed by running the `run_quantification.sh` script in the relevant directory that has been created by the `piquant` command `prepare_quant_dirs`.

The `run_quantification` script takes a number of command line flags which control its operation:

- `-p`: If specified, any preparatory action is taken that is necessary for the particular quantification tool before transcript abundances can be calculated.
- `-q`: If specified, transcript abundances are calculated for the relevant set of simulated reads.
- `-a`: If specified, data necessary for the assessment of the accuracy of transcript expression estimation is assembled, and measures of accuracy calculated.

These three modes of operation are discussed below. Note that when performing batch quantification, the `piquant` command `prequantify` executes `run_quantification.sh` scripts with the `-p` flag, while the command `quantify` executes scripts with the `-q` and `-a` flags.

Preparing for quantification

Running `run_quantification.sh` with the `-p` flag results in the following steps being executed. Note that for any particular quantification tool, running a `run_quantification.sh` script for this tool with the `-p` flag a second (or subsequent) time will be a no-op.

Tool-specific preparation

Any actions particular to the quantification tool to be used that must be taken prior to quantifying transcripts, but which only need to be executed once for a particular set of input transcripts and genome sequences, are performed here (for example, preparing a *Bowtie [Bowtie]* index for the genome). Any data created by these actions will be written to a directory `quantifier_scratch` that is created alongside the read simulation and transcript quantification directories.

For more details on the prequantification actions performed for each particular quantification tool, see *Quantification tools*.

Calculate number of transcripts per gene

Next, the support script `count_transcripts_for_genes` (see *Count transcripts for genes*) is used to calculate the number of transcripts shared by each gene in the set determined by the main transcript GTF file specified when the `run_quantification.sh` script was created. This data is stored in the file `transcript_counts.csv` in the directory `quantifier_scratch`, as described above.

Note that this action will only be performed once, regardless of how many `run_quantification.sh` scripts are run. The per-gene transcript counts thus calculated will be used when assessing the accuracy of transcript abundance estimation (see *Assessing quantification performance*).

Calculate unique sequence per transcript

Finally, the support script `calculate_unique_transcript_sequence` (see *Calculate unique transcript sequence*) is used to calculate the length of sequence in base pairs that is unique to each transcript enumerated in the transcript GTF file specified when the `run_quantification.sh` script was created. This data is stored in the file `unique_sequence.csv` in the directory `quantifier_scratch`, as described above (see *Assessing quantification performance*).

Again, this action will only be performed once for any particular set of input transcripts. The unique sequence lengths thus calculated will be used when assessing abundance estimation accuracy.

Performing quantification

Running `run_quantification.sh` with the `-q` flag causes the relevant quantification tool to be run on the appropriate set of simulated RNA-seq reads, to estimate transcript abundance (depending on the particular quantification tool, this can be time, memory and/or CPU intensive). Note that in contrast to the case of pre-quantification tasks, re-running `run-quantification.sh` with this flag will cause transcript abundance estimates to be recalculated.

For more details on the particular commmands executed for each quantification tool, see *Quantification tools*.

Assessing quantification accuracy

Running `run_quantification.sh` with the `-a` flag results in the following two steps being executed. As above, for performing quantification itself, re-running `run_quantification.sh` with this flag will repeat the assessment of quantification accuracy.

Assemble data

The support script `assemble_quantification_data` (see *Assemble data for a single quantification run*) assembles the data required to assess the accuracy of transcript abundance estimation from the following sources:

- The *FluxSimulator [FluxSimulator]* main transcript expression profile file created during read simulation, containing the 'ground truth' relative transcript abundances.
- A quantification tool-specific output file containing estimated transcript abundances.

- The file `transcript_counts.csv` containing per-gene transcript counts, created by the step *Calculate number of transcripts per gene* above.
- The file `unique_sequence.csv` containing lengths of sequence unique to each transcript, created by the step *Calculate unique sequence per transcript* above.

Assembled data is written to a CSV file `tpms.csv` in the quantification directory. This contains, for each transcript in the input set:

- the transcript identifier
- the transcript sequence length in bases
- the number of bases that are unique to the transcript
- the number of isoforms of the transcript's gene of origin
- the "real" transcript abundance used by *FluxSimulator* to simulate reads (measured in transcripts per million or TPMs)
- the transcript abundance estimated by the quantification tool (measured in transcripts per million)

Perform accuracy analysis

Finally, the support script `analyse_quantification_run` reads the CSV file `tpms.csv` produced by the assembly step above, and calculates statistics and plots graphs that can be used to assess the accuracy of transcript abundance estimation by the particular quantification tool. The statistics calculated, transcript classification measures used, and graphs drawn are described in full in *Assessing quantification performance*.

CHAPTER 7

Quantification tools

By default, the *piquant* pipeline has the ability to run the following six transcript quantification tools. The pipeline can, however, be easily extended to run additional quantification tools by editing the `quantifiers.py` Python module, as described in *Adding a new quantifier*.

Attention: It is important to clarify that rather than testing the performance of quantification tools alone, *piquant* is actually testing the performance, as regards the accuracy of transcript quantification, of mapping plus quantification tool pipelines (at least in the case of quantification tools which require mapping of reads prior to quantification). It can easily be understood, for example, how difficulties encountered when mapping reads to the genome might adversely affect quantification performance, through factors beyond a quantification tool's control.

Cufflinks

Note: *piquant* has been tested with *Cufflinks [Cufflinks]* version 2.2.1 and *TopHat [TopHat]* version 2.0.10.

In preparation for quantifying transcripts with *Cufflinks*, the following prequantification tasks are executed (these steps are a necessary preliminary for mapping simulated reads to the genome with TopHat):

- A *Bowtie [Bowtie]* index is built for the genome using the `bowtie-build` command.
- A FASTA file for the genome, corresponding to the *Bowtie* index, is constructed using the `bowtie-inspect` command.

When quantifying transcripts with *Cufflinks* for a set of simulated RNA-seq reads, reads are first mapped to the genome using the splice-aware mapper *TopHat*, with the following command line options (see the TopHat manual for further details on these options):

- `--library-type <type>`: The library type is set to `fr-secondstrand` if reads from a stranded protocol are being quantified, and to `fr-unstranded` for unstranded reads.
- `--no-coverage-search`: Coverage-based search for junctions is disabled.

Cufflinks is then run to estimate transcript abundances with the following command line options (see the Cufflinks manual for further details on these options):

- `--library-type <type>`: The library type is set to `fr-unstranded` or `fr-secondstrand` as for *TopHat* above.
- `-u`: Reads mapping to multiple locations in the genome are more accurately weighted.
- `-b <genome FASTA file>`: *Cufflinks*' bias detection and correction algorithm is run.

After transcript abundance estimation has completed, of the files output by *Cufflinks*, only `isoforms.fpkm_tracking` is retained (unless the `--nocleanup` option was specified when the `run_quantification.sh` script was created). Relative transcript abundances are extracted from this file in units of FPKM (fragments per kilobase of exon per million reads mapped) and then converted to relative abundances measured in TPM (transcripts per million).

(Note: for an excellent discussion of RNA-seq expression units, see this blog post).

RSEM

Note: *piquant* has been tested with *RSEM [RSEM]* version 1.2.19 and *Bowtie* version 1.0.0.

In preparation for quantifying transcripts with *RSEM*, the `rsem-prepare-reference` tool from the *RSEM* package is used to construct sequences in FASTA format for the input set of transcripts (see here for more details on the `rsem-prepare-reference` tool).

Then, when quantifying transcripts with *RSEM* for a set of simulated RNA-seq reads, the tool `rsem-calculate-expression` is executed with the `--strand-specific` command line option in the case that reads have been simulated for a stranded protocol. See here for more details on the `rsem-calculate-expression` tool.

After transcript abundance estimation has completed, of the files output by *RSEM*, only `<sample_name>.isoforms.results` is retained (unless the `--nocleanup` option was specified when the `run_quantification.sh` script was created). Relative transcript abundances are extracted from this file in units of TPM (transcripts per million).

eXpress

Note: *piquant* has been tested with *eXpress [eXpress]* version 1.5.1 and *Bowtie [Bowtie]* version 1.0.0.

In preparation for quantifying transcripts with *eXpress*, the `rsem-prepare-reference` tool from the *RSEM* package is used to construct transcript sequences, as described above.

When quantifying transcripts with *eXpress* for a set of simulated RNA-seq reads, reads are first mapped to the transcript sequences using *Bowtie*, with the following command line options, which have, in general, been chosen to provide similar alignment behaviour as is implemented within the *RSEM* pipeline (see the Bowtie manual for further details on these options):

- `-e 99999999`: The maximum permitted total of quality values at all mismatched read positions throughout the entire alignment.
- `-l 25`: A seed length for alignments of 25 base pairs.

- `-I 1`: A minimum insert size of 1 base pair for valid paired-end alignments.
- `-X 1000`: A maximum insert size of 1000 base pairs for valid paired-end alignments.
- `-a`: All valid alignments are reported per read or read pair.
- `-m 200`: All alignments are suppressed for a particular read or read pair if more than 200 alignments exist for it.
- `-S`: Alignments are printed in SAM *[SAM]* format.
- `--norc`: Only specified if stranded reads are being quantified, this option causes only paired-end read configurations corresponding to fragments from the forward strand to be considered.

The alignments produced by *Bowtie* are piped to the `view` command of the *SAMtools* package to convert them to BAM format, for subsequent input to *eXpress*. *eXpress* is executed with the `--f-stranded` (for single-end reads) or `--fr-stranded` (for paired-end reads) command line options in the case that reads have been simulated for a stranded protocol. See the eXpress manual for further details on the options available.

After transcript abundance estimation has completed, of the files output by *eXpress*, only `results.xprs` is retained (unless the `--nocleanup` option was specified when the `run_quantification.sh` script was created). Relative transcript abundances are extracted from this file in units of TPM (transcripts per million).

Sailfish

Note: *piquant* has been tested with *Sailfish* *[Sailfish]* version 0.8.0.

In preparation for quantifying transcripts with *Sailfish*, the *Sailfish* `index` command is executed to create a kmer index for the input transcript set (for more information on *Sailfish* commands, see the *Sailfish* manual).

Then, when quantifying transcripts with *Sailfish* for a set of simulated RNA-seq reads, the *Sailfish* `quant` command is executed with the following settings for the library type (`-l`) option, depending on whether single- or paired-end, and stranded or unstranded reads are being quantified:

- `U` for single-end reads of unknown strandedness
- `SF` for single-end stranded reads
- `IU` for paired-end reads of unknown strandedness
- `ISF` for paired-end stranded reads.

After transcript abundance estimation has completed, of the files output by *Sailfish*, only `quant.sf` is retained (unless the `--nocleanup` option was specified when the `run_quantification.sh` script was created). Relative transcript abundances are extracted from this file in units of TPM (transcripts per million).

Salmon

Note: *piquant* has been tested with *Salmon* *[Salmon]* version 0.5.1.

In preparation for quantifying transcripts with *Salmon*, the *Salmon* `index` command is executed to create a Salmon index for the input transcript set. The `--type` argument is set to `quasi` to use Salmon's quasi-mapping method of lightweight alignment (for more information on *Salmon* commands, see the *Salmon* manual).

Then, when quantifying transcripts with *Salmon* for a set of simulated RNA-seq reads, the *Salmon* `quant` command is executed with the same settings for the library type (`-l`) option as shown for Sailfish above.

After transcript abundance estimation has completed, of the files output by *Salmon* only `quant.sf` is retained (unless the `--nocleanup` option was specified when the `run_quantification.sh` script was created). Relative transcript abundances are extracted from this file in units of TPM (transcripts per million).

Kallisto

Note: *piquant* has been tested with *Kallisto [Kallisto]* version 0.42.4.

In preparation for quantifying transcripts with *Kallisto*, the *Kallisto* `index` command is executed to create a Kallisto index for the input transcript set (for more information on *Kallisto* commands, see the *Kallisto* manual).

Then, when quantifying transcripts with *Kallisto* for a set of simulated RNA-seq reads, the *Kallisto* `quant` command is executed, with the `--single` option specified and a value of 200 for the `--fragment-length` option (estimated average fragment length) when single-end reads are being quantified. The `--bias` option is specified in all cases, indicating that *Kallisto* should performed sequence-based bias correction.

After transcriptome abundance estimation has completed, of the files output by *Kallisto* only `abundance.tsv` is retained (unless the `--nocleanup` option was specified when the `run_quantification.sh` script was created). Relative transcript abundances are extracted from this file in units of TPM (transcripts per million).

CHAPTER 8

Assessing quantification performance

After reads have been simulated for a set of input transcripts, and quantification tools have been executed to estimate transcript abundance, the final stage of the *piquant* pipeline is to calculate statistics and draw graphs to aid the assessment of transcript quantification performance and resource usage. Note that performance is assessed both at the level of individual quantification runs (i.e. a particular transcript quantification tool executed once for reads simulated according to a certain set of sequencing parameters), and also across multiple quantification runs for comparison of performance. The data and plots generated in each case are detailed below (see *Assessment of a single quantification run* and *Assessment of multiple quantification runs*); however, we first describe the statistics calculated, and the classifiers used to split transcripts into groups sharing similar properties.

Statistics

For each execution of a particular transcript quantification tool for reads simulated according to a certain set of sequencing parameters, a number of statistics are calculated from the real and estimated transcript abundances. Those calculated by default are listed below; however it is easy to extend *piquant* to calculate additional statistics (see *Adding a new statistic*).

Note that each statistic is calculated both for the set of estimated transcript abundances as a whole, and for each group of transcripts determined to share similar properties by each transcript classifier (see *Transcript classifiers*).

Note also that each statistic can be marked as being suitable for producing interesting graphs or not; all statistics described below are suitable for graphing unless stated otherwise.

Number of 'expressed' transcripts

This is simply the number of transcripts in the RNA-seq data set with non-zero "ground truth" TPM. Note, however, that all real and estimated TPM values below the threshold defined by the command line option `--not-present-cutoff` are truncated to zero, before this or any other statistics are calculated. Estimating the abundance of very rare transcripts is difficult, and truncation is performed to avoid biasing analyses with differences between very low real and estimated TPM values that are likely of little biological interest.

This statistic is marked as being not suitable for producing graphs.

Spearman correlation

The Spearman rank correlation coefficient between real and estimated TPMs for transcripts considered to be expressed after truncation of TPM values below the low abundance threshold. When assessing quantification performance, a higher correlation coefficient is considered to be better.

Error fraction

The fraction of expressed transcripts for which the estimated TPM is greater than a certain threshold percentage higher or lower than the real TPM; when assessing quantification performance, a lower error fraction is considered to be better. The threshold percentage is defined by the command line option `--error-fraction-threshold`, which must be specified when executing the *piquant* command `prepare_quant_dirs`. The default value is set at 10%.

Median percent error

For expressed transcripts, the median value of the percentage errors of estimated compared to real TPMs; when assessing quantification performance, a median percent error closer to zero is considered to be better. This statistic can also indicate whether a particular quantification tool tends to over- or under-estimate transcript abundances, for transcripts as a whole, or for certain classes of transcript.

Sensitivity

The sensitivity (or true positive rate) of a transcript quantification method is calculated to be the fraction of expressed transcripts (that is, both "true positives" and "false negatives") which were correctly identified by the quantification tool as being present (just the "true positives"):

$$sensitivity = \frac{TP}{TP + FN}$$

Specificity

The specificity (or true negative rate) of a transcript quantification method is calculated to be the fraction of transcripts considered to be not expressed (that is, their real TPM value lies below the truncation threshold - both "true negatives" and "false positives"), which were correctly identified as not being present (just the "true negatives"):

$$specificity = \frac{TN}{TN + FP}$$

Transcript classifiers

Transcript classifiers split the whole set of input transcripts into discrete groups, these groups sharing some similar property; such a division of transcripts then allows the performance of quantification tools to be assessed across different types of transcripts. The transcript classifiers provided by default are listed below; however it is easy to extend *piquant* to add additional classifiers (see *Adding a new transcript classifier*).

Note, however, that transcript classifiers fall into one of two distinct types, and these types are described first.

"Grouped" classifiers

The first type of transcript classifiers generally split the set of input transcripts into fixed groups dependent on some property inherent in the transcripts (or their simulated abundances) themselves. For example, one could consider "short", "medium" or "long" transcripts, or those expressed at "low", "medium" or "high" simulated abundance.

The following "grouped" classifiers are provided:

- *Number of transcripts of originating gene*
- *Real transcript abundance*
- *Transcript length*
- *Transcript unique sequence percentage*
- *Transcript unique sequence length*

"Distribution" classifiers

The second type of transcript classifiers split the set of input transcripts into two groups, those above and below some threshold, where that threshold is generally the value of some property of quantification. For example, one could consider transcripts whose estimated abundance is more or less than a certain percentage different from the real abundance. By varying the threshold value, these classifiers can be used to produce graphs of the distribution of the property in question.

The following "distribution" classifier is provided:

- *Absolute percent error*

Number of transcripts of originating gene

This classifier simply groups transcripts according to the number of isoforms of their originating gene.

Real transcript abundance

This classifier groups transcripts by a measure of their real abundance. Five categories of prevalence are defined according to the log (base 10) of their real abundance in transcripts per million:

- Log real TPM <= 0 (<=1 transcript per million)
- Log real TPM <= 0.5 (>1 and <=3.16 transcripts per million)
- Log real TPM <= 1: (>3.16 and <=10 transcripts per million)
- Log real TPM <= 1.5: (>10 and <=31.6 transcripts per million)
- Log real TPM > 1.5: (>31.6 transcripts per million)

Transcript length

This classifier groups transcripts by their length in bases. Four categories are defined:

- *very short*: length <= 500 bases
- *short*: > 500 and <= 1000 bases
- *medium*: > 1000 and <= 3000 bases

- *long*: > 3000 bases

Transcript unique sequence percentage

This classifier groups transcripts by the percentage of their sequence which they do not share with any other transcript within their gene of origin. Five categories of transcripts are defined:

- > 0 and <=20% unique sequence
- > 20 and <=40% unique sequence
- > 40 and <=60% unique sequence
- > 60 and <=80% unique sequence
- > 80 and <=100% unique sequence

Transcript unique sequence length

This classifier groups transcripts by the absolute length of sequence which they do not share with any other transcript within their gene of origin. Five categories are defined according to the length of unique sequence:

- 0 unique bases
- > 0 and <= 100 unique bases
- > 100 and <= 300 unique bases
- > 300 and <= 1000 unique bases
- > 1000 unique bases

Absolute percent error

This "distribution" classifier splits transcripts into two groups according to whether the absolute percentage difference between each transcripts estimated and real abundances is greater or less than a given amount.

Resource usage statistics

For each execution of a particular transcript quantification tool for reads simulated according to a certain set of sequencing parameters (and also for the single execution of the prequantification steps for each quantification tool), the following resource usage statistics are recorded:

- *Real time*: The total elapsed real time of all quantification (or prequantification) commands in seconds, log base 10 (via the `%e` format option of the GNU `time` command)
- *User time*: The total number of CPU-seconds (log base 10) that all quantification (or prequantification) commands spent in user mode (via the `%U` format option of GNU `time`).
- *System time*: The total number of CPU-seconds (log base 10) that all quantification (or prequantification) commands spent in kernel mode (via the `%S` format option of GNU `time`).
- *Maximum memory*: The maximum resident memory size of any quantification (or prequantification) command during its execution, in gigabytes (via the `%M` format option of GNU `time`).

Assessment of a single quantification run

Statistics and plots for a single execution of a quantification tool are produced by the support script `analyse_quantification_run` (see *Perform accuracy analysis*) that is run by invoking `run_quantification` with the `-a` command line option (see *Quantifying expression*). The following CSV files and plots (written as PDF files by default) are produced:

CSV files

- `<run-id>_transcript_stats.csv`: A CSV file containing a single row, with a field for each defined statistic (see *Statistics* above) which has been calculated over the whole set of input transcripts. CSV fields are also present describing the quantification tool and sequencing parameters used (i.e. read length, sequencing depth etc.).
- `<run-id>_gene_stats.csv`: A corresponding CSV file, also containing a single row, with a field for each defined statistic which has been calculated over the whole set of input *genes*. Both real and estimated gene "TPMs" are calculated by summing the respective TPM values for that gene's transcripts. As above, CSV fields are also present describing the quantification tool and sequencing parameters used.
- `<run-id>_transcript_stats_by_<classifier>.csv`: A CSV file is created for each "grouped" transcript classifier (see *"Grouped" classifiers*). Each CSV file contains the same fields as `<run-id>_transcript_stats.csv`; however, statistics are now calculated for distinct subsets of transcripts as determined by the transcript classifier, and the CSV file contains one row for each such group. For example, the CSV file `<run-id>_by_gene_trancript_number.csv` contains statistics calculated over those transcripts whose originating gene has only one isoform, those for which the gene has two isoforms, and so on.
- `<run-id>_transcript_distribution_stats_<asc|desc>_by_<classifier>.csv`: Two CSV files ("ascending" and "descending") are created for each "distribution" transcript classifier (see *"Distribution" classifiers*). For a range of values of the classifier's threshold variable (such range being appropriate to the classifier), the "ascending" file contains a row for each threshold value, indicating the fraction of expressed transcripts lying below the threshold. Similarly, for the same range of values, the "descending" file indicates the fraction of transcripts lying above the threshold.
- `<run-id>_quant_usage.csv`: A CSV file containing a single row, with a field for each resource usage statistic (see *Resource usage statistics* above) calculated over the commands used during quantification. CSV fields are also present describing the quantification tool and sequencing parameters used.
- `<run-id>_prequant_usage.csv`: A corresponding CSV file containing resource usage statistics calculated over the commands used during prequantification. Note that this file will only exist if prequantification commands (which are executed only once per quantifier) happened to be run in this directory.

Note that neither of the resource usage CSV files will exist if the *piquant* command `prepare_quant_dirs` was run with the `--nousage` option.

Plots

- `<run-id>_transcript_TPMs_log10_scatter.pdf`: A scatter plot of log-transformed (base 10) estimated against real transcript abundances measured in transcripts per million, for transcripts with non-zero real and estimated abundances.
- `<run-id>_gene__TPMs_log10_scatter.pdf`: A scatter plot of log-transformed (base 10) estimated against real gene abundances measured in transcripts per million, for genes with non-zero real and estimated abundances.

- <run-id>_<statistic>_by_<classifier>.pdf: For each "grouped" transcript classifier, and each statistic marked as being suitable for producing graphs (see *Statistics* above), a plot is created showing the value of that statistic for each group of transcripts determined by the classifier.
- <run-id>_<classifier>_expressed_TPMs_boxplot.pdf: A boxplot is created for each "grouped" transcript classifier showing, for each group of transcripts determined by the classifier, the characteristics of the distribution of log (base 10) ratios of estimated to real transcript abundances for transcripts within that group with non-zero real and estimated abundances.
- <run-id>_<classifier>_expressed_TPMs_<asc|desc>_distribution.pdf: Two plots are drawn for each "distribution" transcript classifier. These correspond to the data in the CSV files described above for these classifiers, and show, for expressed transcripts, the cumulative distribution of the fraction of transcripts lying below or above the threshold determined by the classifier.

Assessment of multiple quantification runs

Statistics and plots comparing multiple quantification runs are produced by executing the *piquant* command analyse_runs (see *Analyse quantification results*). Note that depending on the number of combination of quantification and read simulation parameters that analyse_runs is executed for, a very large number of graphs may be produced; it may, therefore, be useful to concentrate attention on those parameter values which are of greatest interest.

The following CSV files and plots (written as PDF files by default) are produced:

CSV files

- overall_transcript_stats.csv: A CSV file with a field for each defined statistic which has been calculated over the whole set of input transcripts for each quantification run. This data is concatenated from the individual per-quantification run <run-id>_transcript_stats.csv files described above.
- overall_gene_stats.csv: A corresponding CSV file with a field for each defined statistic which has been calculated over the whole set of input genes for each quantification run. This data is concatenated from the individual per-quantification run <run-id>_gene_stats.csv files described above.
- overall_transcript_stats_by_<classifier>.csv: A CSV file for each "grouped" transcript classifier, containing the same fields as overall_transcript_stats.csv, with statistics calculated for distinct subsets of transcripts as determined by the classifier, for each quantification run. This data is concatenated from the individual per-quantification run <run-id>_transcript_stats_by_<classifier>.csv files described above.
- overall_transcript_distribution_stats_<asc|desc>_by_<classifier>.csv: Two CSV files ("ascending" and "descending") for each "distribution" transcript classifier, indicating the fraction of transcripts lying above or below values of the classifier threshold variable, for each quantification run. This data is concatenated from the individual per-quantification run <run-id>_transcript_distribution_stats_<asc|desc>_by_<classifier>.csv files.
- overall_quant_usage.csv: A CSV file with a field for each resource usage statistic which has been calculated for each quantification run. This data is concatenated from the individual per-quantification run <run-id>_quant_usage.csv files described above.
- overall_prequant_usage.csv: A CSV file with a field for each resource usage statistic which has been calculated when prequantification steps were run for each quantifier. This data is concatenated from the individual per-quantifier <run-id>_prequant_usage.csv files described above.

Note that neither of the resource usage CSV files will exist if the *piquant* command analyse_runs was run with the --nousage option.

Plots

Plots produced by the `analyse_runs` commands fall into four categories: *"Overall statistics" graphs*

In the sub-directory `overall_transcript_stats_graphs`, a sub-directory `per_<parameter_1>` is created for each quantification and simulation parameter for which quantification runs were performed for more than one value of that parameter (for example, for read lengths of 35, 50 and 100 base pairs, or for single- and paired-end reads). A boxplot is produced in this directory for each graphable statistic, showing the distribution of values of that statistic over all quantification runs which share each different value of *parameter 1*:

```
overall_<statistic>_per_<parameter_1>.pdf
```

Also within each `per_<parameter_1>` directory, a further `by_<parameter_2>` directory is created for each quantification and simulation parameter for which quantification runs were performed for more than one value of that second parameter (excluding *parameter 1* itself). Within each `by_<parameter_2>` directory, a `<statistic>` directory is created for each statistic marked as capable of producing graphs.

Within each statistic directory, a boxplot is produced showing the distribution of values of that statistic over all quantification runs which share each different value of *parameter 1*, but further grouped into those quantification runs which share each different value of *parameter 2*:

```
overall_<statistic>_per_<parameter_2>_per_<parameter_1>.pdf
```

Furthermore, in the case that *parameter 2* takes numerical values (for example, read length or read depth) graphs are written which plot statistics on the y-axis against values of *parameter 2* on the x-axis; a separate coloured line is shown on these graphs for each value of *parameter 1*. A plot will be produced for every combination of values of quantification and read simulation parameters, excluding *parameter 1* and *parameter 2*:

```
overall_<statistic>_vs_<numerical_parameter_2>_per_<parameter_1>_<other_parameter_
↪values>.pdf
```

So, for example, the directory `overall_transcript_stats_graphs/quant_method` will contain a boxplot for each graphable statistic, showing the distribution of values of that statistic over all quantification runs sharing the same quantification method. Then, each statistic's directory below, say, `overall_transcript_stats_graphs/quant_method/by_read_depth/` will contain, firstly, a boxplot of the distribution of values of that statistic over all quantification runs which share the same quantification method, further grouped into those runs which share the same read depth. Secondly, the directory will contain a plot of that statistic on the y-axis, against read depth on the x-axis, with a line for each quantification method, for each combination of read length, single- or paired-end reads, etc. as specified by the `analyse_runs` command that was executed.

The sub-directory `overall_gene_stats_graphs` is structured in the same way as the `overall_transcript_stats_graphs` directory, but contains graphs of statistics plotted at the level of gene, rather than transcript, TPMs.

"Grouped statistics" graphs

In the sub-directory `grouped_stats_graphs`, a sub-directory `grouped_by_<classifier>` is created for each "grouped" transcript classifier. Graphs written below this directory will plot statistics calculated for groups of transcripts determined by that classifier. Firstly, a boxplot is produced for each graphable statistic, showing the distribution of values of that statistic for each group of transcripts determined by the classifier over all quantification runs:

```
grouped_<statistic>_per_<classifier>.pdf
```

Also within each `grouped_by_<classifier>` directory, a sub-directory `per_<parameter>` is created for each quantification and simulation parameter for which quantification runs were performed for more than one value of

that parameter. Within each `per_<parameter>` directory, a `<statistic>` directory is created for each statistic marked as capable of producing graphs.

Within each statistic directory, a boxplot is produced showing the distribution of values of that statistic for each group of transcripts determined by the classifier over all quantification runs, but further grouped into those runs which share each different value of *parameter*:

```
grouped_<statistic>_per_<parameter>_per_<classifier>.pdf
```

In addition, a complementary boxplot shows the distribution of values of the statistic grouped into those runs which share each different value of *parameter*, and then secondarily grouped according to the transcript classifier:

```
grouped_<statistic>_per_<classifier>_per_<parameter>.pdf
```

Furthermore, a set of graphs are written which plot statistics with a separate, coloured line for each value of *parameter*:

```
grouped_<statistic>_vs_<classifier>_per_<parameter>_<other_parameter_values>.pdf
```

A plot will be produced for every combination of values of quantification and read simulation parameters, excluding the "per" parameter described above. For example, the `sensitivity` directory below `grouped_stats_graphs/grouped_by_transcript_length/per_read_length` will contain a plot of sensitivity on the y-axis, against transcript length on the x-axis, with a line for each simulated read length, for each combination of quantification method, read depth, etc. as specified by the `analyse_runs` command that was executed.

"Distribution statistics" graphs

In the sub-directory `distribution_stats_graphs`, a sub-directory `<classifier>_distribution` is created for each "distribution" transcript classifier. Graphs written below this directory will plot the cumulative distribution of the fraction of transcript lying below or above values of the threshold determined by the classifier.

Within each `<classifier>_distribution` directory, a sub-directory `per_<parameter>` is created for each quantification and simulation parameter for which quantification runs were performed for more than one value of that parameter. Graphs written into this directory will plot statistics with a separate, coloured line for each value of that parameter, and will be named:

```
distribution_<classifier>_per_<parameter>_<asc|desc>_<other_parameter_values>.pdf
```

As before, a plot will be produced for every combination of values of quantification and read simulation parameters, excluding the "per" parameter.

"Resource usage statistic" graphs

In the sub-directory `resource_usage_graphs`, a directory structure is created in exactly the same way as for "Overall statistics" graphs (see *above*). However, in this case, the graphs plotted measure resource usage statistics rather the than accuracy statistics calculated over sets of transcripts or genes.

The `resource_usage_graphs` directory also contains, at the top level, two graphs pertaining to prequantification: `prequant_time_usage.pdf` is a bar plot comparing the real, user and kernel mode time taken by prequantification for each quantification method, and `prequant_memory_usage.pdf` is a bar plot comparing the maximum resident memory occupied by any process during prequantification.

CHAPTER 9

Typical pipeline usage

To illustrate usage of the *piquant* pipeline, we'll compare the accuracy of transcript quantification of the *eXpress* *[eXpress]* and *Sailfish* *[Sailfish]* tools:

- for two read lengths, 50 and 100 base pairs
- for two sequencing depths, 10x and 30x coverage
- for both single- and paired-end reads
- unstranded reads will be simulated with errors and sequencing bias, and with no reads arising from "noise" transcripts

The input genome sequence and transcript definitions will be for the human genome, as defined in Ensembl release 75 *[Ensembl]*.

1. Create output directory and write parameters file

We write an options file containing command line options common to the *piquant* commands we will subsequently execute:

```
--quant-method Express,Sailfish
--read-length 50,100
--read-depth 10,30
--paired-end False,True
--errors True
--bias True
--stranded False
--noise-perc 0
--transcript-gtf ~/data/genome/human/ensembl-75/Homo_sapiens.GRCh37.75.gtf
--genome-fasta ~/data/genome/human/ensembl-75/genome-fa-per-chromosome/
```

Note: The indicated genome FASTA and transcript GTF files have here been downloaded from Ensembl. The

transcript GTF file has been filtered to only contain features of feature type "exon".

2. Prepare read directories

Prepare the directories in which RNA-seq reads will subsequently be simulated:

```
piquant prepare_read_dirs --options-file=piquant_options.txt
```

The default parent output directory for read simulation directories (`output`) is created, and eight read directories are written into it:

- `10x_50b_se_errors_unstranded_bias_no_noise`: i.e. 10x sequencing depth, 50 base-pairs read length, single-end reads
- `10x_50b_pe_errors_unstranded_bias_no_noise`: i.e. 10x sequencing depth, 50 base-pairs read length, paired-end reads
- `10x_100b_se_errors_unstranded_bias_no_noise`: i.e. 10x sequencing depth, 100 base-pairs read length, single-end reads
- `10x_100b_pe_errors_unstranded_bias_no_noise`: i.e. 10x sequencing depth, 100 base-pairs read length, paired-end reads
- `30x_50b_se_errors_unstranded_bias_no_noise`: i.e. 30x sequencing depth, 50 base-pairs read length, single-end reads
- `30x_50b_pe_errors_unstranded_bias_no_noise`: i.e. 30x sequencing depth, 50 base-pairs read length, paired-end reads
- `30x_100b_se_errors_unstranded_bias_no_noise`: i.e. 30x sequencing depth, 100 base-pairs read length, single-end reads
- `30x_100b_pe_errors_unstranded_bias_no_noise`: i.e. 30x sequencing depth, 100 base-pairs read length, paired-end reads

3. Create reads

We're now ready to simulate RNA-seq reads for our chosen sets of sequencing parameters. Note that the number of experiments that can be simulated at the same time will depend on the memory and processing capabilities of the hardware on which *piquant* is run. Here we'll assume we only have enough memory and processing power available to simulate four experiments at a time; hence we'll execute the following pair of commands to simulate reads for each sequencing depth, allowing all *FluxSimulator* processes to terminate in the first case before initiating the next batch of simulations:

```
piquant create_reads --options-file=piquant_options.txt --read-depth=10
piquant create_reads --options-file=piquant_options.txt --read-depth=30
```

4. Check reads

The *piquant* commmand `check_reads` can be used to confirm that read simulation completed successfully:

```
piquant check_reads --options-file=piquant_options.txt
```

A message is output to standard error for each RNA-seq experiment simulation which failed to successfully complete; success in all cases is indicated by no output from the `check_reads` command.

5. Prepare quantification directories

Prepare the directories in which transcript quantification will be performed:

```
piquant prepare_quant_dirs --options-file=piquant_options.txt
```

In this case, sixteen quantification directories are written into the default parent output directory `output` - one for each combination of the eight RNA-seq experiments simulated and the two quantification tools.

6. Perform prequantification steps

Prequantification steps appropriate to the *eXpress* and *Sailfish* tools (and for subsequent analysis of quantification accuracy) are performed using the *piquant* command `prequantify`:

```
piquant prequantify --options-file=piquant_options.txt
```

In this case, the tasks performed are:

- Construction of sequences for transcripts from the input transcript reference GTF file and genome sequence FASTA files.
- Creation of a *Sailfish* kmer index for the transcripts
- Calculation of the number of isoforms for each gene defined in the input transcript reference (see *Count transcripts for genes*).
- Calculation of the unique sequence percentage for each transcript (see *Calculate unique transcript sequence*).

7. Quantify transcripts

We can now run our chosen transcriptome quantification tools on each set of simulated RNA-seq data. As in the case when simulating reads, the number of tool instances that can be run simultaneously will depend on the memory and processing capabilities of the hardware on which *piquant* is run. We'll assume that we only have enough resource available to run four quantification tool instances at a time; hence we'll execute the following four commands to run *eXpress* and *Sailfish* on our single-end and paired-end RNA-seq data sets, allowing all processes to terminate in each case before initiating the next batch of quantifications:

```
piquant quantify --options-file=piquant_options.txt --quant-method=Express --paired-
↪end=False
piquant quantify --options-file=piquant_options.txt --quant-method=Express --paired-
↪end=True
piquant quantify --options-file=piquant_options.txt --quant-method=Sailfish --paired-
↪end=False
piquant quantify --options-file=piquant_options.txt --quant-method=Sailfish --paired-
↪end=True
```

8. Check quantification

The *piquant* command `check_quant` can be used to confirm that quantification completed successfully:

```
piquant check_quant --options-file=piquant_options.txt
```

A message is output to standard error for each quantification run which failed to successfully complete; success in all cases is indicated by no output from the `check_quant` command.

9. Analyse quantification runs

Finally, statistics and graphs describing the accuracy of transcript quantification can be produced via the *piquant* command `analyse_runs`:

```
piquant analyse_runs --options-file=piquant_options.txt
```

In this case statistics and graphs are written into the default analysis output directory `output/analysis` (which is also created, if it does not exist).

CHAPTER 10

Support scripts

RNA-seq read simulation, transcript quantification, and abundance estimate accuracy analysis performed via commands of the `piquant` script are supported by a number of supplementary Python scripts. These are normally executed when running a `run_simulation.sh` or `run_quantification.sh` shell script; however, if necessary, they can also be run independently.

Further information on each script and their command line options is given in the sections below. Note first, that all scripts share the following common command line option:

- `--log-level`: One of the strings "debug", "info", "warning", "error" or "critical" (default "info"), determining the maximum severity level at which log messages will be written to standard error.

Analyse a single quantification run

`analyse_quantification_run` is executed when a `run_quantification.sh` script is run with the `-a` flag. It reads the `tpms.csv` file produced by `assemble_quantification_data` (see *below*), and then calculates statistics and plots graphs to assess the accuracy of transcript abundance estimates produced in a single quantification run.

For full details of the analyses produced, see *here*.

Usage:

```
analyse_quantification_run
    [--log-level=<log-level> --plot-format=<plot-format>]
    [--grouped-threshold=<grouped-threshold>]
    [--error-fraction-threshold=<ef-threshold>]
    [--not-present-cutoff=<cutoff>]
    [--prequant-usage-file=<prequant-usage-file>]
    [--quant-usage-file=<quant-usage-file>]
    --quant-method=<quant-method> --read-length=<read-length>
    --read-depth=<read-depth> --paired-end=<paired-end>
    --errors=<errors> --bias=<bias> --stranded=<stranded>
    --noise-perc=<noise-depth-percentage>
    <tpm-file> <out-file>
```

The following command-line options and positional arguments are required:

- `--quant-method`: The quantification method by which transcript abundance estimates were produced.
- `--read-length`: An integer, the length of reads in the simulated RNA-seq data.
- `--read-depth`: An integer, the depth of sequencing in the simulated RNA-seq data.
- `--paired-end`: A boolean, `True` if the simulated RNA-seq data consists of paired-end reads, or `False` if it consists of single-end reads.
- `--errors`: A boolean, `True` if the simulated RNA-seq data contains sequencing errors.
- `--bias`: A boolean, `True` if sequence bias has been applied to the simulated RNA-seq data.
- `--stranded`: A boolean, `True` if the simulated reads were stranded.
- `--noise-perc`: An integer, the depth of sequencing of "noise" transcripts in the simulated RNA-seq data, as a percentage of the depth of sequencing of the main transcript set.
- `<tpm-file>`: A CSV file describing the per-transcript abundance estimates produced by a quantification run.
- `<out-file>`: A prefix for output CSV and graph files written by this script.

while these command-line parameters are optional:

- `--plot-format`: Output format for graphs, one of "pdf", "svg" or "png" (default "pdf").
- `--grouped-threshold`: The minimum number of transcripts required, in a group determined by a transcript classifier, for a statistic calculated for that group to be shown on a plot (default: 300).
- `--error-fraction-threshold`: Transcripts whose estimated TPM is greater than this percentage higher or lower than their real TPM are considered above threshold for the "error fraction" statistic.
- `--not-present-cutoff`: This cut-off value for a transcript's TPM is used to determined whether the transcript is considered to be present or not.
- `--prequant-usage-file`: A CSV file containing per-prequantification command resource usage statistics recorded using the GNU `time` command.
- `--quant-usage-file`: A CSV file containing per-quantification command resource usage statistics recorded using the GNU `time` command.

Assemble data for a single quantification run

`assemble_quantification_data` is also executed when a `run_quantification.sh` script is run with the `-a` flag. It assembles data required to assess the accuracy of transcript abundance estimates produced in a single quantification run, and writes these data to an output CSV file. See *here* for full details of the data sources and output file contents.

Usage:

```
assemble_quantification_data
    [--log-level=<log-level>]
    --method=<quantification-method> --out=<output-file>
    <pro-file> <transcript-count-file> <unique-sequence-file>
```

The following command-line options and positional arguments are required:

- `--method`: The quantification method by which transcript abundance estimates were produced.

- `--out`: The output CSV file name.
- `<pro-file>`: Full path of the *FluxSimulator* *[FluxSimulator]* expression profile file which contains 'ground truth' transcript abundances.
- `<transcript-count-file>`: Full path of a file containing per-gene transcript counts, as produced by *the script* `count_transcripts_for_genes`.
- `<unique-sequence-file>`: Full path of a file containing lengths of sequence unique to each transcript, as produced by *the script* `calculate_unique_transcript_sequence`.

Calculate reads required for sequencing depth

`calculate_reads_for_depth` is run when a `run_simulation.sh` script is executed. It calculates the approximate number of reads required to be simulated for a set of transcripts in order to provide the specified sequencing depth, given a particular length of read.

Usage:

```
calculate_reads_for_depth
    [--log-level=<log-level>]
    <pro-file> <read-length> <read-depth>
```

The following positional arguments are required:

- `<pro-file>`: The *FluxSimulator* expression profile file from which reads will be simulated.
- `<read-length>`: An integer, the length of reads in base pairs.
- `<read-depth>`: An integer, the mean sequencing depth desired.

Calculate unique transcript sequence

`calculate_unique_transcript_sequence` is executed when a `run_quantification.sh` script is run with the `-p` flag. It calculates the length of sequence in base pairs that is unique to each transcript from which reads will be simulated.

Usage:

```
calculate_unique_transcript_sequence
    [--log-level=<log-level>]
    <gtf-file>
```

The following positional argument is required:

- `<gtf-file>`: Full path to the GTF file defining transcripts and genes.

Count transcripts for genes

`count_transcripts_for_genes` is also executed when a `run_quantification.sh` script is run with the `-p` flag. It calculates the number of transcripts shared by the gene of origin for each transcript from which reads will be simulated.

Usage:

```
count_transcripts_for_genes
    [--log-level=<log-level>]
    <gtf-file>
```

The following positional argument is required:

- `<gtf-file>`: Full path to the GTF file defining transcripts and genes.

Fix antisense reads

`fix_antisense_reads` is run when a `run_simulation.sh` script is executed and stranded single-end reads are being simulated. In this case, the reads produced by *FluxSimulator* correspond to both the sense and antisense strands. Those reads in the input FASTA or FASTQ file corresponding to the antisense strand are reverse complemented.

Usage:

```
fix_antisense_reads
    [--log-level=<log-level> --out-prefix=<out-prefix>]
    <reads-file>
```

The following positional argument is required:

- `<reads-file>`: A FASTA or FASTQ file containing single-end reads for which antisense reads are to be switched to the sense strand.

while the following command-line option is optional:

- `--out-prefix`: String to be prepended to the input file name to form the output file name [default: "sense"].

Randomise read strands

`randomise_read_strands` is run when a `run_simulation.sh` script is executed and unstranded paired-end reads are being simulated. In this case, the reads produced by *FluxSimulator* effectively originate from the sense strand. The script randomly reassigns pairs of paired-end reads in the input FASTA or FASTQ file such that the first read no corresponds to the antisense strand.

Usage:

```
randomise_read_strands
    [--log-level=<log-level> --out-prefix=<out-prefix>]
    <reads-file>
```

The following positional argument is required:

- `<reads-file>`: A FASTA or FASTQ file containing paired-end reads for which read pairs strands are to be randomly reassigned.

while the following command-line option is optional:

- `--out-prefix`: String to be prepended to the input file name to form the output file name [default: "unstranded"].

Simulate sequence bias in reads

`simulate_read_bias` is run when a `run_simulation.sh` script is executed. It approximates a particular type of sequence bias by preferentially selecting reads from an input FASTA or FASTQ file the beginning of whose sequence is closer to having a specified nucleotide composition.

Usage:

```
simulate_read_bias
    [--log-level=<log-level>  --out-prefix=<out-prefix>  --paired-end]
    --num-reads=<num-reads>
    <pwm-file> <reads_file>
```

The following command-line options and positional arguments are required:

- `--num-reads`: Number of reads to output.
- `<pwm-file>`: Full path to a file containing a position weight matrix; this PWM defines a preferential nucleotide composition for bases at the start of reads. Reads whose starting sequence composition scores higher against this PWM are more likely to be selected for output.
- `<reads-file>`: FASTA or FASTQ file containing reads upon which bias is to be imposed.

while these command-line parameters are optional:

- `--out-prefix`: Prefix for FASTA or FASTQ file to which biased reads are written (default "bias").
- `--paired-end`: Indicates the reads file contains paired-end reads.

CHAPTER 11

Extending piquant

piquant is extensible in three principal ways:

- *Adding a new quantifier*: Adding an additional quantification tool or pipeline whose comparative performance against other quantifiers can then be assessed.
- *Adding a new statistic*: Adding an additional statistic to be calculated for each quantification run.
- *Adding a new transcript classifier*: Adding an additional classifier to split transcripts into discrete groups, so that the performance of quantification tools can be assessed across these sets of transcripts.

All three methods of extension currently require some coding in Python.

Note: It is also possible to extend *piquant* to add a new sequencing parameter (e.g. read depth, read length, errors, bias etc), over different values of which reads can be simulated. Doing this, however, is more involved - please contact the author for further information if you wish to do this.

Adding a new quantifier

To enable *piquant* to run a particular quantification tool or pipeline, a new class should be added to the Python module `quantifiers.py`, marked with the decorator `@_quantifier`, and fulfilling the API requirements detailed below. Any such tool will then be automatically available to be included in quantification runs from the *piquant* command line.

A quantifier class has three main responsibilities:

- It must supply commands to be written to `run_quantification.sh` scripts that will be executed when the scripts are run with the command line flag `-p`; that is, preparatory actions that must be taken prior to quantifying transcripts with this quantification tool, but that only need to be executed once for a particular set of input transcripts and genome sequences.
- It must supply commands to be written to `run_quantification.sh` scripts that will be executed when the scripts are run with the command line flag `-q`; that is, actions that must be taken to calculate transcript abundances with this quantification tool for a particular set of simulated reads.

- It must be able to return the abundance calculated by the quantification tool for a specified transcript.

In detail, in addition to being marked with the decorator `@_quantifier`, a quantifier class must implement the following methods:

get_name()

`get_name` should return the string to be given when specifying a list of quantifiers to be used by *piquant* via the command-line or parameters file option `--quant-method`.

write_preparatory_commands(*writer*, *params*)

`write_preparatory_commands` writes commands to a `run_quantification.sh` script that should be executed prior to quantifying transcripts with the particular quantification tool, but that only need to be executed once for a particular set of input transcripts and genome sequences - for example, preparing a *Bowtie* index for the genome, or constructing transcript sequences.

Commands are written via the `writer` parameter, an instance of the `BashScriptWriter` class (see *below*), which facilitates writing to a Bash script.

`params` is a dictionary of key-value pairs containing items that may be of use to the quantifier during preparation or subsequent quantification:

- `TRANSCRIPT_GTF_FILE`: Full path to the GTF file containing transcript definitions.
- `GENOME_FASTA_DIR`: Full path to the directory containing genome sequence FASTA files.
- `QUANTIFIER_DIRECTORY`: Full path to a directory `quantifier_scratch`, created within the *piquant* output directory, that quantifiers can write files to necessary for their operation which only need to be created once (for example, a *Bowtie* or *Sailfish* index).
- `NUM_THREADS`: The maximum number of threads to be used by any multithreaded program that is to be executed.
- `FASTQ_READS`: A boolean, `True` if reads have been simulated with errors (and hence quality values), and are thus written in a FASTQ file.
- `STRANDED_READS`: A boolean, `True` if reads have been simulated as coming from a stranded protocol.
- `SIMULATED_READS`: If single-end reads are being quantified, the full path to the file containing simulated reads. This key is not present in the dictionary if paired-end reads are being quantified.
- `LEFT_SIMULATED_READS`: If paired-end reads are being quantified, the full path to the file containing the first read for each pair of simulated reads. This key is not present in the dictionary if single-end reads are being quantified.
- `RIGHT_SIMULATED_READS`: If paired-end reads are being quantified, the full path to the file containing the second read for each pair of simulated reads. This key is not present in the dictionary if single-end reads are being quantified.

write_quantification_commands(*writer*, *params*)

`write_quantification_commands` writes commands to a `run_quantification.sh` that will be executed to calculate transcript abundances with this quantification tool for a particular set of simulated reads.

Commands are again written via the `writer` parameter, an instance of the `BashScriptWriter` class. `params` is a dictionary of key-value pairs containing the same items as described for `write_preparatory_commands` *above*.

write_cleanup(*writer*)

Running a quantification tool may produce many files in addition to that needed to assess the tool's performance (i.e. the file containing estimated transcript abundances), and if multiple quantification runs are performed, these may occupy significant disk space. `write_cleanup` allows an opportunity for commands to be writen to remove these

files once quantification has been performed. As before, such commands can be written via the `writer` parameter, an instance of the `BashScriptWriter` class.

`get_transcript_abundance`(*transcript_id*)

`get_transcript_abundance` should return the transcript abundance estimated by the quantification tool for the transcript specified by the parameter `transcript_id`; as this method will be called for each transcript in the input set, it should generally read transcript abundances from the output files of the quantification tool only once. Transcript abundances should be returned in units of TPM (transcripts per million). If the quantification tool does not supply abundance estimates in TPM, a transformation to these units may require to be perfomed (for example, see `_Cufflinks.get_transcript_abundance()`, which transforms the FPKM values output by Cufflinks into TPM).

The BashScriptWriter class

`BashScriptWriter` is a simple utility class to facilitate the writing of commands by quantifier classes to *piquant*'s `run_simulation.sh` and `run_quantification.sh` scripts. The most common methods are:

`add_line`(*line_string*)

The command specified by the parameter `line_string` will be written to the script at the appropriate indendation level.

`section`()

To be used in a Python `with` statement. Commands, comments etc. added within this context will be grouped together in the Bash script, followed by a blank line.

`if_block`(*test_command*)

To be used in a Python `with` statement. Commands, comments etc. added within this context will be grouped together within a Bash `if/then/fi` block. The parameter `test_command` specifies the condition to be tested within the `if` statement.

`add_echo`(*text*)

An echo statement will be written to the Bash script to print the string specified by the parameter `text`.

`add_pipe`([*pipe_commands*])

The commands specified by the function's parameters will be joined together by pipes and written to the Bash script.

`add_comment`(*comment*)

The text specified by the parameter `comment` will be written to the Bash script as an appropriately-formatted comment.

Note: An exception to the use of BashScriptWriter is in the case where commands are being written that should contribute to the resource usage statistics recorded during prequantification or quantification. In this case, the methods `_add_timed_{pre}quantification_<command|pipe>` of the `_BaseQuantifier` class should be used instead.

Adding a new statistic

To add a new statistic, a class should be added to the Python module `statistics.py`, marked with the decorator `@_statistic`, and fulfilling the API requirements detailed below. Any such statistic will be automatically included

in the post-quantification analysis performed by *piquant*: graphs will be produced showing the variation of the statistic as measured for different quantification tools as sequencing parameters and transcript classification measures change.

A statistics class must have the following attributes and methods (note that the attributes can most easily be provided by extending the class `_BaseStatistic`):

`name`

A short name for the statistic, to be used in filenames and CSV column headers.

`title`

A human-readable description for the statistic, to appear in graph titles and axis labels.

`graphable`

A boolean, `True` if graphs of the statistic should be plotted as part of *piquant*'s analysis.

`calculate`(*tpms*, *expressed_tpms*)

`calculate` should compute the statistic for a set of transcript abundances estimated by a particular quantification tool. The parameter `tpms` is a pandas DataFrame describing the results of a quantification run, while `expressed_tpms` is a DataFrame describing those results of the quantification run for which real transcript abundances were above the threshold value indicating "presence" of the transcript.

The `tpms` and `expressed_tpms` DataFrame objects have a row for each estimated transcript abundance, and the following columns:

- `transcript`: Transcript identifier as specified in the input transcripts GTF file.
- `length`: Transcript length in base pairs.
- `unique-length`: Length in base pairs of transcript sequence which does not overlap with the exons of any other transcript.
- `num-transcripts`: Number of isoforms for this transcript's originating gene.
- `real-tpm`: Ground-truth transcript abundance used to produce the simulated RNA-seq data set, measured in transcripts per million.
- `calc-tpm`: Transcript abundance estimated by the quantification tool, measured in transcripts per million.

`calculate` should return a single number, the computed statistic.

`calculate_grouped`(*grouped*, *grp_summary*, *expressed_grouped*, *expressed_grp_summary*)

`calculate_grouped` should compute a set of statistic values for the results of a quantification run which have been grouped according to a certain method of classifying transcripts. The parameter `grouped` is a pandas GroupBy instance, describing the results of a quantification run grouped by the transcript classifier; `group_summary` is a DataFrame containing basic summary statistics calculated for each group of transcripts. The parameters `expressed_grouped` and `expressed_grp_summary` are analogous to the first two parameters, but describe only results of the quantification run for which real transcript abundances were above the threshold value indiciated "presence" of the transcript.

`calculate_grouped` should return a pandas Series instance, enumerating the statistic as calculated for each transcript group. When adding a new statistic, it may be easiest to adapt one of the existing `calcualate_grouped` methods to your needs.

`stat_range(vals_range)`:

The `stat_range` method controls the y-axis bounds in graphs created for this statistic. The `vals_range` parameter is a tuple of two values, the minimum and maximum values of the statistic that will be plotted in a particular graph. `stat_range` should return either a tuple of two values or `None`.

If a tuple is returned, each value should either be a number or `None`. The first value will be the minimum bound of the y-axis in the graph to be drawn; a value of `None` indicates that no special bound is to be imposed and the y-axis minimum will be chosen automatically according to the minimum value of the statistic. Likewise, the second value controls the maximum bound of the y-axis. Returning `None` instead of a tuple means that both y-axis bounds will be chosen automatically.

Adding a new transcript classifier

Adding a new classifier of transcripts is perhaps simpler than adding a new quantification tool or analysis statistic; in the Python module `classifiers.py`, an instance of the class `_Classifier` should be added to the list `_CLASSIFIERS`. Any such classifier will automatically be included in the post-quantification analysis performed by *piquant*, and graphs will be produced showing the variation of statistics as measured across groups of transcripts as defined by the classifier.

Parameters to be supplied to the `_Classifier` constructor are as follows:

- `column_name`: A short name for the classifier, to be used in filenames and CSV column headers.
- `value_extractor`: A function which takes a row of a pandas DataFrame containing the results of a quantification run (as described *above* - such a row describes quantification for a single transcript) and returns a numeric classification value for the transcript indicated by the row.
- `grouped_stats` *[Optional - default: True]*: A boolean. If `True`, the instance is a *"grouped" classifier*, which splits transcripts into fixed groups dependent on some property inherent in the transcripts (or their estimated abundances) themselves. If `False`, the instance is a *"distribution" classifier*, which splits transcripts into two groups, those above and below some threshold (where that threshold is generally the value of some property of quantification).
- `distribution_plot_range` *[Optional - default: None]*: If `grouped_stats` is `False`, this parameter should either be a tuple of two numbers or `None`. If a tuple is supplied, these should be the minimum and maximum values of the "distribution" classifier threshold to be used in plots produced by this classifier.
- `plot_title` *[Optional - default: None]*: A human-readable description for the classifier, to appear in graph titles and axis labels. If not supplied, the value of the `column_name` parameter will be used.

Note that a subclass, `_LevelsClassifier`, of `_Classifier` is supplied, which aids the construction of classifiers which group transcripts based on ranges of some parameter that takes many possible values (for example, transcript length in base pairs, or transcript abundance measured in TPM). Parameters to be supplied to the `_LevelsClassifier` constructor are as follows:

- `column_name`: As for `_Classifier`.
- `value_extractor`: A function which takes a row of a pandas DataFrame (as described for `_Classifier` above) and extracts a numeric classification value for the transcript indicated by the row. Note, however, that transcripts are classified into groups based on the particular range this values falls into, as determined by the `levels` and `closed` parameters below.
- `levels`: A list of numbers defining the ranges of values (as determined by the `value_extractor` function) for which transcripts are considered to belong to the same group. The first group consists of all transcripts whose value is less than or equal to the first item in `levels`; the second group those transcripts whose value is greater than the first item in `levels` and less than or equal to the second item, and so on. The nature of the final group is determined by the parameter `closed` below.
- `closed` *[Optional - default: False]*: A boolean. If `False`, the final group for the classifier consists of all transcripts whose value (as determined by the `value_extractor` function) is greater than or equal to the last item in `levels`. If `True`, there is no such open range: the final group consists of all transcripts whose value is greater than or equal to the last but one item in `levels`, and less than or equal to the last item.

- `plot_title` *[Optional - default: None]*: As for `_Classifier`.

CHAPTER 12

References

Bibliography

[Bowtie] Langmead et al., Ultrafast and memory-efficient alignment of short DNA sequences to the human genome, Genome Biology (2009). Software homepage.

[Cufflinks] Trapnell et al., Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation, Nature Biotechnology (2010); Roberts et al., Improving RNA-Seq expression estimates by correcting for fragment bias, Genome Biology (2011). Software homepage.

[Ensembl] Flicek et al, Ensembl 2014, Nucleic Acids Research (2014). Ensembl homepage.

[eXpress] Roberts and Pachter, Streaming fragment assignment for real-time analysis of sequencing experiments, Nature Methods (2013). Software homepage.

[FluxSimulator] Griebel et al., Modelling and simulating generic RNA-Seq experiments with the flux simulator, Nucleic Acids Research (2012). Software homepage.

[Hansen] Hansen et al., Biases in Illumina transcriptome sequencing caused by random hexamer priming, Nucleic Acids Research (2010).

[Kallisto] Bray et al., Near-optimal RNA-Seq quantification, arxiv (2015). Software homepage.

[RSEM] Li and Dewey, RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome, BMC Bioinformatics (2011). Software homepage.

[Sailfish] Patro et al., Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms, Nature Biotechnology (2014). Software homepage.

[Salmon] Patro et al., Salmon: Accurate, Versatile and Ultrafast Quantification from RNA-seq Data using Lightweight-Alignment, bioarxiv (2015). Software homepage.

[SAM] Li et al., The Sequence alignment/map (SAM) format and SAMtools. SAMtools homepage.

[TopHat] Kim et al., TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions, Genome Biology (2013). Software homepage.

Index